BrightSign®

OBJECT REFERENCE MANUAL

Firmware Versions 6.0.x

BrightSign, LLC. 16780 Lark Ave., Suite B Los Gatos, CA 95032 | 408-852-9263 | www.brightsign.biz

TABLE OF CONTENTS

INTRODUCTION

BrightSign players use a standardized library of BrightScript objects to expose functionality for public software development. To publish a new API for interacting with BrightSign hardware, we create a new BrightScript object.

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

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

INTERFACES AND METHODS OVERVIEW

Every BrightScript object consists of one or more "interfaces." An interface consists of one or more "methods." For example, the *roVideoPlayer* object has several interfaces, including *ifMessagePort*. The interface *ifMessagePort* has one method: `SetPort()`.

Example: The abstract interface *ifMessagePort* 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.

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

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

The above 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, the following two lines of code carry out the exact same function:

```
gpio.SetPort(p)
gpio.ifMessagePort.SetPort(p)
```

BrightScript Objects consist *only* of interfaces, and 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” or “get” methods in an interface.

Classes

A "class name" is used to create a BrightScript object. For example, the class name for a video playback instance is *roVideoPlayer*, so, to initialize a video playback instance, you would use code similar to the following:

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

Note that “video” can be any name that follows the syntax outlined in the next section.

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.

Depending on the BrightSign model, zones can contain video, images, HTML content, audio, a clock, or text. 4Kx42, XDx32, and XDx30 models can display two video zones on screen, while the HDx22, HDx20, and LSx22 models can only display one. 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.

As of firmware 6.0.x, zone support is enabled by default. When zones are enabled, the image layer is on top of the video layer by default. The default behavior can be modified using the *roVideoMode.SetGraphicsZOrder()* method.

Zone support can be disabled by calling `EnableZoneSupport(false)`. 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 the following structure:

1. Wait for an event.
2. Process the event.
3. Return to step 1.

An event can be any number occurrences: a button has been pressed; a timer has been triggered; a UDP message has been received; a video has finished playing back; etc.

By convention, event scripting for BrightScript objects follows this work flow:

1. An object of the type *roMessagePort* is created by the user's script.
2. Objects that can send events (i.e. those that support the *ifMessagePort/ifSetMessagePort* interface) are instructed to send their events to this message port using the `SetPort()` method. You can set up multiple message ports and have each event go to its own message port, but it is usually simpler to create one message port and have all the events sent to this one port.
3. The script waits for an event. The actual function to do this is *ifMessagePort.WaitMessage()*, but the built-in `Wait()` statement in BrightScript allows you to do this more easily.
4. If multiple event types are possible, your script should determine which event the wait function 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 receive and be able to process them.

BrightSign Object Library

The following chapters provide definitions for 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 for functionality.

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: The *roArray* object is created with two parameters.

`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.

The `dim` statement may be used instead of the `CreateObject` function to create a new array. The `dim` statement is sometimes advantageous because it automatically creates array-of-array structures 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) that 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 this method will return Invalid.
- `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 for converting strings to or from a byte array, as well as to or from ASCII hex or ASCII base64. 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.

The byte array will automatically resize to become larger as needed. If you wish to disable this behavior, use the `SetResize()` method. 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(file_path As String) As Boolean`: Writes the bytes contained in the byte array to the specified file. This method returns True if successful.
- `WriteFile(file_path As String, start_index As Integer, length As Integer) As Boolean`: Writes a subset of the bytes contained in the byte array to the specified file. This method writes `length` bytes, beginning at `start_index` of the byte array.
- `ReadFile(file_path As String) As Boolean`: Reads the specified file into the byte array. This operation will discard any data currently contained in the byte array.

- `ReadFile(file_path As String, start_index As Integer, length As Integer) As Boolean`: Reads a section of the specified file into the byte array. This method reads `length` bytes, beginning at `start_index` of the file. This operation will discard any data currently contained in the byte array.
- `AppendFile(file_path As String) As Boolean`: Appends the contents of the byte array to the specified file.
- `SetResize(minimum_allocation_size As Integer, autoresize As Boolean)`: Expands the size of the byte array to the `minimum_allocation_size` if it is less than the `minimum_allocation_size`. This method also accepts a Boolean parameter that specifies whether the byte array should be resized automatically or not.
- `ToHexString() As String`: Returns a hexadecimal string representation of the contents of the byte array. Each byte is represented as two hex digits.
- `FromHexString(hex_string As String)`: Writes the contents of the passed hexadecimal string to the byte array. The passed string must contain an even number of hex digits. This operation will discard any data currently contained in the byte array.
- `ToBase64String() As String`: Returns the contents of the byte array as a base64-formatted string.
- `FromBase64String(base_64_string As String)`: Writes the contents of a valid base64-formatted string to the byte array. This operation will discard any data currently contained in the byte array.
- `ToAsciiString() As String`: Returns the contents of the byte array as an ASCII-formatted string.
- `FromAsciiString(a As String)`: Writes the contents of a valid ASCII-formatted string to the byte array. This operation will discard any data currently contained in the byte array.
- `GetSignedByte(index As Integer) As Integer`: Returns the signed byte at the specified zero-based index in the byte array. To read an unsigned byte within a byte array, use the *ifArrayGet.GetEntry()* method or the `[]` array operator.
- `GetSignedLong(index As Integer) As Integer`: Retrieves the integer located at the specified long-word index of the byte array. Note that this method cannot accept a byte index as its parameter.
- `IsLittleEndianCPU() As Boolean`: Returns True if the CPU architecture is little-endian.

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

This object provides a set of standard, module-scope functions that are stored in the global object. If one of these global functions is referenced, the compiler directs the runtime to call the appropriate global object member.

Note: *Global trigonometric functions accept and return values in radians, not degrees.*

Interfaces: *ifGlobal*

The *ifGlobal* interface provides the following:

- `CreateObject(name As String) As Object`: Creates a BrightScript object corresponding to the specified class `name`. This method returns invalid if object creation fails. Some objects have optional parameters in their constructor, which must be passed after the class `name`.
  Example:

```
sw = CreateObject("roGpioControlPort")
serial = CreateObject("roSerialPort", 0, 9600)
```

- `RestartScript()`: Exits the current script. The system then scans for a valid autorun file to run.
- `RestartApplication()`: Restarts the BrightSign application.
- `Sleep(milliseconds As Integer)`: Instructs the script to pause for a specified amount of time without wasting CPU cycles. The sleep interval is specified in milliseconds.
- `asc(letter As String) As Integer`: Returns the ASCII code for the first character of the specified string. A null-string argument will cause an error.
- `chr(character As Integer) As String`: Returns a one-character string containing a character reflected by the specified ASCII or control. For example, because quotation marks are normally used as string delimiters, you can pass ASCII code 34 to this function to add quotes to a string.
- `len(target_string As String) As Integer`: Returns the number of characters in a string.

- `str(value As Double) As String`: Converts a specified float value to a string. This method also returns a string equal to the character representation of a value. For example, if "A" is assigned a value of 58.5, then calling `str(A)` will return "58.5" as a string.
- `strI(value As Integer) As String`: Converts a specified integer value to a string. This method also returns a string equal to the character representation of a value. For example, if "A" is assigned a value of 58.5, then calling `stri(A)` will return "58" as a string.
- `val(target_string As String) As Double`: Returns a number represented by the characters in the string argument. This is the opposite of the `str()` function. For example, if "A" is assigned the string "58", and "B" is assigned the string "5", then calling `val(A+".")+B)` will return the float value 58.5.
- `abs(x As Double) As Double`: Returns the absoule vale of the argument *x*.
- `atn(x As Double) As Double`: Returns the arctangent (in radians) of the argument *x* (i.e. `Atn(x)` returns "the angle whose tangent is *x*"). To get the arctangent in degrees, multiply `Atn(x)` by 57.29578.
- `csng(x As Integer) As Float`: Returns a single-percision float representation of the argument *x*.
- `cdbl(x As Integer) As Double`: Returns a double-percision float representation of the argument *x*.
- `cint(x As Double) As Integer`: Returns an integer representation of the argument *x* by rounding to the nearest whole number.
- `cos(x As Double) As Double`: Returns the cosine of the arugment *x*. The argument must be in radians. To obtain the cosine of *x* when *x* is in degrees, use `Cos(x*.01745329)`.
- `exp(x As Double) As Double`: Returns the natural exponential of *x*. This is the inverse of the `log()` function.
- `fix(x As Double) As Integer`: Returns a truncated representation of the argument *x*. All digits to the right of the decimal point are removed so that the resultant value is an integer. For non-negative values of *x*, `fix(x)` is equal to `int(x)`. For negative values of *x*, `fix(x)` is equal to `int(x)+1`.
- `int(x As Double) As Integer`: Returns an integer representation of the argument *x* using the largest whole number that is not greater than the argument. For example, `int(2.2)` returns 2, while `fix(-2.5)` returns -3.
- `log(x As Double) As Double`: Returns the natural logarithm of the argument *x*(i.e. $log_e(x)$). This is the inverse of the `exp()` function. To find the logarithm of a number to a base *b*, use the following formula*:*

  $log_b(x) = log_e(x)/log_e(b)$.

- `sgn(x As Double) As Integer`: Returns an integer representing how the float argument *x* is signed: -1 for negative, 0 for zero, and 1 for positive.
- `sgnI(x As Integer) As Integer`: Returns an integer representing how the integer argument *x* is signed: -1 for negative, 0 for zero, and 1 for positive.
- `sin(x As Double) As Double`: Returns the sine of the argument *x*. The argument must be in radians. To obtain the sine of *x* when *x* is in degrees, use `sin(x*.01745329)`.
- `tan(x As Double) As Double`: Returns the tangent of the argument *x*. The argument must be in radians. To obtain the tangent of *x* when *x* is in degrees, use `tan(x*.01745329)`.
- `sqr(x As Double) As Double`: Returns the square root of the argument *x*. This function is the same as *x^(1/2)*, but calculates the result faster.
- `Left(target_string As String, n As Integer) As String`: Returns the first *n* characters of the specified string.
- `Right(target_string As String, n As Integer) As String`: Returns the last *n* characters of the specified string.
- `StringI(n As Integer, character As Integer) As String`: Returns a string composed of a character symbol repeated *n* times. The character symbol is passed to the method as an ASCII code integer.
- `String(n As Integer, character As String) As String`: Returns a string composed of a character symbol repeated *n* times. The character symbol is passed to the method as a string.
- `Mid(target_string As String, start_position As Integer, length As Integer) As String`: Returns a substring of the target string. The first integer passed to the method specifies the starting position of the substring, and the second integer specifies the length of the substring. The start position of a string begins with 1.
- `instr(start_position As Integer, search_text As String, substring_to_find As String) As Integer`: Returns the position of a substring within a string. This function is case sensitive and returns 0 if the specified substring is not found. The start position of a string begins with 1.
- `GetInterface(object As Object, ifname As String) As Interface`: Returns a value of the type Interface. All objects have one or more interfaces. In most cases, you can skip interface specification when calling an object component. This will not cause problems as long as the method names within a function are unique.

- `Wait(timeout As Integer, port As Object) As Object`: Instructs the script to wait on an object that has an *ifMessagePort* interface. This method will return the event object that was posted to the message port. If the timeout is specified as zero, `Wait()` will wait indefinitely; otherwise, `Wait()` will return Invalid after the specified number of milliseconds if no messages have been received.

  Example:

```
p =   CreateObject("roMessagePort")
sw =  CreateObject("roGpioControlPort")
sw.SetPort(p)
msg=wait(0, p)
print type(msg)     ' should be roGpioButton
print msg.GetInt()  ' button number
```

- `ReadAsciiFile(file_path As String) As String`: Reads the specified text file and returns it as a string.
- `WriteAsciiFile(file_path As String, buffer As String) As Boolean`: Creates a text file at the specified file path. The text of the file is passed as the second parameter. This method cannot be used to edit files: A preexisting text file will be overwritten if it has the same name and directory path as the one being created.

Note: *The* roCreateFile *object provides more flexibility if you need to create or edit files.*

- `ListDir(path As String) As Object`: Returns an *roList* object containing the contents of the specified directory path. File names are converted to all lowercase.
- `MatchFiles(path As String, pattern_in 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. Each listed result contains only the part of the filename that is matched against the pattern, not the full path. The match is only applied in the specified directory; you will get no results if the pattern contains a directory separator. The pattern is a case insensitive wildmat expression. It may contain the following special characters:
  - `?` -- Matches any single character.
  - `*` -- Matches zero or more arbitrary characters.

    - `[…]`-- Matches any single character specified within the brackets. The closing bracket is treated as a member of the character class if it immediately follows the opening bracket (i.e. "[]]" matches a single closed bracket). Within this class, "-" can be used to specify a range unless it is the first or last character (e.g. "[A-Cf-h"] is equivalent to "[ABCfgh]"). A character class may be negated by specifying "^" as the first character. To match a literal of this character, place it elsewhere in the class.

  Note: *The special characters "?", "*", and "[" lose their function if preceded by a single "\", and a single "\" can be matched using "\\".*

- `LCase(target_string As String) As String`: Converts the specified string to all lower case.
- `UCase(target_string As String) As String`: Converts the specified string to all upper case.
- `DeleteFile(file_path As String) As Boolean`: Deletes the file at the specified file path. This method returns False if the delete operation fails or if the file does not exist.
- `DeleteDirectory(diretory As String) As Boolean`: Deletes the specified directory. This method will recursively delete any files and directories that are necessary for removing the specified directory. This method returns False if it fails to delete the directory, but it may still delete some of the nested files or directories.
- `CreateDirectory(directory As String) As Boolean`: Creates the specified directory. Only one directory can be created at a time. This method returns True upon success and False upon failure.
- `RebootSystem()`: Causes a soft reboot.
- `ShutdownSystem()`
- `UpTime(dummy As Integer) As Double`: Returns the uptime of the system (in seconds) since the last reboot.
- `FormatDrive(drive As String, fs_type As String) As Boolean`: Formats the specified drive using one of the file systems listed below. This function returns True upon success and False upon 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(drive As String) As Boolean`: Ejects the specified drive (e.g. "SD:") and returns True if successful. If the script is currently accessing files from the specified drive, the ejection process will fail.

- `CopyFile(source As String, destination As String) As Boolean`: Copies the file at the specified source file-path name to the specified destination file-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(target_string As String) As Integer`: Converts the target string to an integer. Any non-integer characters (including decimal points and spaces), and any numbers to the right of a non-integer character, will not be part of the integer output.
- `rnd(a As Dynamic) As Dynamic`
- `RunGarbageCollector() As roAssociativeArray`: Destroys objects that are currently in a state of circular reference counting. BrightScript normally removes any objects that become unreferenced as part of its automated garbage collection algorithm. However, objects that reference each other will never reach a reference count of zero, and will need to be destroyed manually using this method. This method is useful when destroying old presentation data structures and generating a new presentation. This method returns an associative array outlining the results of the garbage-collection process.
- `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 method does not fail; however, if the specified default drive does not exist, it will not be possible to retrieve anything.
- `EnableZoneSupport(enable As Boolean)`: Allows for display of multiple video, HTML, image, and text zones. As of firmware 6.0.x, zone support is enabled by default.
- `EnableAudioMixer(a As Boolean)`
- `Pi() As Double`: Returns the value of pi as a double-precision floating-point number.
- `ParseJson(json_string As String) As Object`: Parses a string formatted according to the RFC4627 standard and returns an equivalent BrightScript object, which can consist of the following: Booleans, integers,

floating point numbers, strings, *roArray* objects, and *roAssociativeArray* objects. The `ParseJson()` method has the following properties:

- Invalid will be returned if the string is not syntactically correct.
- Any *roAssociativeArray* objects that are returned will be case sensitive.
- An error will be returned if an *roArray* or *roAssociativeArray* is nested more than 256 levels deep.

Example: The following script demonstrates how to use `ParseJson()` to process a JSON object containing the titles and URLs of a set of images.

JSON Script

```
{
  "photos" : [
          {
                  "title" : "View from the hotel",
                  "url" : "http://example.com/images/00012.jpg"
          },
          {
                  "title" : "Relaxing at the beach",
                  "url" : "http://example.com/images/00222.jpg"
          },
          {
                  "title" : "Flat tire",
                  "url" : "http://example.com/images/00314.jpg"
          }
  ]
}
```

BrightScript

```
searchRequest = CreateObject("roUrlTransfer")
searchRequest.SetURL("http://api.example.com/services/rest/getPhotos")
response = ParseJson(searchRequest.GetToString())
For Each photo In response.photos
        GetImage(photo.title, photo.url)
End For
```

- `FormatJson(json As roAssociativeArray, flags As Integer) As String`: Converts an associative array to a JSON string (i.e. formatted according to the RFC4627 standard). The following are supported data types: Boolean, Integer, Float, String, *roArray*, and *roAssociativeArray*. If the `flags` parameter is set to `0` or not specified, non-ASCII characters are escaped in the output string as "\uXXXX", where "XXXX" is the hexadecimal representation of the Unicode character value. If the `flags` parameter is set to `1`, non-ASCII characters are not escaped. If arrays or associative arrays are nested more than 256 levels deep, an error will occur. If an error occurs, an empty string will be returned.

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*:

- print o: Prints *o.GetInt()*
- i%=o: Assigns the integer i% the value of *o.GetInt()*
- k=o: Presumably k is *typeOmatic*, so it becomes another reference to the *roInt* 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:

Note: *The function indexes of* ifStringOps *methods start at zero, while the function indexes of global methods start at one.*

- `SetString(str As String, str_len As Integer)`: Sets the string using the specified string and string-length values.
- `AppendString(str As String, str_len As Integer)`: Appends the string using the specified string and string-length values. This method modifies itself—this can cause unexpected results when you pass an intrinsic string type, rather than a string object.

  Example:

```
x="string"
x.ifstringops.appendstring("ddd",3)
print x  'will print 'string'

y=box("string")
y.ifstringops.appendstring("ddd",3)
print y 'will print 'stringddd'
```

- `Len() As Integer`
- `GetEntityEncode() As String`
- `Tokenize(delim As String) As Object`
- `Trim() As String`
- `ToInt() As Integer`
- `ToFloat() As Float`
- `Left(chars As Integer) As String`
- `Right(chars As Integer) As String`
- `Mid(start_index As Integer) As String`
- `Mid(start_index As Integer, chars As Integer) As String`
- `Instr(substring As String) As Integer`
- `Instr(start_index As Integer, substring 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: *An integer value of* `5` *is converted to type* roInt *automatically because the* `AddTail()` *method expects a BrightScript Object as its parameter.*

```
BrightScript> list=CreateObject("roList")
BrightScript> list.AddTail(5)
BrightScript> print type(list.GetTail())
```

Example: *Here the* `ListDir()` *method returns an* roList *object containing* roString *objects:*

```
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.

Example:

```
list = CreateObject("roList")
list.AddTail("a")
list.AddTail("b")
list.AddTail("c")
```

```
list.AddTail("d")

list.ResetIndex()
x= list.GetIndex()
while x <> invalid
       print x
       x = list.GetIndex()
end while

print list[2]
```

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 roArray`: 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 roList`: 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*.

Example: 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.

```
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`: Parses the XML content passed to it. In the event of failure, this method returns False. However, it also populates *roXMLElement* with whatever text could be successfully parsed. To avoid passing along erroneous strings, it is always best to have the script check the return value of `Parse()` before using them.
- `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`

Example: 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.

PRESENTATION AND WIDGET OBJECTS

roAudioEventMx

The roAudioPlayerMx object can generate *roAudioEventMx* messages with the following values:

- `8 EVENT_MEDIAENDED`
- `14 EVENT_OVERLAY_MEDIAENDED`
- `16 EVENT_MEDIAERROR`
- `17 EVENT_OVERLAY_MEDIAERROR`

"Media ended" events are sent when a track finishes and there are no more queued tracks for the player, while "Media error" events are sent when a queued file is not found (e.g. when it does not exist).

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: The *roAudioOutput* object requires a single output parameter upon 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.
- `SetTone(treble As Integer, bass As Integer) As Boolean`: Sets the treble and bass of the specified output. The treble and bass integers can range from -1000 to 1000, with 0 indicating no modification to the audio signal. Each increment represents a change of 0.01db.
- `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.
- `SetAudioDelay(delay_in_milliseconds As Integer) As Boolean`: Delays the audio for a specific audio ouput by lagging decoded samples before they reach that output. Delays are limited to 150ms or less. Currently, the system software only supports positive delays; therefore, if you need to set the audio ahead of the video, you will need to use `SetVideoDelay()` instead.

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*, *ifMessagePort*, *ifUserData*, *ifMediaTransport*, *ifAudioControl*.

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

The *ifMessagePort* interface provides the following:

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

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.

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

The *ifAudioControl* interface provides the following:

- `SetPcmAudioOutputs(outputs As roArray) As Boolean`: Specifies which audio connectors should output PCM audio. This method accepts one or more outputs in the form of an *roArray* of *roAudioOutput* instances.

- `SetCompressedAudioOutputs(outputs As roArray) As Boolean`: Specifies which audio connectors should output compressed audio (e.g. Dolby AC3 encoded audio). This method accepts one or more outputs in the form of an *roArray* of *roAudioOutput* instances.

Note: *When one or both of the above output methods are called, they will override the settings of the following* ifAudioControl *methods:* *`SetAudioOutput(), MapStereoOutput(), SetUsbAudioPort(), MapDigitalOutput().`*

- `SetMultichannelAudioOutputs(array As Object) As Boolean`:
- `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 Dynamic) As Boolean`: Specifies the volume of the audio output as either a percentage or decibel amount. To use a percentage measurement, pass an integer value between 0 and 100. To use a decibel measurement, pass an *roAssociativeArray* containing a single `{db:<value As Float>}` parameter. The decibel measurement is an absolute value: passing 0 specifies no change to the audio output, and the effective range of measurements is from aprroximately -80 to 20 decibels.
- `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`
- `SetAudioDelay(delay_in_milliseconds As Integer) As Boolean`: Adds a presentation time stamp (PTS) offset to the audio. This makes it possible to adjust for file multiplexing differences. Delays are limited to

150ms or less. Currently, the system software only supports positive delays; therefore, if you need to set the audio ahead of the video, you will need to use `SetVideoDelay()` instead.

- `SetVideoDelay(delay_in_milliseconds As Integer) As Boolean`: Adds a presentation time stamp (PTS) offset to the video. This makes it possible to adjust for file multiplexing differences. Delays are limited to 150ms or less.

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`

If a video file is playing or has played, you need to call *roVideoPlayer.Stop()* before changing the audio output.

Example: This example shows how to use the `SetPcmAudioOutputs` and `SetCompressedAudioOutputs` methods in conjunction with *roAudioOutput*. The video player is configured to output decoded audio to the analog output or compressed audio to the HDMI and SPDIF outputs.

```
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)
```

Note: *In most cases, rerouting audio outputs during audio/video playback will cause playback to stop. The system software will still be responsive, so you can use commands to exit playback during or following an audio output modification.*

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

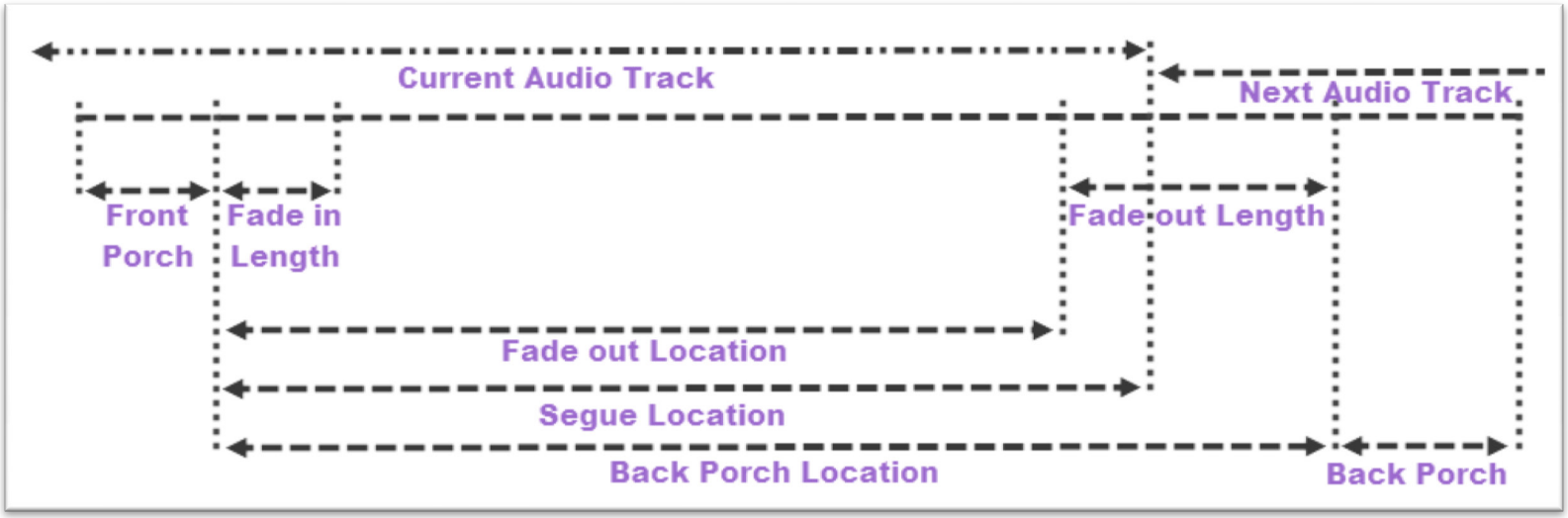
This object allows you to mix audio files, as well as HLS audio streams. Each `roAudioPlayerMx` object conatins two internal audio players: The main audio playlist consists of queued audio tracks that play sequentially, while the audio overlay plays files on top of the main playlist. A fade will not 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) to skip from the start of the track. This value is 0 by default.
- `FadeOutLocation`: The location (in milliseconds) of the fade out relative to the value of the `FrontPorch`. If the `FrontPorch` value is 0 (which is the default setting), and the `FadeOutLength` value is non-zero, then the fade out is calculated back from the end of the file.
- `FadeOutLength`: The length of the fade out (in milliseconds). This value is 0 by default.
- `SegueLocation`: The location (in milliseconds) 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 included, the value defaults to the `FadeOutLocation`.
- `BackPorchLocation`: The location (in milliseconds) 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 included, 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.

- `Overlay`: 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 segue 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.
- `UserString`: A string that can be set to a unique value for each *roAudioPlayerMx* instance. This string is returned with every event generated by the instance. Since all current platforms can support multiple *roAudioPlayerMx* instances running at the same time, the `UserString` allows the script to distinguish between event returns.

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*, *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.

Object Creation: Like other widgets, *roCanvasWidget* is created with an *roRectangle* to set its size and position on the screen.

```
CreateObject ("roCanvasWidget", r As roRectangle) As Object
```

Interfaces: *ifCanvasWidget*

The *ifCanvasWidget* interface provides the following:

- `Hide() As Boolean`: Hides the widget.
- `Show() As Boolean`: Shows the widget.
- `SetRectangle(r As roRectangle) As Boolean`: Changes the size and positioning of the widget rectangle using the passed *roRectangle* object.
- `SetLayer(content As Object, z-level As Integer) As Boolean`: 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) As Boolean`: Clears the specified layer.
- `Clear() As Boolean`: Clears all of the layers.
- `EnableAutoRedraw(enable As Boolean) As Boolean`: 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 the image
- `encryptionalgorithm`: The file-encryption algorithm. Currently the options are "AesCtr" and "AesCtrHmac".
- `encryptionkey`: The key to decrypt the image file. This is a byte array consisting of 128 bits of key, followed by 128 bits of IV.

Note: *See the Image Decryption section in the* roImagePlayer *entry for details on displaying encrypted images.*

- `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: The *roClockWidget* object is created with several parameters.

```
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.
- `res`: A *resources.txt* file that allows localization via the *roResourceManager* object (see below for further details).
- `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 the *resources.txt* file 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)
```

The *ifWidget* interface provides the following:

- `SetForegroundColor(color As Integer) As Boolean`: Sets the foreground *color* in ARGB format.
- `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:/Arial.ttf`).
- `SetBackgroundBitmap(bitmap_filename As String, stretch As Boolean) As Boolean`: Sets the background bitmap image. If `stretch` is True, then the image is stretched to the size of the window.
- `SetBackgroundBitmap(parameters As roAssociativeArray, stretch As Boolean) As Boolean`: Sets the background bitmap image. If *stretch* is True, then the image is stretched to the size of the window. The associative array can contain the following parameters:
  - `Filename`: The name of the image file
  - `EncryptionAlgorithm`: The file-encryption algorithm. Currently the options are "AesCtr" and "AesCtrHmac".
  - `EncryptionKey`: The key to decrypt the image file. This is a byte array consisting of 128 bits of key, followed by 128 bits of IV.

  Note: *See the Image Decryption section in the* roImagePlayer *entry for details on displaying encrypted images.*

- `SetSafeTextRegion(region 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.
- `SetRectangle(r As roRectangle) As Boolean`: Changes the size and positioning of the widget rectangle using the passed *roRectangle* object.

roHdmiInputChanged, roHdmiOutputChanged

The *roVideoMode* object generates an *roHdmiInputChanged* or *roHdmiOutputChanged* event object whenever the hotplug status of the HDMI input or output changes.

At least one *roHdmiOutputChanged* event object will always be generated for a hotplug event, even for very quick disconnect/connect hotplug events. In most cases, a disconnect/connect hotplug event will generate two event objects.

Interfaces: *ifInt*, *ifIntOps*

The *ifInt* interface provides the following:

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

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 positioning of the widget on the screen.

```
CreateObject("roHtmlWidget", rect As roRectangle)
```

Interfaces: *ifHtmlWidget*, *ifMessagePort*, *ifUserData*

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.
- `SetRectangle(r As roRectangle) As Boolean`: Changes the size and positioning of the widget rectangle using the passed *roRectangle* object.
- `SetURL(URL 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(asset_pool As roAssetPool, asset_collection As roAssetCollection, pool_prefix As String, uri_prefix As String) As Boolean`: Sets the mapping between the URL space and the pool files.

- `SetZoomLevel(scale_factor as Float)`: Adjusts the scale factor for the displayed page (the default equals 1.0).
- `EnableSecurity(a As Boolean) 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.
- `SetPortrait(portrait_mode As Boolean) 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(alpha As Integer) As Boolean`: Sets the overall alpha level for the widget (the default equals 255).
- `EnableScrollbars(scrollbars As Boolean) 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) As Boolean`: 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()`.
- `SetHWZDefault(default As String)`: Sets the default HWZ mode for HTML video. Normally, HWZ must be enabled in each `<video>` tag, but passing "on" to this string enables HWZ for all `<video>` elements.
- `ForceGpuRasterization(enable As Boolean) As Boolean`: Enables GPU rasterization for HTML graphics. This method will take effect for subsequent page loads only. If Chromium determines that a page is not compatible, it will refuse to enable GPU rasterization for that page.
- `SetUserStylesheet(URI As String) As Boolean`: Applies the specified user stylesheet to the page(s) loaded in the widget. The parameter can be a URI specifying any `file:` resource in the storage. The stylesheet can also be specified as inline data in the following form: "data:text/css;charset=utf-8;base64,<base64 encoded data>". This method will fail if you specify the inline data in any other order or if you use any data format other than base64.

- `SetAppCacheDir(file_path As String) As Boolean`: 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) As Boolean`: 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.
- `FlushCachedResources() As Boolean`: Discards any resources that WebKit has cached in memory.
- `SetLocalStorageDir(file_path As String)`: Creates a "Local Storage" subfolder in the specified directory. This folder is used by local storage applications such as the JavaScript *storage* class.
- `SetLocalStorageQuota(maximum As Integer)`: Sets the total size (in bytes) allotted to all local storage applications. The default total size is 5MB.
- `SetWebDatabaseDir(file_path As String)`: Specifies the directory that should be used for web database applications (e.g. "SD:/webdb"). This method must be called before using web database applications such as Web SQL or IndexedDB.
- `SetWebDatabaseQuota(maximum As Integer)`: Sets the total size (in bytes) allotted to all web database applications. The default total size is 5MB.
- `EnableJavaScript(enable As Boolean) As Boolean`
- `AllowJavaScriptURLs(url_collection As roAssociativeArray)`: Allows the specified JavaScript BrightScript classes to be used by the specified URLs (all BrightScript classes in JavaScript are disabled by default). This method accepts an associative array that maps JavaScript BrightScript classes to the URL(s) that are allowed to use them.
    - An `all` key indicates that all classes are authorized for the associated URL(s).
    - An asterisk `"*"` value indicates that all URLs are authorized for the associated BrightScript class.
    - A `"local"` value indicates that all local pages are authorized for the associated BrightScript class.

  Example: The following will enable all BrightScript classes for all URLs.

```
html.AllowJavaScriptUrls({ all: "*" })
```

Example: The following will enable all BrightScript classes for local pages and the BrightSign homepage.

```
html.AllowJavaScriptUrls({ all: "local", "http://www.brightsign.biz" })
```

- `PostJSMessage(data As roAssociativeArray) As Boolean`: Posts a collection of key:value pairs to the *BSMessagePort* JavaScript class (see the JavaScript Objects for BrightScript tech note for more details). This method does not support passing nested associative arrays.
- `StartInspectorServer(port As Integer) As Boolean`: Enables the JavaScript console, which allows you to debug JavaScript applications while a webpage is running. To access the console, navigate to the player IP address at the specified port number. See this page for documentation relating to the JavaScript console.
- `SetUserAgent(user_agent As String) As Boolean`: Changes the default user-agent string reported by WebKit.

  Example: The following is a default user-agent string sent by an XD1230.

```
BrightSign/4.7.85.2-8-g1a6e6f6-td-debug (XD1230) Mozilla/5.0 (compatible; Linux mips)
AppleWebKit/537.4 (KHTML, like Gecko)
Chromium/18.0.1025.168 Chrome/18.0.1025.168 Safari/537.4
```

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`

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.

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

This object allows you to access decoded image-file data. You can then copy or manipulate that data.

Object Creation: An *roImageBuffer* object is instantiated with an *roImagePlayer* object and a string specifying the file path of an image file.

```
CreateObject("roImageBuffer", image_player As Object, file_path As String)
```

Example:

```
imgPlayer = CreateObject("roImagePlayer")

imgBuffer = CreateObject("roImageBuffer", imgPlayer, "SD:/content/image.png")
```

Interfaces: *ifImageBufferControl*

The *ifImageBufferControl* interface provides the following:

- `DisplayBuffer(x As Integer, y As Integer) As Boolean`: Displays the image on screen. The x and y integers specify the coordinates of the top-left corner of the image.
- `GetBufferByteArray() As roByteArray`: Returns the decoded image-file data as an *roByteArray*.
- `GetBufferMetadata() As roAssociativeArray`: Returns an associative array containing the following keys:
  - `width`: The width of the image file
  - `height`: The height of the image file
- `ConvertFormat(a As String) As Object`

roImagePlayer

This object displays static bitmap images on the video display. The simplest way to use *roImagePlayer* is to make calls to `DisplayFile()` with the filename as a String. Alternatively, you can use `PreloadFile()` in conjunction with `DisplayPreload()` to have more control. For more pleasing aesthetics when generating an image player, use the *roImageWidget* object.

Object Creation: The image player is displayed by first creating *roRectangle* and *roImagePlayer* instances, then calling `SetRectangle()` using the *roRectangle* instance as the argument.

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

Interfaces: *ifImageControl*

The *ifImageControl* interface provides the following:

- `DisplayFile(image_filename As String) As Boolean`: Displays the image with the specified filename. The `image_filename` string must point to a *.png*, *.jpeg*, or 8-bit, 24-bit, or 32-bit *.bmp* file. Note that *.jpeg* image files with CMYK color profiles are not supported.
- `DisplayFile(parameters As roAssociativeArray) As Boolean`: Displays an image using an associative array of display parameters:
    - `Filename`: The name of the image file
    - `Mode`: The image mode. See the entry for `SetDefaultMode()` below for more details.
    - `Transition`: The image transition setting. See the entry for `SetDefaultTransition()` below for more details.

- `EncryptionAlgorithm`: The file-encryption algorithm. Currently the options are "AesCtr" and "AesCtrHmac".
- `EncryptionKey`: The key to decrypt the image file. This is a byte array consisting of 128 bits of key, followed by 128 bits of IV.

Note: *See the Image Decryption section below for details on displaying encrypted images.*

- `PreloadFile(image_filename As String) As Boolean`: Loads the specified image file into an offscreen memory buffer.
- `PreloadFile(parameters As roAssociativeArray) As Boolean`: Loads an image file into an offscreen memrory buffer. Image display properties are determined by an associative array of parameters:
  - `Filename`
  - `Mode`: See the entry for `SetDefaultMode()` below for more details.
  - `Transition`: See the entry for `SetDefaultTransition()` below for more details.
- `DisplayPreload() As Boolean`: Uses the on-screen memory buffer to display the image stored in the offscreen memory buffer using `PreloadFile()`. There are only two memory buffers: one is displayed on screen; and the other is used for preloading images. `PreloadFile()` can be called multiple times before `DisplayPreload()` is called, and will keep loading into the same off-screen buffer. The `DisplayFile()` method calls `PreloadFile()` followed immediately by `DisplayPreload()`, so any previously preloaded image will be lost. If no image is preloaded, `DisplayPreload()` will have no effect.
- `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`: Sets the default image display mode for `DisplayFile()` and `PreloadFile()`. If `SetDefaultMode()` is not called, then the default mode is set to 0 (equivalent to the image being centered without scaling). The supported display mode are listed below:
  - **0 – Center image**: No scaling takes place. Cropping only occurs if the image is bigger than the window.

    - **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 window, with its aspect ratio maintained.
    - **3 – Scale to fill**: The image is stretched so that it fills the window and the whole image is viewable. The aspect ratio will not be maintained if it is different from the window.
- `SetDefaultTransition(transition As Integer) As Boolean`: Sets the transition to be used when the next image is displayed. The following are available transitions:
    - 0: No transition: immediate blit
    - 1-4: Wipes from top, bottom, left, or right.
    - 5-8: Explodes from centre, top left, top right, bottom left, or bottom right.
    - 10-11: Uses vertical or horizontal venetian-blind effect.
    - 12-13: Combs vertical or horizontal.
    - 14: Fades out to background color, then back in.
    - 15: Fades between current image and new image.
    - 16-19: Slides from top, bottom, left or right.
    - 20-23: Slides entire screen from top, bottom, left, or right.
    - 24-25: Scales old image in, then the new one out again (this works as a pseudo rotation around a vertical or horizontal axis).
    - 26-29: Expands a new image onto the screen from right, left, bottom, or top.
- `SetRectangle(r As roRectangle) As Boolean`: Changes the size and positioning of the image rectangle using the passed *roRectangle* object.
- `SetTransform(transform As String) As Boolean`: Applies one of eight transforms to the image. Calls to this method only take effect when the next file is displayed. Note that the image rectangle itself does not change to accommodate the new height and width ratio of a transformed image. This method can be called separately on multiple *roImagePlayer* or *roImageWidget* instances.

    - `identity`: No transformation (default behavior)
    - `rot90`: 90 degree clockwise rotation
    - `rot180`: 180 degree rotation
    - `rot270`: 270 degree clockwise rotation
    - `mirror`: Horizontal mirror transformation
    - `mirror_rot90`: Mirrored 90 degree clockwise rotation
    - `mirror_rot180`: Mirrored 180 degree clockwise rotation
    - `mirror_rot270`: Mirrored 270 degree clockwise rotation
- `OverlayImage(image_filename As String, x As Integer, y As Integer) As Boolean`: Composites the image with the specified filename on top of the primary `DisplayFile()` image. Use the `x` and `y` integers to specify its location within the image widget.
- `GetRectangle() As roRectangle`: Returns an *roRectangle* object that has the same location and dimensions as the *roRectangle* object used to define the image window.
- `CreateTestHole(hole As roRectangle) As Boolean`: Creates a hole in the image with the location and dimensions specified in the passed *roRectangle* instance. Any video windows located directly beneath the image will show through. This method will disrupt image playback and should be used for test purposes only.
- `SetTransitionDuration(duration As Integer) As Boolean`: Sets the amount of time it takes (in milliseconds) for a specified transition effect to take place. The default transition duration is 1000 milliseconds.
- `DisplayBuffer(a As Object, b As Integer, c As Integer) As Boolean`
- `Hide() As Boolean`: Hides the image currently being displayed by the *roImagePlayer* widget.
- `Show() As Boolean`: Shows the image currently being displayed by the *roImagePlayer* widget.

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.

To display images in a zone, `SetRectangle()` must be called, and `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
```

Image Decryption

The *roImagePlayer*, *roImageWidget*, *roClockWidget*, *roTextWidget*, and *roCanvasWidget* objects can be used to display encrypted images. Each object has an image playback method that accepts an associative array, which can include the `EncryptionAlgorithm` and `EncryptionKey` decryption parameters.

You can call `roDeviceInfo.HasFeature("media_decryption")` to determine if a player model and firmware version supports image decryption.

```
print "Play ENCRYPTED image in an image widget"

imagePlayer = CreateObject("roImageWidget", r1)

aa=CreateObject("roAssociativeArray")
aa.filename = "sd:/images_enc.jpg"
aa.encryptionalgorithm = "AesCtr"
aa.encryptionkey = CreateObject("roByteArray")
aa.encryptionkey.fromhexstring("01030507090b0d0f00020406080a0c0e000000000000000000000000000000000")

imagePlayer.DisplayFile(aa)
sleep(10000)
imagePlayer.Hide()
```

```
print "Play ENCRYPTED image with PlayStaticImage"

videoPlayer = CreateObject("roVideoPlayer")

aa=CreateObject("roAssociativeArray")
aa.filename = "sd:/images_enc.jpg"
aa.encryptionalgorithm = "AesCtr"
aa.encryptionkey = CreateObject("roByteArray")
aa.encryptionkey.fromhexstring("01030507090b0d0f00020406080a0c0e000000000000000000000000000000000")

videoPlayer.PlayStaticImage(aa)
sleep(10000)
videoPlayer = invalid

print "Show CLOCK image"

resourceManager = CreateObject("roResourceManager", "sd:/resources.txt")

clockWidget = CreateObject("roClockWidget", r1, resourceManager, {})

aa=CreateObject("roAssociativeArray")
aa.filename = "sd:/images_enc.jpg"
aa.encryptionalgorithm = "AesCtr"
aa.encryptionkey = CreateObject("roByteArray")
aa.encryptionkey.fromhexstring("01030507090b0d0f00020406080a0c0e000000000000000000000000000000000")

clockWidget.SetBackgroundBitmap(aa, True)
```

```
clockWidget.Show()
sleep(10000)
clockWidget.Hide()

print "Text widget with encrypted background image"

twParams = CreateObject("roAssociativeArray")
twParams.LineCount = 1
twParams.TextMode = 1
twParams.Rotation = 0
twParams.Alignment = 1

tw=CreateObject("roTextWidget",r1,1,2,twParams)
tw.SetBackgroundColor(&h00ff0000)
tw.SetForegroundColor(&hff00ff00)
tw.PushString("Encrypted Background")
'tw.SetRectangle(r)

aa=CreateObject("roAssociativeArray")
aa.filename = "sd:/images_enc.jpg"
aa.encryptionalgorithm = "AesCtr"
aa.encryptionkey = CreateObject("roByteArray")
aa.encryptionkey.fromhexstring("01030507090b0d0f00020406080a0c0e000000000000000000000000000000000")

tw.SetBackgroundBitmap(aa, True)
tw.Show()
sleep(10000)
```

```
tw.Hide()

cw=CreateObject("roCanvasWidget", rect)
canvas_aa=CreateObject("roAssociativeArray")
canvas_aa.Filename = "sd:/images_enc.jpg"
canvas_aa.Encryptionalgorithm = "AesCtr"
canvas_aa.EncryptionKey = CreateObject("roByteArray")
canvas_aa.EncryptionKey.FromHexString("01030507090b0d0f00020406080a0c0e0000000000000000000000000000
00000")
cw.SetLayer(canvas_aa, 1)
cw.Show()
```

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*.

Object Creation: The image widget area is generated using an *roRectangle* object.

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

Interfaces: *ifImageControl*

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

This object includes overloaded `PreloadFile()` and `DisplayFile()` methods. These methods receive an *roAssociativeArray* object that 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 method calls.

Example: This code uses `PreloadFile()` method for a multiscreen display:

```
i=CreateObject("roImageWidget")
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* values are the same as those documented in the *roImagePlayer.ifImageControl* section, but the multiscreen parameters are unique. The *MultiscreenWidth* and *MultiscreenHeight* parameters specify the width and height of the multi-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("roImageWidget")
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 video player for letterboxing, it will occur if the video does not fit exactly into the new rectangle.

roShoutcastStream

This object allows playback of shoutcast streams.

Object Creation: The *roShoutcastStream* object is created with a URL object, a maximum buffer size (in seconds), and an initial buffering duration (in seconds).

```
CreateObject("roShoutcastStream", url_transfer As Object, buffer_size As Integer,
buffer_duration As Integer)
```

Interfaces: *ifShoutcastStream*, *ifMessagePort*, *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 *ifMessagePort* 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, y 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 Dynamic) As Void`: Writes the characters specified at the current cursor position and advances the cursor to one position beyond the last character. This method can support either a string or an *roByteArray*. If the block is a string, any null bytes will terminate the block.
- `SetSendEol(string As String) As Void`: Sets the sequence sent at the end of a `SendLine()` value. You should leave this at the default ASCII value of 13 (Carriage Return) for normal use. If you need to change this value to another non-printing character, use the `chr()` global function.

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 display text on the screen.

Object Creation: This object is created with several parameters.

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

- `r` : An *roRectangle* instance that contains the text
- `line_count`: The number of lines of text to show within the rectangle
- `text_mode`: The animation characteristics of the text:
  - `0`: An animated view similar to teletype
  - `1`: Static text
  - `2`: Simple text with no queue of strings
  - `3`: Smooth right-to-left scrolling ticker
- `pause_time`: The length of time each string is displayed before displaying the next string. This does not apply to text mode 2 or 3 because the strings on screen are updated immediately.

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

- `r` : An *roRectangle* that contains the text
- `line_count`: The number of lines of text to show within the rectangle
- `text_mode`: The animation characteristics of the text:
  - `0`: An animated view similar to teletype
  - `1`: Static text
  - `2`: Simple text with no queue of strings
  - `3`: Smooth right-to-left scrolling ticker (strings are separated by a diamond)

- `array`: An associative array that can include the following values:
  - "LineCount": The number of lines of text to show within the rectangle.
  - "TextMode": The animation characteristics of the text:
    - `0`: An animated view similar to teletype
    - `1`: Static text
    - `2`: Simple text with no queue of strings
    - `3`: Smooth right-to-left scrolling ticker (strings are separated with a diamond by default; the separator can be modified using the `SetSeparator()` method)
  - "PauseTime": The length of time each string is displayed before displaying the next string. This does not apply to text mode 2 or 3 because the strings on screen are updated immediately.
  - "Rotation": The rotation of the text within the widget:
    - `0`: 0 degrees
    - `1`: 90 degrees. This value can also be represented in degrees (`90`) or radians (`.5Π`).
    - `2`: 180 degrees. This value can also be represented in degrees (`180`) or radians (`Π`).
    - `3`: 270 degrees. This value can also be represented in degrees (`270`) or radians (`1.5Π`).
  - "Alignment": The alignment of the text:
    - `0`: Left
    - `1`: Center
    - `2`: Right

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.
- `SetStringSource(file_path As String) As Boolean`: Displays the text file at the specified path as a single, continuous string. This method is only applicable to text mode 3 (scrolling ticker). When the end of the file is reached, the text widget loops to the beginning, using a diamond symbol as the separator.
- `SetAnimationSpeed(speed As Integer) As Boolean`: Sets the speed at which animated text displays. This method is applicable to text modes 0 and 3 only:
    - Mode 0: The default `speed` value is 10000. Setting an integer above this value decreases the speed of the teletype-style ticker: For example, specifying a value of 20000 will decrease the default speed at which text displays by half, while a value of 5000 will double the default speed.
    - Mode 3: The default `speed` value is 10000. Because the speed of a scrolling ticker is measured in pixels per second (PPS), the speed must be a multiple of the current framerate, or else it will be rounded down to the nearest multiple (for example, a framerate of 60p will honor PPS values of 60, 120, 180, etc.). The software determines the speed of the scrolling ticker by performing the following calculation on the passed `speed` parameter:
      `PPS = (speed * 60) / 10000`
- `SetSeparator(separator As String) As Boolean`: Changes the separator between strings. The default diamond separator will be replaced by the contents of the passed string. This method applies to Text Mode 3 (smooth scrolling ticker) only. The following strings indicate special symbols: `":diamond:"`, `":circle:"`, `":square:"`.
- `SetMultiscreen(offset As Integer, size As Integer, ip_address As String, port As Integer) As Boolean`: Allows for a smooth-scrolling ticker to be displayed across multiple screens. The master

screen is designated as the instance with the rightmost `offset` of all the players in the multiscreen array; all `PushString()` and `Show()` calls (as well as any other changes) must be made on the master instance. Slave instances of the text widget will remain blank until the master starts. This method requires the following parameters:

- `offset`: The offset (in pixels) of the display in the multiscreen array. For example, using an `offset` of `1920` in a two-screen array of 1920x1080 screens would specify this player as the right-hand (master) display.
- `size`: The total width (in pixels) of the multiscreen array. For example, defining a `size` of `3840` would specify a two-screen array of 1920x1080 screens.
- `ip_address`: A string specifying the multicast IP address for the PTP synchronization process (e.g. "239.192.0.0")
- `port`: A string specifying the multicast port for the PTP synchronization process (e.g. "1234").

Note: *Players can support more than one multiscreen ticker at a time.*

This *ifWidget* interface provides the following:

- `Show() As Boolean`: Displays the widget. After creation, the widget is hidden until `Show()` is called.
- `Hide() As Boolean`: Hides the widget.
- `SetForegroundColor(color As Integer) As Boolean`: Sets the foreground color in ARGB format.
- `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:/ComicSans.ttf`).
- `SetBackgroundBitmap(bitmap_filename As String, stretch As Boolean) As Boolean`: Sets the background bitmap image. If `stretch` is True, then the image is stretched to the size of the window.
- `SetBackgroundBitmap(parameters As roAssociativeArray, stretch As Boolean) As Boolean`: Sets the background bitmap image. If *stretch* is True, then the image is stretched to the size of the window. The associative array can contain the following parameters:
  - `Filename`: The name of the image file

    - `EncryptionAlgorithm`: The file-encryption algorithm. Currently the options are "AesCtr" and "AesCtrHmac".
    - `EncryptionKey`: The key to decrypt the image file. This is a byte array consisting of 128 bits of key, followed by 128 bits of IV.

  Note: *See the Image Decryption section in the* roImagePlayer *entry for details on displaying encrypted images.*
- `SetSafeTextRegion(region As roRectangle) As Boolean`: Specifies the rectangle within the widget where the text can be drawn safely.
- `SetRectangle(r As roRectangle) As Boolean`: Changes the size and positioning of the widget rectangle using the passed *roRectangle* object.
- `GetFailureReason() As String`: Yields additional useful information if a function return indicates an error.

The top 8 bits of the color value are "alpha," affecting both the foreground text color and 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 are declared as separate classes. Events can have one of the following values, which are retrieved using the `GetInt()` method:

3 `Playing`: The current media item has started playing.

8 `MediaEnded`: The media item has completed playback.

12 `TimeHit`: A particular timecode has been reached. See the entry on Timecode Events for more details.

13 `Overlay_Playing`: An *roAudioPlayerMx* instance has begun playback of an audio file.

14 `Overlay_MediaEnded`: An *roAudioPlayerMx* instance has completed playback of an audio file.

15 `Overlay_TimeHit`: The `EventTimeStamp` of an *roAudioPlayerMx* instance has been been reached.

16 `MediaError`: A media error has been detected.

17 `Overlay_MediaError`: A media error has been detected during *roAudioPlayerMx* playback.

18 `FadingOut`: The current media item has begun to fade out. See the *roVideoPlayer.SetFade()* entry for more details.

19 `DecoderEOS`

20 `Overlay_FadingOut`: The `FadeOutLocation` of an *roAudioPlayerMx* instance has been reached.

21 `Overlay_DecoderEOS`

25 `SourceChanged`: The resolution of the source video has changed.

26 `Underflow`: The stream seems to be underflowing. This event usually indicates that the streaming latency is set too low. It will be generated every few seconds as long as underflow is detected.

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 HDMI input or video provided by a video capture dongle. Note that the *ifVideoInput* methods do not apply to HDMI input, which can be achieved by passing an unmodified *roVideoInput* instance to the *roVideoPlayer.PlayFile()* method (see below for examples).

roVideoInput is created with no parameters:

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

Interfaces: *ifVideoInput*

The *ifVideoInput* interface provides the following:

- `GetStandards() As roArray`
- `GetInputs() As roArray`: 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(standard As String) As Boolean`
- `GetCurrentStandard() As String`
- `SetInput(input As String) As Boolean`
- `GetCurrentInput() As String`: Use the above to get and set the input and video standard.
- `GetControls() As roArray`: 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 uses the HDMI Input as the video source to create a full-screen display.

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

vm = CreateObject("roVideoMode")
vm.SetMode("1920x1080x60p")

r = CreateObject("roRectangle", 0, 0, 1920, 1080)
v.SetRectangle(r)

v.PlayFile(i)
```

Example: This script uses the video capture dongle as the video source to create a full-screen display.

```
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.PlayFile(i)
```

roVideoMode

This object allows you to configure resolution and other video output settings. 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.

The *roVideoMode* object generates an *roHdmiInputChanged*/*roHdmiOutputChanged* event object whenever the hotplug status of the HDMI input or output changes.

Interfaces: *ifVideoMode*, *ifMessagePort*, *ifUserData*

The *ifVideoMode* interface provides the following:

- `SetMode(mode As String) As Boolean`: Sets the video output mode. If the specified video mode is different from the current video mode of the object, the unit will reboot and change the video mode to the new setting during system initialization. The supported video modes are listed in the Video Resolutions FAQ. This method also accepts "auto" as a mode parameter. The following optional parameters can be appended to the string:
  - `<resolution>:<color_space>:<depth>bit`: The video profile for HDMI output. For example, to output 4Kp60 at the 4:2:0 color space with 10 bits of depth, you would pass the following string: `"3840x2160x60p:420:10bit"`.
  - `<resolution>:preferred`: A preferred video-mode flag. This instructs the player to only use the video mode if the display EDID indicates that it is supported. Otherwise, the output will default to "auto" mode. If no EDID is detected at bootup, the player will output the preferred video mode. If an HDMI hotplug event occurs afterward, then the player will perform the preferred video-mode check again. The `:preferred` flag currently ignores video profile settings (i.e. color space and bit depth).

Note: *BrightSign hardware has a video anti-aliasing low-pass filter that is set automatically.*

- `SetModeForNextBoot(video_mode As String) As Boolean`: Specifies the target video mode of the device the next time it reboots. Once a video mode is specified using `SetMode()`, it can only be changed by a device reboot.
- `Set3dMode(mode As Integer) As Boolean`: Sets the 3D video output mode, which is specified by passing one the following parameters:
  - `0`: Standard mono video (default)
  - `1`: Side-by-side stereo video
  - `2`: Top-and-bottom stereo video
- `GetBestMode(connector As String) As String`
- `GetMode() As String`: Returns the current video mode of the device, which is specified using the `SetMode()` method.
- `GetActiveMode() As AssociativeArray`: Returns information about the current video mode as an associative array. All values in the *roAssociativeArray* are strings:
  - `videomode`: The current video mode (e.g. "3840x2160x60p")
  - `colordepth`: The current color depth ("8bit", "10bit", or "12bit")
  - `colorspace`: The current color space ("RGB" or "YUV")
  - `preferred`: A "true" or "false" string indicating whether the current video mode is the preferred mode, which is set using the `SetMode()` method
- `GetConfiguredMode() As AssociativeArray`: Returns information about the video mode configured using the `SetMode()` method. This method returns an *roAssociativeArray* of strings. If the video mode is set to "auto", this method will return Invalid:
  - `videomode`: The configured video mode (e.g. "3840x2160x60p")
  - `colordepth`: The configured color depth ("8bit", "10bit", or "12bit")
  - `colorspace`: The configured color space ("rgb", "yuv420", or "yuv422")
  - `preferred`: A "true" or "false" string indicating whether the configured video mode is the preferred mode, which is specified using the `SetMode()` method.
  - `width`: The width of the video plane

    - `height`: The height of the video plane
    - `graphicsPlaneHeight`: The height of the graphics plane. The maximum value is "1080", even for 4K video.
    - `graphicsPlaneWidth`: The width of the graphics plane. The maximum value is "2048", even for 4K video.
    - `framerate`: The framerate of the video output
    - `interlaced`: A "true" or "false" string indicating whether the video output is interlaced
    - `dropframe`: A "true" or "false" string indicating whether the video output is dropping frames to account for an FPS discrepancy between the source video and the display
- `GetModeForNextBoot() As String`: Returns the target video mode of the device the next time it reboots. The return value is specified with the `SetModeForNextBoot()` method.
- `GetFPS() As Integer`: Returns the current framerate of the video output.
- `Screenshot(parameters As roAssociativeArray) As Boolean`: Captures a screenshot of the video and graphics layer as a *.jpeg* or *.bmp* file in the `temp:/` directory. The screenshot process is configured by passing an associative array of parameters to the method:
    - `filename`: The name of the image file that will be saved.
    - `Width`: The width dimension of the image file.
    - `Height`: The height dimension of the image file.

    Note: *The default dimensions of the image file is 640x480.*
    - `filetype`: A string determining whether the image is a "JPEG" or "BMP" file type.
    - `quality`: An integer value (between 0 and 100) that determines the image quality of the screenshot. This parameter is set to 50 by default.
    - `Async`: An integer value that determines whether the screenshot should be taken synchronously or asynchronously. If set to 0, the function returns True after the image file has successfully finished writing. If set to 1, the function will return True prior to saving the file, then return an *roScreenShotComplete* event once the file has finished writing.
- `GetResX() As Integer`: Returns the current width of the graphics plane.
- `GetResY() As Integer`: Returns the current height of the graphics plane.

- `GetVideoResX() As Integer`: Returns the current width of the video plane.
- `GetVideoResY() As Integer`: Returns the current height of the video plane.
- `GetOutputResX() As Integer`: Returns the width of the display for the current video mode.
- `GetOutputResY() As Integer`: Returns the height of the display for the current video mode.
- `GetSafeX() As Integer`: Returns the horizontal coordinate for the upper-left corner of the "safe area". For modes that are generally displayed with no overscan, this will be zero.
- `GetSafeY() As Integer`: Returns the vertical coordinate for the upper-left corner of the "safe area". For modes that are generally displayed with no overscan, this will be zero.
- `GetSafeWidth() As Integer`: Returns 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`: Returns 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

- `SetGraphicsZOrder(order As String)`: Specifies the order of the graphics plane (which includes all graphical elements) in relation to the video plane(s). This method accepts three parameters:
  - `"front"`: Places the graphics plane in front of the video plane(s).
  - `"middle"`: Places the graphics plane between two video planes. This option is only applicable to XDx30, XDx32, and 4Kx42 models.
  - `"back"`: Places the graphics plane behind the video plane(s).

  If the player is rendering two videos, the `front` and `back` options will always place the graphics plane in front of or behind both video planes. To determine the z-order of video planes in relation to one another, use the `ToFront()` and `ToBack()` methods provided by the *roVideoPlayer* object. The following table shows all possible video and graphics z-order arraignments that can be specified using the `SetGraphicsZOrder()` method and calling the `ToFront()` and `ToBack()` methods on a "Video1" *roVideoPlayer* instance.

`SetGraphicsZOrder()`	`front`	`middle`	`back`

ToFront()/ToBack()	ToFront()	ToBack()	ToFront()	ToBack()	ToFront()	ToBack()
Z-Order	Graphics	Graphics	Video1	Video2	Video1	Video2
	Video1	Video2	Graphics	Graphics	Video2	Video1
	Video2	Video1	Video2	Video1	Graphics	Graphics

- `SetImageSizeThreshold(parameters As roAssociativeArray) As Boolean`: Changes the maximum allowed resolution for images. The default image resolution limit is 2048x1080. Displaying images larger than the default value may deplete the graphics memory and cause a crash, so we recommend testing a script that uses this method thoroughly before deploying it in a production environment. This method accepts an associative array with the following parameters:
  - `width`: Overrides the maximum allowed width with the specified value.
  - `height`: Overrides the maximum allowed height with the specified value.
  - `ignore`: Disables resolution limits completely if the value is `1`.
- `AdjustGraphicsColor(parameters As roAssociativeArray) As Boolean`: Adjusts the video and graphics output of the player using the following parameters, which can be passed to the method as an associative array: "brightness", "hue", "contrast", "saturation". Each parameter has a default value of 0 and can accept a range of values between -1000 and 1000.
- `GetHdmiOutputStatus() As roAssociativeArray`: Returns an associative array of Boolean and integer values if an HDMI output is currently connected to a display device. This method will return Invalid if the HDMI output is currently not connected to a display device. The associative array contains the following parameters:
  - `output_present`: Returns True if the HDMI output is connected to a display device or False if no device is present.
  - `output_powered`: Returns True if the display device is on (i.e. RX powered) or False if it is off.
  - `audio_bits_per_sample`: The number of bits per audio sample
  - `audio_format`: The format of the audio output. A "PCM" value indicates that the player is sending decoded output.
  - `audio_channel_count`: The number of audio channels in the output

    - `audio_sample_rate`: The audio sample rate (in hertz)
- `GetHdmiInputStatus() As roAssociativeArray`: Returns an associative array of Boolean and integer values if an HDMI input is currently connected to the device (4K1142, XD1132, XD1230 only). This method will return Invalid if there is currently no HDMI input source. The associative array contains the following parameters:
    - `width`: Lists the pixel width of the video input.
    - `height`: Lists the pixel height of the video input.
    - `interlaced`: Returns True if the video input is interlaced or False if it is not interlaced.
    - `device_present`: Returns True if there is an HDMI input device present or False if there is no HDMI input device present.
- `GetTxHDCPStatus() As roAssociativeArray`: Returns an associative array indicating the current HDCP status of the HDMI output. The associative array currently contains a single key labeled `state`, which can have the following values:
    - `"not-required"`: HDCP has not been requested.
    - `"authenticated"`: HDCP has been enabled and successfully negotiated.
    - `"authentication-in-progress"`: HDCP has been enabled, but authentication has not been completed.
    - `"authentication-failed"`: HDCP has been requested but could not be negotiated.
- `ForceHDCPOn(force As Boolean) As Boolean`: Forces HDCP authentication on the HDMI output if passed True. Passing False to this method will prevent forced authentication attempts with subsequent hotplug events. This method will return False if the player does not support HDCP or if `ForceHDCPOn()` has already been called with the same value.
- `DisableHDCPRepeater(disable As Boolean) As Boolean`: Prevents HDCP authentication from taking place on the HDMI input if passed True. The HDMI source will treat the player like any other non-HDCP authenticated HDMI sink. This method returns False if the HDCP state could not be changed, indicating that there's no HDMI input on the player or that HDCP has already been disabled.

- `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.
- `EnableVideo(enable As Boolean) As Boolean`: Enables video output from the device if True. Setting this method to False disables all video output from the device. This method is set to True by default.
- `IsAttached(connector As String) As Boolean`: Returns True if the specified video connector is attached to an output device. This method can be passed the following parameters (note that they are case sensitive):
  - "hdmi"
  - "vga"
- `HdmiAudioDisable(disable As Boolean) As Boolean`: Disables audio output if True. This method is set to False by default.
- `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.

- `SaveEdids(filename As String) As Boolean`: Saves the EDID information of the display(s) connected via HDMI and/or VGA. The EDID fields are saved sequentially as raw binaries into the specified file. The EDID sets are

two 2kb each, resulting in a maximum file size of 4kb. This method returns True upon success and False upon failure.

- `GetEdidIdentity(video_connector As Boolean) As roAssociativeArray`: Returns an associative array with EDID information from a compatible monitor/television. Passing True with this method specifies EDID over HDMI, while passing False specifies EDID over VGA. These are the possible parameters returned in the associative array:
  - `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(true)` at this point retrieves the new EDID information.

The *ifMessagePort* interface provides the following:

- `SetPort(obj As Object) As Void`

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.

"Auto" Video Mode

If the mode is set to "auto," the BrightSign player will use the following algorithm to determine the best video mode to use based on connected hardware:

1. Try VGA: If VGA is attached, use the highest-resolution mode (as reported by the monitor) that the player supports.
2. Try HDMI: If HDMI is attached, use the highest-resolution mode (as reported by the monitor) that the player supports.
3. Default to 1024x768x75p.
4. If an HDMI hotplug event occurs at any point, recheck the monitor EDID to determine if the highest-resolution mode has changed. If it has changed, reboot the player and use the new video mode.

roVideoPlayer

This object is used to play back video files (using the generic *ifMediaTransport* interface). If the message port is set, the object will send events of the type *roVideoEvent*. All object calls are asynchronous. That is, video playback is handled in a different thread from the script, and the script will continue to run while video is playing. Decoded video will be scaled to the output resolution specified by *roVideoMode*.

To display video in a zone, you must call `SetRectangle()`. In firmware versions 6.0.x and later, zone support is enabled by default

Interfaces: *ifIdentity*, *ifMessagePort*, *ifUserData*, *ifAudioControl*, *ifAudioAuxControl*, *ifVideoControl*, *ifMediaTransport*.

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort) As Void`

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.

See *roAudioPlayer* for documentation of *ifAudioControl*.

The *ifAudioAuxControl* interface provides the following:

- `MapStereoOutputAux(mapping As Integer) As Boolean`

- `SetVolumeAux(a As Integer) As Boolean`
- `SetChannelVolumesAux(channel_mask As Integer, b As Integer) As Boolean`
- `SetAudioOutputAux(audio_output As Integer) As Boolean`
- `SetAudioModeAux(audio_mode As Integer) As Boolean`
- `SetAudioStreamAux(stream_index As Integer) As Boolean`
- `SetUsbAudioPortAux(a As Integer) As Boolean`

The *ifVideoControl* interface provides the following:

- `PlayStaticImage(filename As String) As Boolean`: Uses the video decoder to display an image. On 4Kx42 models, you can use the video decoder to display 4K images.
- `PlayStaticImage(parameters As roAssociativeArray) As Boolean`: Uses the video decoder to display an image. The passed associative array can contain the following parameters:
  - `Filename`: The name of the image file
  - `EncryptionAlgorithm`: The file-encryption algorithm. Currently the options are "AesCtr" and "AesCtrHmac".
  - `EncryptionKey`: The key to decrypt the image file. This is a byte array consisting of 128 bits of key, followed by 128 bits of IV.

  Note: *See the Image Decryption section in the* roImagePlayer *entry for details on displaying encrypted images.*
- `SetViewMode(mode As Integer) As Boolean`: Sets the scaling of the video in relation to the video window. The passed integer can be one of the following values:
  - `0`: Scales the video to fill the window. The aspect ratio of the source video is ignored, so the video may appear stretched/squashed.
  - `1`: Letterboxes and centers the window. The aspect ratio of the source window is maintained.
  - `2`:(Default) Scales the video to fill the window. The aspect ratio is maintained, so the video may appear cropped.

Note: *View modes rely on correct aspect-ratio marking for video files, and not all files may be marked correctly.*

- `SetRectangle(r As roRectangle) As Void`: Specifies the placement and dimensions of the video window using a passed *roRectangle* instance.
- `Hide() As Boolean`: Hides the video window.
- `Show() As Boolean`: Shows the video window.
- `EnableSafeRegionTrimming(a As Boolean) As Boolean`
- `AdjustVideoColor(parameters As roAssociativeArray) As Boolean`: Adjusts the video and graphics output of the player using the following parameters, which can be passed to the method as an associative array: "brightness", "hue", "contrast", "saturation". Each parameter has a default value of 0 and can accept a range of values between -1000 and 1000.
- `SetKeyingValue(keying_settings As roAssociativeArray) As Boolean`: Applies a mask to each pixel in the video window. If the pixel value falls within the specified range of chroma and luma key values, the pixel will appear transparent, allowing video and graphics behind it to show through. If the pixel value does not fall within the specified range, the pixel is unaltered. The chroma and luma key values are set using integers contained in the passed associative array:
  - `luma`
  - `cr`
  - `cb`

  Each integer value is arranged as follows: `[8 bits of mask][8 bits of high-end range][8 bits of low-end range]`. For example, an 0xff8040 value for luma would mask luma at 0xff (no change) and then apply a range from 0x40 to 0x80 for changing to transparent alpha.

Note: *Chroma and luma keying work well with simple shapes and patterns; complex patterns like hair or grass will not be masked effectively*.

- `SetTransform(transform As String) As Boolean`: Applies one of eight transforms to the video plane. This method works equally well with all video sources (files, streams, HDMI input) and can be called separately on multiple *roVideoPlayer* instances. Calls to this method only take effect when the next file/source is played, and transitions to a transformed video do not take place seamlessly.
  - `identity`: No transformation (default behavior)

    - `rot90`: 90 degree clockwise rotation
    - `rot180`: 180 degree rotation
    - `rot270`: 270 degree clockwise rotation
    - `mirror`: Horizontal mirror transformation
    - `mirror_rot90`: Mirrored 90 degree clockwise rotation
    - `mirror_rot180`: Mirrored 180 degree clockwise rotation
    - `mirror_rot270`: Mirrored 270 degree clockwise rotation
- `GetFilePlayability(filename As String) As roAssociativeArray`: Returns an associative array indicating the playability of the video file. For the following keys, a "playable" value indicates that the component is playable, while a "no media" value indicates that there is no media—any other value indicates that the media is unplayable.
    - `audio`: The audio file associated with the video
    - `video`: The video file associated with the video
    - `file`: The video container file
- `GetProbePlayability(probe_string As String) As roAssociativeArray`: Returns an associative array indicating the playability of the probe string. For the following keys, a "playable" value indicates that the component is playable, while a "no media" value indicates that there is no media—any other value indicates that the media is unplayable.
    - `audio`: The audio file associated with the video
    - `video`: The video file associated with the video
    - `file`: The video container file
- `GetStreamInfo() As roAssociativeArray`: Returns an associative array containing information about an IP stream. The associative array contains the following parameters:
    - `Source`: The source address of the IP stream
    - `SrcAddress`: The IP address of the source

- `DstAddress`: The multicast address on which the IP stream is being transmitted. This value may be absent if the RTSP service has not redirected the stream (in this case, the IP address of the player may be displayed instead).
- `encapsulation`: The encapsulation of the IP stream. This value can be "ES" (elementary stream), "TS" (transport stream), or "UNKNOWN".
- `AudioFormat`: The format of the audio file
- `AudioSampleRate`: The audio sample rate (in hertz)
- `AudioChannelCount`: The number of audio channels
- `AudioDuration`: The duration of the audio track
- `VideoFormat`: The format of the video file
- `VideoColorDepth`: The color depth of the video (in bits)
- `VideoWidth`: The width of the video (in pixels)
- `VideoHeight`: The height of the video (in pixels)
- `VideoAspectRatio`: The aspect ratio of the video
- `VideoDuration`: The duration of the video

- `GetStreamStatistics() As roAssociativeArray`: Returns an associative array containing statistics associated with the IP stream. The associative array contains the following parameters:

  Note: *All counters are reset every time `PlayFile()` is called. The audio keys will not be included in the associative array if there is no audio in the stream.*

  - `Bitrate`: The video bitrate
  - `NumDisplayed`: The number of video frames displayed. This is based on the refresh rate of the monitor.
  - `NumUnderflowed`: The number of times the video FIFO has under-flowed. This usually indicates that the buffer size needs to be increased.
  - `NumDecodeErrors`: The number of video frames with decode errors
  - `NumDecoded`: The total number of video frames decoded
  - `NumAudioDecoded`: The total number of audio frames decoded
  - `NumAudioDecodeErrors`: The number of audio frames with decode errors

    - `NumAudioDummy`: The total number of missing audio frames. This value will increment when an audio frame goes missing or a timestamp is incorrect. A couple of frames will often be registered when streaming begins.
    - `NumAudioUnderflows`: The number of times the audio FIFO has under-flowed. This usually indicates that the buffer size needs to be increased.
    - `VideoFramesPerSecond`: The video frame rate (in frames per second)
    - `VideoInterlaced`: A flag indicating whether the video frames are interlaced or progressive
- `SetPreferredVideo(description As String) As Boolean`: Chooses a video stream from the video input based on the parameters in the passed string.
- `SetPreferredAudio(description As String) As Boolean`: Chooses an audio stream from the video input based on the parameters in the passed string.
- `SetPreferredCaptions(description As String) As Boolean`: Chooses a data stream from the video input based on the parameters in the passed string.
- `SetDecoder(decoder As Integer) As Boolean`: Assigns the *roVideoPlayer* instance to a video decoder. Possible values include 0 and 1 for dual-decoder platforms (4Kx42, XDx32, XDx30, HDx22). You can revert back to automatic decoder allocation by passing -1 to this method.
- `SetMosaic(mosaic As Boolean) As Boolean`: Specifies that the *roVideoPlayer* instance use its video decoder as a mosaic decoder.
- `SetMaxResolution(x As Integer, y As Integer) As Boolean`: Specifies the maximum resolution that the mosaic decoder will support. This determines the number of mosaic decoders you can successfully create. You can clear this value by passing 0 to this method.

The *ifMediaTransport* interface provides the following:

- `PlayFile(source As Object) As Boolean`: Plays a video file or HDMI Input. To play a file, pass a string specifying the file name and path. To play HDMI Input, pass an *roVideoInput* instance.
- `PlayFile(parameters As roAssociativeArray) As Boolean`: Plays video using the parameters passed as an associative array. This method has several uses: Playing synchronized (and multiscreen) video using the parameters provided by the *roSyncManager* object; playing streaming video from an *roRtspStream* object; playing

encrypted files; or playing RF Input using the associative array provided by the `CreateChannelDescriptor()` method on the *roChannelManager* object.

- `SetPlaybackSpeed(speed as Float) As Boolean`: Modulates the playback speed of the video, using the float 1.0 as the value for standard playback speed. To fast-forward the video, pass a value greater than 1.0; to rewind the video, pass a negative value. A value between 0 and 1.0 will the play the video in slow motion.
- `PreloadFile(parameters As roAssociativeArray) As Boolean`
- `Stop() As Boolean`
- `Play() As Boolean`
- `SetLoopMode(mode As Dynamic) As Boolean`: Specifies the looping mode for media playback. If this method is passed True, a single media file will loop seamlessly. Setting this method to False, which is the default behavior, allows for playback of multiple files in a playlist—with noticeable gaps between the end and beginning of the file. Alternatively, this method can accept an associative array with the following parameters:
  - `enable`: A Boolean value. If the `enable_if_seamless` parameter is present in the associative array, the `enable` parameter enables or disables conditional seamless looping. If the `enable_if_seamless` parameter is not present, the `enable` parameter enables or disables standard seamless looping.
  - `enable_if_seamless`: A Boolean value that enables or disables conditional seamless looping. If this parameter is True, looping will occur only if the file is capable of being looped seamlessly. If this parameter is False, seamless looping is disabled completely.

  Note: *Media End events are only sent if seamless looping is disabled, or if `enable_if_seamless` is enabled and the file cannot be looped seamlessly.*
- `AddEvent(user_data As Integer, time_in_ms As Integer) As Boolean`: Adds a trigger that will generate an *roVideoEvent* when it reaches the specified time. The user data will be passed with the event and can be retrieved using the *roVideoEvent.GetData()* method. See the Video Timecode Events section below for more details.
- `ClearEvents() As Boolean`: Removes all events that have been added using the `AddEvent()` method.
- `StopClear() As Boolean`

- `Pause(parameters As roAssociativeArray) As Boolean`: Pauses the video file or stream. This method accepts an optional associative array containing the following parameter:
  - `SyncIsoTimeStamp`: The time stamp for pausing synchronized video. This value is provided by the *roSyncManager.Synchronize()* method on the master unit and the *roSyncManagerEvent.GetIsoTimeStamp()* method on slave unit(s).
- `Resume(parameters As roAssociativeArray) As Boolean`: Resumes a paused video file or stream. This method accepts an optional associative array containing the following parameter:
  - `SyncIsoTimeStamp`: The time stamp for resuming synchronized video. This value is provided by the *roSyncManager.Synchronize()* method on the master unit and the *roSyncManagerEvent.GetIsoTimeStamp()* method on slave unit(s).
- `PlayEx(a As Object) As Boolean`: This object has been deprecated. We suggest using the `PlayFile()` method for video playback instead.
- `GetPlaybackPosition() As Integer`: Returns the amount of time the current file or IP stream has been playing (in milliseconds). If `SetLoopMode()` is set to True, the value will not reset when playback loops. If looping playback or IP streaming continues uninterrupted for approximately 25 days, the value will wrap around and become negative.
- `GetDuration() As Integer`: Returns the total playback duration (in milliseconds) of the current file.
- `Seek(position As Integer) As Boolean`: Seeks to the specified position in the audio/video file(measured in milliseconds). If the file is currently playing, then it will continue to play; otherwise, it will remain paused after seeking. This method only supports the MP4/MOV video container; all standard audio formats are supported.
- `SetFade(parameters As roAssociativeArray) As Boolean`: Fades out both the video and audio at the end of the current video file. When the fade begins, an *roVideoEvent* object with the `18 - FadingOut` value will be posted to the message port. This method accepts an associative array, which can currently contain only one parameter:
  - `FadeOutLength`: The length of time (in milliseconds) over which the audio/video fades out. This amount is measured backwards from the end of the video file.

The *ifZorderControl* interface provides the following:

- `ToFront() As Boolean`: Places the video layer of the *roVideoPlayer* instance in front of the other video player.
- `ToBack() As Boolean`: Places the video layer of the *roVideoPlayer* instance behind the other video player.

Note: *This feature is not available on HD players, which only support a single video player. For more information on ordering video layers relative to the graphics layer, refer to the* roVideoMode.SetGraphicsZOrder *entry.*

Timecode Events

You can use the `AddEvent()` method to add triggers for *roVideoEvent* events, which will generate the `12 - Timecode Hit` value at the specified millisecond times in a video file. Use the *roVideoEvent.GetData()* method to retrieve the user data passed with `AddEvent()`.

Example: This script uses timecode events. The script prints 2, 5, and 10 at 2 seconds, 5 seconds, and 10 seconds into the video, respectively. 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
```

Multiscreen video playback

The *roVideoPlayer* object features overloaded `PreloadFile()` and `PlayFile()` functions. These take an *roAssociativeArray*, rather than a string, as a parameter. These alternative methods are used in conjunction with *roSyncManager* to stretch an image across multiple screens in an array or display windowed portions of a video, though they can also be used in place of the original function calls for standard operations.

Example: This script uses the `PreloadFile()` method for 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()
```

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

```
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)
```

To create a multiple-screen matrix in portrait mode, call `SetViewMode(0)` and `SetTransform("rot90")` before calling `PlayFile()`.

Example: This script creates a 3x1 portrait-mode multiscreen display.

```
v=CreateObject("roVideoPlayer")
a=CreateObject("roAssociativeArray")
a["Filename"] = "test.ts"
a["MultiscreenWidth"] = 3
a["MultiscreenHeight"] = 1
a["MultiscreenX"] = 0
a["MultiscreenY"] = 0
v.PreloadFile(a)
v.SetViewMode(0)

v.Play()
```

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`.

Playing Encrypted Files

The *roVideoPlayer* object can be used to play audio/video files that have been encrypted using AES. The associative array passed to the `PlayFile()` method can accept two parameters for file decryption:

- `EncryptionAlgorithm`: The file-encryption algorithm. Currently the options are "AesCtr" and "AesCtrHmac".
- `EncryptionKey`: A byte array consisting of 128 bits of key, followed by 128 bits of IV.

Note: *File decryption is only supported on the 4Kx42, XDx32, XDx30, and HDx22 platforms.*

Example:

```
v = CreateObject("roVideoPlayer")
aa=CreateObject("roAssociativeArray")
aa.filename = "wall-sync2.mp4"
aa.encryptionalgorithm = "AesCtr"
aa.encryptionkey = CreateObject("roByteArray")
aa.encryptionkey.fromhexstring("01030507090b0d0f00020406080a0c0e000000000000000000000000000000000
0")

v.PlayFile(aa)
```

Preferred Streams

If multiple video, audio, or data streams are encapsulated in the video input, you can use the `SetPreferredVideo()`, `SetPreferredAudio()`, and `SetPreferredCaptions()` methods to determine which stream to use. For example, if

a video may contain English and Spanish audio tracks, you can call `SetPreferredAudio()` to specify that the Spanish track should be played if it exists, with the video defaulting to English otherwise.

Preferred streams are chosen by matching the patterns in the passed string(s) against the textual description of the stream:

1. The passed string is a semicolon-separated list of templates.
2. Each template is a comma-separated list of patterns.
3. Each pattern is a `[field_name]=[field_value]` pair that is matched directly against the stream description.

`SetPreferredVideo(description As String) As Boolean`

Each template in the passed video `description` string can contain the following patterns:

- `pid=[integer]`: The packet identifier (PID) of the video stream you wish to display
- `codec=[video_codec]`: The preferred video codec, which can be any of the following:
  - `MPEG1`
  - `MPEG2`
  - `MPEG4Part2`
  - `H263`
  - `H264`
  - `VC1`
  - `H265`
- `width=[integer]`: The preferred video width
- `height=[integer]`: The preferred video height
- `aspect=[float(x.yy)]`: The preferred aspect ratio of the video stream as a floating-point number with two fractional digits.
- `colordepth=[integer]`: The preferred color depth of the video.

Example:

```
"pid=7680, codec=H264, width=1280, height=720, aspect=1.78, colordepth=8;;"
```

`SetPreferredAudio(description As String) As Boolean`

Each template in the passed `description` string can contain the following patterns:

- `pid=[integer]`: The packer identifier (PID) of the audio stream you wish to play
- `codec=[audio_codec]`: The preferred audio codec, which can be any of the following:
  - `MPEG`
  - `MP3`
  - `AAC`
  - `AAC-PLUS`
  - `AC3`
  - `AC3-PLUS`
  - `DTS`
  - `PCM`
  - `FLAC`
  - `Vorbis`
- `channels=[integer]`: The preferred number of audio channels (from `1` to `8`)
- `freq=[frequency]`: The preferred sample frequency of the audio track, which can be any of the following:
  - `32000`
  - `44100`
  - `4800`
- `lang=[language]`: A code that determines the preferred language of the audio track (e.g. `eng`, `spa`). The language codes are specified in the ISO 639-2 standard.
- `type=[audio_type]`: The preferred audio type, which can be one of the following:
  - `Main audio`

- Clean effects
- Hearing impaired
- Visual impaired commentary

Example:

```
“pid=4192, codec=AC3, channels=5, freq=48000, lang=eng, type=Main audio;;"
```

`SetPreferredCaptions(description As String) As Boolean`

Each template in the passed `description` string can contain the following patterns:

- `pid=[integer]`: The packer identifier (PID) of the caption stream you wish to play
- `type=[subtitle_type]`: The encoding standard of the subtitles. This value can be one of the following:
  - `CEA708`: If the CEA-708 standard is not present, the `subtitle_type` will default to CEA-608 (if it is present).
  - `CEA608`
  - `DVB`
- `lang=[language]`: A code that determines the preferred language of the subtitles (e.g. `eng`, `spa`). The language codes are specified in the ISO 639-2 standard.
- `service=[integer]`: The preferred service number of the caption stream

Example:

```
"pid=0, type=Cea708, lang=eng service=1;;"
```

Note the following rules when matching templates to video, audio, or caption stream descriptions:

- For a template to match a stream description, every pattern within the template must match.
- The first listed template to match the stream description (if any) will be used.
- An empty template string will match any stream description.

- All value comparisons are case-insensitive, and all integer values must have no leading zeroes.
- Numerical values must match the stream description exactly. For example, the pattern `pid=016` will never match the stream PID value of 16.
- To indicate logical negation, apply the "!" exclamation mark to the beginning of a pattern. For example, specifying `SetPreferredVideo("!codec=H265")` will match only streams that are not encoded using H.265.
- Apply the ">" greater-than symbol before an integer to indicate that, for a successful match, the value in the stream description must be *greater than* the value following the symbol. For example, specifying `SetPreferredVideo("width=<1921,height=<1081")` will match only videos that are no larger than full-HD.
- Apply the "<" less-than symbol before an integer to indicate that, for a successful match, the value in the stream description must be *less than* the value following the symbol.

The following examples illustrate some of the pattern matching behavior described above:

- The following template list contains three patterns: `lang=eng`, `lang=spa`, and an empty template. The first pattern specifies an English language channel; if the English channel does not exist, the second pattern specifies a Spanish language channel. The third pattern specifies any other channel if the first two don't exist (the empty template matches anything).

```
SetPreferredAudio("lang=eng;lang=spa;;")
```

- Since the following template list is empty, no captions are specified. This can be used to disable captions altogether.

```
SetPreferredCaptions("")
```

- The following template list contains an empty template. Since an empty template matches anything, the first video stream encountered will be played. This is the default behavior of all attributes.

```
SetPreferredVideo(";")
```

- The following template list specifies a 48KHz audio stream if there is one; otherwise, no audio stream will be played. Observe that the list is not correctly terminated with a semicolon; in this case, the semi-colon is implicitly supplied.

```
SetPreferredAudio("freq=48000")
```

- The following template list contains two templates. Note that all patterns within a template must match the stream description for the entire template to match. In this example, an AAC-encoded English track is preferred; an MP3-encoded English track is designated as the second option; and any track will be chosen if neither template is matched.

```
SetPreferredAudio("codec=aac,lang=eng;codec=mp3,lang=eng;;")
```

roTouchEvent, roTouchCalibrationEvent

roTouchEvent

The *roTouchEvent* object is generated by the *roTouchScreen* object whenever a touch or mouse event is detected within a defined region.

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

The *ifInt* interface provides the following:

- `GetInt() As Integer`: Retrieves the region ID of the event.
- `SetInt(a As Integer)`: Sets the region ID of the event.

The *ifPoint* interface provides the following:

- `GetX() As Integer`: Retrieves the x coordinate of the mouse/touch event.
- `GetY() As Integer`: Retrieves the y coordinate of the mouse/touch event.
- `SetX(a As Integer)`: Sets the x coordinate of the event.
- `SetY(a As Integer)`: Sets the y coordinate of the event.

The *ifEvent* interface provides the following:

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

roTouchCalibrationEvent

Interfaces: *ifInt*

The *ifInt* interface provides the following:

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

roTouchScreen

This object accepts inputs from touchscreen panels or mice. For each recognized input, the object will generate an *roTouchScreen* event.

Not all touchscreens are supported. However, we are always working to extend driver support. Please see this FAQ for a full list of supported touchscreens, or contact sales@brightsign.biz if you want to know whether a specific touch-screen model is supported. The *roTouchScreen* object responds to the clicks of a USB mouse in the same way it responds to touch events on a touchscreen.

To set up touchscreen/mouse interactivity, follow this outline:

1. Create an *roTouchScreen* instance.
2. Use `SetPort()` to specify an *roMessagePort* instance to receive the *roTouchScreen* events.
3. Define one or more touch regions.
    a. A touch region may be rectangular or circular.
    b. When a touch or click occurs anywhere inside the area of a touch region, an event will be sent to the message port.
4. Process the events.

Note: *If touch regions overlap such that a click or 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 use the `EnableRollover()` method to add an *on* and *off* image for a region. 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, region_id As Integer) As Void`: Adds a rectangular touch region to the screen. The `region_id` is used to associate the touch region with *roTouchEvent* events and to link the region with rollover images.
- `AddCircleRegion(x As Integer, y As Integer, radius As Integer, region_id As Integer) As Void`: Adds a circular touch region to the screen. The `region_id` is used to associate the touch region with *roTouchEvent* events and to link the region with rollover images.
- `ClearRegions()`: Clears the list of regions added using `AddRectangleRegion()` or `AddCircleRegion()` so that any contacts 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(filename As String, x As Integer, y As Integer) As Void`: Specifies a BMP or PNG file as the mouse cursor icon. This method also accepts a "hot spot" (i.e. the point within the icon rectangle that will trigger events when the mouse is clicked) as a set of **x,y** coordinates. The icon can be a rectangle of any width or height. The colors are specified internally in YUV (6-4-4 bits respectively), but pixels in the passed image file 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).
- `EnableCursor(enable As Boolean) As Void`: Displays a cursor on screen if passed True.
- `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. This method 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. This method 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 regions as 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. Note that the bitmap is square for circular regions. The default origin for circular regions is `x - r, y - r`, where `x, y` is the center of the circle, and `r` is the radius.
- `IsMousePresent() As Boolean`: Returns True if a relative pointing device is attached to the player. This does not work for absolute devices like touchscreens.
- `EnableSerialTouchscreen(a As Integer) As Boolean`
- `SetSerialTouchscreenConfiguration(a As String) As Boolean`
- `GetDiagnosticInfo() As String`: Returns an HTML string with captured information describing hardware that was connected and events that occurred during the calibration process. This method is used by the calibration script to diagnose touchscreen issues.

The *ifSetMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`

The *ifTouchScreenCalibration* interface provides the following:

- `StartCalibration() As Boolean`
- `GetCalibrationStatus() As Integer`
- `GetDiagnosticInfo() As String`
- `ClearStoredCalibration() As Boolean`
- `StartEventLogging() As Boolean`
- `StopEventLogging() As Boolean`

- `ClearEventLogs() As Boolean`
- `SetCalibrationRanges(x-min As Integer, x-max As Integer, y-min As Integer, y-max As Integer) As Boolean`: Overrides the screen range values provided by the touchscreen. This method is useful when the entirety of the video output is not being displayed on the touch surface. Practical use of this method usually requires a custom calibration script, appropriate images, and a calibration setting matched to a particular setup.

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: 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400.
- `NotUsed1(a As String)`
- `SetMode(a As String) As Boolean`
- `NotUsed2(a As Boolean) As Boolean`

Example: This code loops a video and waits for a mouse click or touchscreen 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(True)
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 OBJECTS

roAppendFile

This object can be used to create a new file or append information to the end of an existing file.

`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.

Interfaces: *ifStreamRead*, *ifStreamSend*, *ifStreamSeek*

The *ifStreamRead* interface provides the following:

- `SetReceiveEol(eol_sequence As String) As Void`: Sets the EOL sequence when reading from the stream. The default EOL character is LF (ASCII value 10). If you need to set this value to a non-printing character, use the `chr()` global function.
- `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. The default value is CR+LF. If you need to set this value to a non-printing character, use the `chr()` global function.
- `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(a As Dynamic) As Void`: Writes the specified characters to the stream. This method can support either a string or an *roByteArray*. If the block is a string, 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.

roCreateFile

This object can be used to write a new file or overwrite an existing file.

`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.

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

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. The default value is CR+LF. If you need to set this value to a non-printing character, use the `chr()` global function.
- `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(a As Dynamic) As Void`: Writes the specified characters to the stream. This method can support either a string or an *roByteArray*. If the block is a string, 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.

roReadFile

This object opens and reads a specified file.

Object Creation: Creating an *roReadFile* object opens the specified file for reading only. Object creation fails if the file does not exist.

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

Interfaces: *ifStreamRead*, *ifStreamSeek*

The *ifStreamRead* interface provides the following:

- `SetReceiveEol(eol_sequence As String) As Void`: Sets the EOL sequence when reading from the stream. The default EOL character is LF (ASCII value 10). If you need to set this value to a non-printing character, use the `chr()` global function.
- `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 *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.

roReadWriteFile

The object opens a file and allows both reading and writing operations on that file.

Object Creation: 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.

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

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

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. The default value is CR+LF. If you need to set this value to a non-printing character, use the `chr()` global function.
- `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(a As Dynamic) As Void`: Writes the specified characters to the stream. This method can support either a string or an *roByteArray*. If the block is a string, 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.

HASHING AND STORAGE OBJECTS

roBlockCipher

This object provides a means for symmetric block encryption. It currently supports AES and CBC ciphers, at block sizes of 128, 192, or 256 bits.

Object Creation: The *roBlockCipher* object is created with an associative array representing a set of parameters.

```
CreateObject("roBlockCipher", parameters As roAssociativeArray)
```

The associative array should contain the following parameters:

- `mode`: "aes-128-cbc", "aes-192-cbc", or "aes-256-cbc"
- `padding`: "zero" or "pkcs7". The object defaults to zero padding if this parameter is omitted.

Padding is required for inputs that are not an exact multiple of the cipher block size. Specifying "zero" will add padding only when needed, while specifying "pkcs7" always adds padding, even if the data is already a multiple of the block size (in this case, an entire block will be added). PKCS#7 padding is automatically removed upon decryption, and zero padding will be retained since there are no means to unambiguously distinguish pad values from data.

Interfaces: *ifBlockCipher*

The *ifBlockCipher* interface provides the following:

- `SetIV(iv As Object) As Void`: Sets the Initialization Vector (IV) for CBC (Cipher-Block-Chaining) modes. If the supplied IV is shorter than required, then it will be zero padded (passing an empty string will set the vector to all

zeroes). The IV will typically contain arbitrary characters and be in the form of an *roByteArray*, though it can also be a string.

- `Encrypt(key As Object, plaintext As Object) As roByteArray`: Uses the specified key to encrypt the plaintext parameter, which can be passed as either a string or an *roByteArray*.
- `Decrypt(key As Object, cipher_text As Object) As roByteArray`: Uses the specified key to decrypt cipher text, which should be passed as an *roByteArray*. Because the cipher text is encrypted, it can contain any character.

Example:

```
' This is Case#4 from RFC3602
key = CreateObject("roByteArray")
iv = CreateObject("roByteArray")
plain = CreateObject("roByteArray")
key.FromHexString("56e47a38c5598974bc46903dba290349")
iv.FromHexString("8ce82eefbea0da3c44699ed7db51b7d9")
plain.FromHexString("a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbebfc0c1c2c3c4c5c
6c7c8c9cacbcccdcecfd0d1d2d3d4d5d6d7d8d9dadbdcdddedf")
c = CreateObject("roBlockCipher", { mode: "aes-128-cbc" })
c.SetIV(iv)
crypt = c.Encrypt(key, plain)
result = crypt.ToHexString()
expected =
UCase("c30e32ffedc0774e6aff6af0869f71aa0f3af07a9a31a9c684db207eb0ef8e4e35907aa632c3ffdf868bb7b29d3
d46ad83ce9f9a102ee99d49a53e87f4c3da55")
' Decrypt example to recover the encrypted data
c.SetIV(iv)
roundtrip = c.Decrypt(key, crypt)
```

```
' Second example selecting PKCS#7 padding
c = CreateObject("roBlockCipher", { mode: "aes-128-cbc", padding: "pkcs7" })
```

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).

Object Creation: The *roBrightPackage* object is created with a `filename` parameter that specifies the name of the *.zip* file.

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

Interfaces: *ifBrightPackage*

The *ifBrightPackage* interface provides the following:

- `Unpack(path As String) As Void`: Extracts the zip file to the specified destination path. Any preexisting files in the target directory will be deleted as part of this operation. Providing a destination path of "SD:/" will wipe all preexisting files from the card and extract the *.zip* contents to the root folder.
- `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
```

roDiskErrorEvent

This object is returned while waiting on a message port that is connected to an *roDiskMonitor* object.

Interfaces: *ifUserData, ifDiskErrorEvent*

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.

The ifDiskErrorEvent interface provides the following:

- `GetDiskError() As Object`: Returns an *roAssociativeArray* that contains the following:

Key	Type	Description
`source`	*roString*	The error type
`time`	*roDateTime*	The time at which the error occurred (with millisecond accuracy)
`device`	*roString*	The internal name for the device generating the error
`error`	*roString*	A description of the error (e.g."Timeout")
`param`	*roString*	The error parameter (use depends on type of error; e.g. the sector number)

Example:

```
aa = msgp.GetDiskError()
report = "Time: " + aa["Time"] + "Error: " + aa["source"] + " " + aa["error"] + " " + aa["device"]
+ " " + aa["param"]
```

Note: *This example uses an implicit conversion of* roDateTime. *You could also use* roDateTime.GetString().

roDiskMonitor

This object provides access to low-level information about disk errors. It provides an event-based interface that delivers *roDiskErrorEvent* objects via *roMessageport*. Error messages are held for five seconds before delivery to minimize the chance of spurious error reports. Errors are not reported if the disk is removed during this five second interval because disk-removal detection takes several seconds. This allows for long-term monitoring of occasional media errors.

This object uses the *ifSetMessagePort* interface to select the event destination.

Object Creation: The *roDiskMonitor* object is created with no parameters.

```
CreateObject("roDiskMonitor")
```

Interfaces: *ifMessagePort*, *ifUserData*

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

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:

```
diskmon=CreateObject("roDiskMonitor")

msgp=CreateObject("roMessagePort")
```

```
diskmon.Setport(msgp)
```

roHashGenerator

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

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

```
CreateObject("roHashGenerator", algorithm As String)
```

The algorithm parameter accepts the following strings:

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

Note: *CRC32 is only available on firmware versions 4.4.x 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*.
- `SetHmacKey(key As Dynamic) As Boolean`: Supplies a cryptographic key for the hashing function. This method accepts a plain-text key.
- `SetObfuscatedHmacKey(key As String) As Boolean`: Supplies a cryptographic key for the hashing function. This method accepts a key that is obfuscated using a shared secret.
- `GetFailureReason() As String`:

roPassKey

This object provides a means for generating keys (hashes) from a password and salt.

Object Creation: The object is passed an associative array that specifies the generation methods and cipher.

```
CreateObject("roPassKey", parameters As roAssociativeArray)
```

The associative array should contain the following parameters:

- `method`: The key derivation method. Currently, only "pbkdf2" can be specified.
- `kefn`: The pseudorandom function (PRF). Currently, only "hmac-sha256" can be specified.
- `keylen`: The key length
- `iterations`: The number of iterations

Interfaces: *ifPassKey*

The *ifPassKey* interface provides the following:

- `GenerateKey(password As Object, salt As Object) As roByteArray`: Generates a key using the supplied password and salt. The parameters may be passed as either strings or *roByteArray* instances. The generated *roByteArray* instance may contain all possible byte values, including NUL.
- `GenerateSalt(length As Integer) As roByteArray`: Generates a salt of the specified length. This salt can be used when calling the `GenerateKey()` method. The generated *roByteArray* instance may contain all possible byte values, including NUL.

Example:

```
' Create input test data
salt = CreateObject("roByteArray")
```

```
pass = CreateObject("roByteArray")
pass.FromAsciiString("password")
salt.FromAsciiString("salt")
' Create the key generator
pk = CreateObject("roPassKey",  { method: "pbkdf2", keyfn: "hmac-sha256", keylen: 32, iterations:
4096 } )
' key with be a roByteArray
key = pk.GenerateKey(pass, salt)
```

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. Performing `Read()` on an entry that does not exist, or on a key within a section that does not exist, will return an empty string ("").
- `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 an empty 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 is the main SQLite object that "owns" the database. You can create as many of these objects as you need.

Interfaces: *ifSqliteDatabase*, *ifMessagePort*

The *ifSqliteDatabse* interface provides the following:

- `Open(path As String) As Boolean`: Opens an existing database file. This method returns True upon success.
- `Create(path As String) As Boolean`: Creates a new, empty database file. This method returns True upon success.
- `Close()`: Closes an open database.
- `CreateStatement(sql_text As String) As Object`: Creates a new *roSqliteStatement* object using the specified SQL string.
- `RunBackground(sql_text As String, associative_array As Object) As Integer`: Runs the specified SQL statement in the background and binds variables using the passed *roAssociativeArray*.
- `SetMemoryLimit(limit As Integer)`: Sets the "soft" memory limit under which SQLite will attempt to remain (see the SQLite documentation for details).

Note: *The `SetMemoryLimit()` method set global parameters. It must, therefore, be called before any other calls are made on the database object.*

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`

Example: Creating a Database

```
db = CreateObject("roSqliteDatabase")
```

```
print db

openResult = db.Create("SD:/test.db")

if openResult
    print "Created OK"
else
    print "Creation FAILED"
    end
endif
```

Example: Creating a Table in a Database

```
createStmt = db.CreateStatement("CREATE TABLE playback (md5 text PRIMARY KEY, path PATH,
playback_count INT);")

print createStmt

if type(createStmt) <> "roSqliteStatement" then
    print "We didn't get a statement returned!!"
    end
endif

sqlResult = createStmt.Run()

print sqlResult
```

```
if sqlResult = SQLITE_COMPLETE
    print "Table Created OK"
else
    print "Table Creation FAILED"
endif

createStmt.Finalise()
```

roSqliteEvent

This event object is returned when a `RunBackground()` operation is called by the associated *roSqliteDatabase* object.

Interfaces: *ifSqliteEvent*

The *ifSqliteEvent* interface provides the following:

- `GetTransactionId() As Integer`: Returns an integer that matches the result of the originating `RunBackground()` operation.
- `GetSqlResult() As Integer`: Returns the result code returned by the *roSqliteStatement.Run()* method. The possible return values are identical to the `Run()` method:
  - `100`: Statement complete
  - `101`: Busy
  - `102`: Rows available

Note: *This method can be used as the asynchronous alternative to the `Run()` method.*

roSqliteStatement

This object is created by calling the `CreateStatement()` method on an *roSqliteDatabase* object.

Interfaces: *ifSqliteStatement*

The *ifSqliteStatement* interface provides the following:

Note: *All bind methods return True upon success.*

- `BindByName(associative_array As Object) As Boolean`: Binds the SQL variable(s) using the names contained in the SQL statement.
- `BindByOffset(associative_array/enumerable As Object) As Boolean`: Binds the SQL variable(s) using the index contained in the SQL statement. If passed an associative array, this method will convert the keys of the associative array into numeric offsets when binding. If passed an enumerable object (e.g. *roArray*), it will bind the values of the enumerable in the order that they are stored.
- `BindText(variable/index As Object, value As String) As Boolean`: Binds the SQL variable indicated by the name or index parameter to the passed string value.
- `BindInteger(variable/index As Object, value As Integer) As Boolean`: Binds the SQL variable indicated by the name or index parameter to the passed integer value.
- `Run() As Integer`: Runs the SQL statement immediately and waits for the integer result. The following are possible integer result codes:
  - `100`: Statement complete
  - `101`: Busy
  - `102`: Rows available
- `RunBackground() As Integer`: Runs the SQL statement in the background. You can use *roSqliteDatabase.SetPort()* to set a message port that will receive an *roSqliteEvent* message at a later point. The `RunBackground()` call will result in an integer transaction ID, which will appear in the *roSqliteEvent* message that matches the transaction.

- `GetData() As Object`: Returns an associative array of name/value pairs that are available after a SELECT (or similar) operation.
- `Finalise()`: Finalizes the statement. This method should be applied to statements before the parent database is closed. The object should not be used after this method is called. Also note that objects are automatically finalized when they are deleted.

Example: Inserting into a Table Using `BindByName()`

```
insertStmt = db.CreateStatement("INSERT INTO playback (md5,path,playback_count)
VALUES(:md5_param,:path_param,:pc_param);")

print insertStmt

if type(insertStmt) <> "roSqliteStatement" then
    print "We didn't get a statement returned!!"
    end
endif

params = { md5_param: "ABDEF12346",  path_param: "/foo/bar/bing/bong", pc_param: 11 }

bindResult = insertStmt.BindByName(params)

if bindResult
    print "BindByName OK"
else
    print "BindByName FAILED"
    end
endif
```

```
sqlResult = insertStmt.Run()

print sqlResult

if sqlResult = SQLITE_COMPLETE
    print "Table Insertion OK"
else
    print "Table Insertion FAILED"
endif

insertStmt.Finalise()
```

Example: Inserting into a Table Using `BindByOffset()`

```
insertStmt = db.CreateStatement("INSERT INTO playback (md5,path,playback_count) VALUES(?,?,?);")

print insertStmt

if type(insertStmt) <> "roSqliteStatement" then
    print "We didn't get a statement returned!!"
    end
endif

params = CreateObject("roArray", 3, false)
params[ 0 ] = "ABDEF12345"
params[ 1 ] = "/foo/bar/bing/bong"
```

```
params[ 2 ] = 10

bindResult = insertStmt.BindByOffset(params)

if bindResult
    print "BindByOffset OK"
else
    print "BindByOffset FAILED"
    end
endif

sqlResult = insertStmt.Run()

print sqlResult

if sqlResult = SQLITE_COMPLETE
    print "Table Insertion OK"
else
    print "Table Insertion FAILED"
endif

insertStmt.Finalise()
```

Example: Inserting into a Table in the Background

```
' This examples assume you have set a message port on your roSqliteDatabase instance
'
```

```
insertStmt = db.CreateStatement("INSERT INTO playback (md5,path,playback_count)
VALUES(:md5_param,:path_param,:pc_param);")

print insertStmt

if type(insertStmt) <> "roSqliteStatement" then
    print "We didn't get a statement returned!!"
    end
endif

params = { md5_param: "ABDEF12348",  path_param: "/foo/bar/bing/bong", pc_param: 13 }

bindResult = insertStmt.BindByName(params)

if bindResult
    print "BindByName OK"
else
    print "BindByName FAILED"
    end
endif

expectedId = insertStmt.RunBackground()

e = mp.WaitMessage(10000)
if e <> invalid then
    if type(e) = "roSqliteEvent" then
```

```
        transId = e.GetTransactionId()
        sqlResult = e.GetSqlResult()
        print transId
        print sqlResult
        if transId <> expectedId then
            print "Incorrect transaction Id"
            end
        endif
        if sqlResult <> SQLITE_COMPLETE then
            print "SQL Insertion Failed"
            end
        endif
    else
        print "RunBackground() - Wrong event - FAILED"
        end
    endif
else
    print "RunBackground() - No Response - FAILED"
    end
endif

' You don't need to call Finalise() since that'll be done by the background processor.
```

Example: Querying from a Table

```
selectStmt = db.CreateStatement("SELECT * FROM playback;")
```

```
if type(selectStmt) <> "roSqliteStatement" then
    print "We didn't get a statement returned!!"
    end
endif

sqlResult = selectStmt.Run()

print sqlResult

while sqlResult = SQLITE_ROWS
    resultsData = selectStmt.GetData()
    print resultsData;
    sqlResult = selectStmt.Run()
end while

selectStmt.Finalise()
```

roStorageAttached, roStorageDetached

These event objects are generated by the *roStorageHotplug* object whenever a storage device becomes attached or detached from the player.

Interfaces: *ifUserData*, *ifString*

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.

The *ifString* interface provides the following:

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

roStorageHotplug

This object provides *roStorageAttached* messages when storage devices appear and *roStorageDetached* messages when storage devices disappear. Currently, only external USB devices are supported, and there is no way to poll for media.

Object Creation: The *roStorageHotplug* object is created with no parameters.

```
CreateObject("roStorageHotplug")
```

Interfaces: *ifUserData*, *ifMessagePort*

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.

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

In order to avoid race conditions at startup, you should check for any storage devices that might have existed prior to the message port being set. We recommend doing this after the object is created and the message port is set, but before instructing the script to wait for any events.

Example

```
Sub Main()
    mp = CreateObject("roMessagePort")
```

```
    sh = CreateObject("roStorageHotplug")
    gpio = CreateObject("roControlPort", "brightsign")

    sh.SetPort(mp)
    gpio.SetPort(mp)

    finished = false
    while not finished
    ev = mp.WaitMessage(0)
    if type(ev) = "roControlDown"
        finished = true
    else if type(ev) = "roStorageAttached"
        print "ATTACHED "; ev.GetString()
    else if type(ev) = "roStorageDetached"
        print "DETACHED "; ev.GetString()
    else
        print type(ev)
        stop
    end if
    end while
End Sub
```

roStorageInfo

This object is used to report storage device usage information.

Object Creation: The *roStorageInfo* object is created with a parameter that specifies the path of the storage device. The path does not need to extend to the root of the storage device.

```
CreateObject("roStorageInfo", path As String)
```

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 (in mebibytes) of the storage device.
- `GetUsedInMegabytes() As Integer`: Returns the amount (in mebibytes) of space currently used on the storage device.
  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 (in mebibytes) on the storage device.
- `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"
```

CONTENT MANAGEMENT OBJECTS

roAssetCollection

This object is used to represent a collection of assets.

Object Creation: The *roAssetCollection* object is created with no parameters.

```
CreateObject("roAssetCollection")
```

You can populate an asset collection with individual calls to `AddAsset()` or `AddAssets()`. You can also populate an asset collection using the `roSyncSpec.GetAssets` method, as shown below:

```
assetCollection = CreateObject("roAssetCollection")

localCurrentSync = CreateObject("roSyncSpec")
       if localCurrentSync.ReadFromFile("local-sync.xml") then
              assetCollection = localCurrentSync.GetAssets("download")
       endif
```

Interfaces: *ifAssetCollection, ifInternalAssetCollection*

The *ifAssetCollection* interface provides the following: stuff

- `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.
- `GetAssetList() As roList`: Returns an *roList* instance containing associative arrays of the asset metadata. This method is not efficient and is, therefore, recommended for debugging and diagnostic purposes only.

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 available hash algorithms.
`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_password`	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.

Important: *Any "optional" fields that are specified when populating the pool must also be specified when retrieving assets from the pool (i.e. they become "mandatory" once they are used for an asset). For example, if the* `hash` *value is specified when fetching into the pool, then it must also be specified when attempting to refer to files in the pool.*

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.
`MD5`	Uses the MD5 hash algorithm to validate files.
(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

This object contains functions for downloading files to the pool.

Object Creation: The *roAssetFetcher* object must be passed an *roAssetPool* object upon creation.

```
CreateObject("roAssetFetcher", pool As roAssetPool)
```

Example:

```
Pool = CreateObject("roAssetPool", "pool")
Fetcher = CreateObject("roAssetFetcher", Pool)
```

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

The *ifAssetFetcher* interface provides the following:

- `GetFailureReason() As String`: Returns an error string if an *roAssetFetcher* method has failed (this is usually indicated by returning False). The error string may help diagnose the failure.
- `SetUserAndPassword(user As String, password As String) As Boolean`: Sets the default user and password strings to be used for all download requests that are not otherwise marked using the following attributes: `<auth inherit="no">` or `<auth user="user" password ="password">`.
- `EnableUnsafeAuthentication(enable 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 *roAssetFetcher* 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.

- `EnableUnsafeProxyAuthentication(enable As Boolean) As Boolean`: Supports basic HTTP authentication against proxies if True (which, unlike `EnableUnsafeAuthentication()`, is the default setting). HTTP authentication uses an insecure protocol, which might allow others to easily determine the password. If this method is False, it will refuse to provide passwords via basic HTTP authentication, and any requests requiring this authentication type will fail.
- `EnableEncodings(enable As Boolean) As Boolean`: Enables HTTP compression, which communicates to the server that the system can accept any encoding that the *roAssetFetcher* object is capable of decoding by itself (this behavior is enabled by default). Supported encodings currently include "deflate" and "gzip", which allow for transparent compression of responses. Clients of the *roAssetFetcher* instance see only the decoded data and are unaware of the encoding being used.
- `AsyncDownload(assets As roAssetCollection) As Boolean`: Begins populating the asset pool using the files listed in the passed *roAssetCollection* instance. Files that are not already in the pool will be downloaded automatically. Events are raised during the download process to indicate whether individual file downloads have succeeded or failed; finally, a single event will be raised indicating whether the entire asset collection has been downloaded successfully or not. See the *roAssetFetcherEvent* and *roAssetFetcherProgressEvent* entries for more details.
- `AsyncSuggestCache(a As Object) As Boolean`
- `AsyncCancel() As Boolean`: Cancels any pending “Async” requests. Note that, prior to and during this method call, events associated with an asynchronous action may be queued. No more events will be queued once this call returns. We recommend collecting any pending events prior to calling any further “Async” methods on the same object to avoid confusion.
- `EnablePeerVerification(verification As Boolean)`
- `EnableHostVerification(verification As Boolean)`
- `SetCertificatesFile(filename As String)`
- `AddHeader(name As String, value As String)`: Specifies a header that will be passed to HTTP requests made by the *roAssetFetcher* object. A particular download will not include the header if it has the `<headers inherit=”no”>` attribute in the sync spec.

- `SetHeaders(headers As roAssociativeArray) As Boolean`: Specifies all headers that will be passed to HTTP requests made by the *roAssetFetcher* object. This method removes any previously set headers. A particular download will not include the headers if it has the `<headers inherit="no">` attribute in the sync spec.
- `SetMinimumTransferRate(bytes_per_second As Integer, period_in_seconds As Integer) As Boolean`: Sets the minimum transfer rate for each file download. A transfer will 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 fifteen minutes or more may be appropriate.
- `SetProxy(proxy As String) As Boolean`: Sets the name or address of the proxy server that will be used by the *roAssetFetcher* instance. The proxy string should be formatted as "http://user:password@hostname:port". It can contain up to four "*" characters; each "*" character can be used to replace 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 player will attempt to use a proxy named "proxy-192.168".
- `SetProxyBypass(hostnames As Array) As Boolean`: Exempts the specified hosts from the proxy setting. The passed array should consist of one or more hostnames. The player will attempt to reach the specified hosts directly rather than using the proxy that has been specified with the `SetProxy()` method. For example, the hostname "example.com" would exempt "example.com", "example.com:80", and "www.example.com" from the proxy setting.
- `SetFileProgressIntervalSeconds(interval As Integer) As Boolean`: Specifies the interval (in seconds) between progress events when an individual file is being downloaded. Setting the interval to `-1` disables all progress events. Setting the interval to `0` specifies that events should be generated as often as possible, though this will slow down the transfer process. If the interval is set to `0` or any positive integer, events will always be generated at the start and end of the file download irrespective of elapsed time. The default interval is 300 seconds.
- `SetFileRetryCount(count As Integer) As Boolean`: Specifies the maximum number of times each file download will be retried before moving on to the next file download. The default retry count is five.

- `SetRelativeLinkPrefix(prefix As String) As Boolean`: Specifies a prefix that will be appended to the front of relative links in the sync spec. Normally, this method is used to make file:/// URIs drive agnostic, but it can also be used to reduce the size of the sync spec if all files are stored in the same place. Non-relative links are not affected by this method.
- `BindToInterface(interface As Integer) As Boolean`: Ensures that the HTTP request goes out over the specified network interface (`0` for Ethernet or `1` for WiFi). The default behavior (which can be specified by passing `-1`) is to send requests using the most appropriate network interface, which may depend on the routing metric configured via the *roNetworkConfiguration* object. If both interfaces are on the same layer 2 network, this method may not work as expected due to the Linux weak-host model.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`

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.

roAssetFetcherEvent

This event is generated by an *roAssetFetcher* object when a file transfer succeeds or fails, or when population of the asset pool as a whole succeeds or fails.

Interfaces: *ifAssetFetcherEvent*, *ifUserData*

The *ifAssetFetcherEvent* interface provides the following:

- `GetEvent() As Integer`: Returns an integer indicating the result of an *roAssetFetcher* download attempt:
  - `1`: POOL_EVENT_FILE_DOWNLOADED
  - `-1`: POOL_EVENT_FILE_FAILED
  - `2`: POOL_EVENT_ALL_DOWNLOADED
  - `-2`: POOL_EVENT_ALL_FAILED
- `GetName() As String`
- `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.

The following error codes are generated by the system software and are outside the range of CURL events:

Status	Name	Description
-1002	ENOENT	The specified file does not exist or cannot be created.
-10001	Cancelled	The operation has been cancelled.
-10002	Exception	The operation caused a local exception. Call `GetFailureReason()` for more details.
-10003	ERROR_EXCEPTION	An unexpected exception occurred.
-10004	ERROR_DISK_ERROR	A disk error occurred (usually as a result of the disk being full).
-10005	ERROR_POOL_UNSATISFIED	The expected files are not present in the pool.
-10006	ERROR_DOWNLOADING_ELSEWHERE	The file is being downloaded by another *roAssetFetcher* instance.
-10007	ERROR_HASH_MISMATCH	A downloaded file did not match its checksum or file size.

- `GetFailureReason() As String`: Returns additional failure information associated with the event (if any).
- `GetFileIndex() As Integer`: Retrieves the zero-based index from the sync spec of the file associated with the event.

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.

roAssetFetcherProgressEvent

This event is generated by the *roAssetFetcher* object at regular intervals during file downloads. Use the *roAssetFetcher.SetFileProgressIntervalSeconds()* method to customize how often progress events are generated.

Interfaces: *ifAssetFetcherProgressEvent*, *ifUserData*

The *ifAssetFetcherProgressEvent* interface provides the following:

- `GetFileName() As String`: Returns the name of the file associated with the event. The file name is retrieved from the sync spec associated with the *roAssetFetcher* that generated the event.
- `GetFileIndex() As Integer`: Returns the zero-based index from the sync spec of the file associated with the event.
- `GetFileCount() As Integer`: Returns the total number of files within the sync spec.
- `GetCurrentFileTransferredMegabytes() As Integer`: Returns the number of transferred megabytes belonging to the file associated with the event.
- `GetCurrentFileSizeMegabytes() As Integer`: Returns the size of the file associated with the event.
- `GetCurrentFilePercentage() As Float`: Returns a floating-point number representing the download percentage of the file associated with the event.

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.

roAssetPool

An *roAssetPool* instance 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: The *roAssetPool* object 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(name As String, collection As Object) As Boolean`: Requests that the files specified in the "download" section of a sync spec receive a certain amount of protection. Specified files will not be deleted when the system software needs to reduce the size of the pool to make space.
- `UnprotectAssets(name As String) As Boolean`: Removes the protected status placed on the specified files by the `ProtectAssets()` method. Asset collections are reference counted at the system-software level. As a result, when calling `UnprotectAssets()`, you must pass the same object that you previously passed to `ProtectAssets()`.
- `UnprotectAllAssets() As Boolean`

- `ReserveMegabytes(size As Integer) As Boolean`: Reserves the specified amount of storage space. This method is dynamic: The system software attempts to keep the space free even when parallel processes are filling up the storage.
- `SetMaximumPoolSizeMegabytes(maximum_size As Integer) As Boolean`: Specifies the maximum size of an *roAssetPool* instance in megabytes. This method is more resource-intensive than `ReserveMegabytes()`, but it is useful when creating multiple pools on a storage device.
- `GetPoolSizeInMegabytes() As Integer`
- `Validate(sync_spec As Object, options As roAssociativeArray) As Boolean`: Checks the SHA1, BESHA1, or MD5 hash value of files that are in the sync spec and are currently present in the pool. This method returns True if all checks pass and False if one or more checks fail. Calling `GetFailureReason()` will return information about the corrupt file(s). Note that a True return may not mean that all files in the sync spec are currently present in the pool. The second parameter represents a table of validation options: The key specifies the option and the value specifies whether the option is enabled or not (as a Boolean value). Currently, the only option is "DeleteCorrupt", which determines whether the method should automatically delete corrupt files or not.
- `QueryFiles(a As Object) As Object`
- `AssetsReady(collection As Object) As Boolean`

roAssetPoolFiles

This object works similarly to the *roSyncPoolFiles* object.

Object Creation: The *roAssetPoolFiles* object is created with two parameters.

```
CreateObject("roAssetPoolFiles", pool As Object, assets As Object)
```

The "assets" can be either an *roAssetCollection* or *roSyncSpec* object. If more than one object requires use of the *roAssetCollection* object, it will be more efficient to convert *roSyncSpec* to *roAssetCollection* by calling `GetAssets()` once and then passing that collection to all objects requiring it.

Interfaces: *ifAssetPoolFiles*

The *ifAssetPoolFiles* interface provides the following:

- `GetFailureReason() As String`: Returns explanatory text if `GetPoolFilePath()` returns an empty string or `GetPoolFileInfo()` returns Invalid.
- `GetPoolFilePath(asset_name As String) As String`: Looks up the specified file name in the asset collection and uses the information to determine the actual name of the file in the pool. This method returns an empty string if the name is not found in the asset collection, or if the file is not found in the pool.
- `GetPoolFileInfo(asset_name As String) As Object`: Looks up the specified file name in the asset collection and returns all available information, including the pool file path, as an associative array. This method returns Invalid if the asset name is not found in the asset collection. If the file is not found in the pool, information from the asset collection will be returned without the pool path. See the table below for a description of assets in the associative array.

Field	Value	Description
name	String	Asset name
link	String	Asset URL
size	String	
hash	String	Hash in algorithm ":" hash format
change_hint	String	Only present if set
auth_user	String	Only present if set
auth_password	String	Only present if set
auth_inherit	Boolean	
headers_inherit	Boolean	
probe	String	Probe data
path	String	Absolute path of the file in the pool (or "invalid" if the file is not in the pool)

roAssetRealizer

This object contains functions for realizing files.

Object Creation: The *roAssetRealizer* object requires two parameters upon creation: an *roAssetPool* object and a destination directory.

```
CreateObject("roAssetRealizer", pool As roAssetPool, destination_directory As String)
```

Example:

```
pool = CreateObject("roAssetPool", "pool")
realizer = CreateObject ("roAssetRealizer", pool, "/")
```

Interfaces: *ifUserData*, *ifAssetRealizer*

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.

The *ifAssetRealizer* interface provides the following:

- `GetFailureReason() As String`: Yields additional useful information if a function return indicates an error.
- `EstimateRealizedSizeInMegabytes(spec As Object) As Integer`: Returns the estimated amount of space that would be taken up by the specified sync spec.
- `Realize(spec As roSyncSpec/roAssetCollection) As Object`: Places the files into the destination directory specified in the passed *roSyncSpec* or *roAssetCollection*. If the pool does not contain the full set of

required files, then this method will immediately fail before any files are changed (this method will always attempt to do as much work as possible before destructively modifying the file system). This method automatically checks the length of the file and any hashes that match the specification. If there is a mistmatch, then the affected file is deleted and realization fails. This method indicates success or failure by returning an *roAssetRealizerEvent* object.

Note: *The pool and the destination must be in the same file system.*

- `ValidateFiles(spec As Object, options As Object) As Object`: Checks the hash of every file in the spec against the corresponding file in the destination path and returns an associative array containing each mismatched fle name mapped to the reason. The options parameter is an *roAssociativeArray*, which can currently support a single option:
    - `"DeleteCorrupt"`: Automatically deletes any files that do not match the expected hash. By default, these hashes are not deleted.

roAssetRealizerEvent

This event object is returned when the *roAssetRealizer.Realize()* method is called. It yields information about the success or failure of the realization process.

Interfaces: *ifAssetRealizerEvent*, *ifUserData*

The *ifAssetRealizerEvent* interface provides the following:

- `GetEvent() As Integer`: Returns an integer value indicating the type of the event:

`101`	`EVENT_REALIZE_SUCCESS`	The specified sync list was successfully realized.
`-102`	`EVENT_REALIZE_INCOMPLETE`	Realization could not begin because at least one of the required files is not available in the pool.
`-103`	`EVENT_REALIZE_FAILED_SAFE`	Realization has failed. Nothing has been written to the destination, so it is likely safe to continue the realization process. More information is about the failure is available via the `GetFailureReason()` and `GetName()` methods.
`-104`	`EVENT_REALIZE_FAILED_UNSAFE`	Realization has failed while running, and changes have been made to destination files. It may not be safe to continue the realization process. More information about the failure is available via the `GetFailureReason()` and `GetName()` methods.

- `GetName() As String`: Retrieves the name of the affected file if the realization process fails.
- `GetResponseCode() As Integer`: Retrieves the *roUrlTransfer* response code associated with the event (if any).
- `GetFailureReason() As String`: Returns additional information if the realization process fails.
- `GetFileIndex() As Integer`: Retrieves the zero-based index number of the the file in the sync spec.

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.

roSyncSpec

This object represents a parsed sync spec. It allows you to retrieve various parts of the specification with methods.

Interfaces: *ifSyncSpec, ifInternalSyncSpec, ifInstanceFromStream*

The *ifSyncSpec* interface provides the following:

- `GetFailureReason() As String`: Returns information if an *roSyncSpec* method indicates failure.
- `ReadFromFile(filename As String) As Boolean`: Populates the sync spec by reading the specified file. This method returns True upon success and False upon failure.
- `ReadFromString(spec As String) As Boolean`: Populates the sync spec by reading the passed string. This method returns True upon success and False upon failure.
- `WriteToFile(filename As String) As Boolean`: Writes out the current sync spec to the specified file. Because the XML is regenerated, it is possible this file may not be textually identical to the specification that was read. This method returns True upon success and False upon failure.
- `WriteToString() As String`: Writes out the current sync spec to a string and returns it. This method returns an empty string if the write operation fails.
- `GetMetadata(section As String) As roAssociativeArray`: Returns an *roAssociativeArray* object containing the information stored in the specified metadata section of the sync spec (typically "client" or "server"). This method returns 0 if the read operation fails.
- `LookupMetadata(section As String, field As String) As String`: Provides a shortcut for looking up specified metadata items in the specified section without needing to create a temporary *roAssociativeArray* object. This method returns an empty string if the read operation fails.
- `GetFileList(section As String) As roList`: Returns an *roList* object containing *roAssociativeArray* objects for each file in the specified section of the sync spec. This method returns Invalid if the read operation fails.

- `GetFile(section As String, index As Integer) As roAssociativeArray`: Returns an *roAssociativeArray* object for the file in the specified section and at the specified index. This method returns Invalid if the read operation fails.
- `GetName() As String`: Returns the name supplied for the sync spec in the `<sync>` XML element.
- `EqualTo(other As roSyncSpec) As Boolean`: Compares the contents of the *roSyncSpec* object with another *roSyncSpec* object. This method compares the parsed contents of each sync spec rather than the XML files themselves.
- `VerifySignature(signature as String, obfuscated_passphrase as String) As Boolean`: De-obfuscates the passphrase and uses it to verify the signature of the sync spec. This method returns True upon success and False upon failure.
- `FilterFiles(section As String, criteria As roAssociativeArray) As roSyncSpec`: Returns a new *roSyncSpec* object that is a copy of the existing object, except that the specified section is filtered using the specified criteria. The criteria are matched against the file metadata. Multiple criteria can be specified in the passed associative array, and all criteria must be met for a file to be returned with the new *roSyncSpec*.

Example: The following function will yield an *roSyncSpec* object with a "download" section that has been filtered so that only files of the group "scripts" will remain.

```
filtered_spec = spec.FilterFiles("download", { group: "scripts" })
```

- `FilesEqualTo(a As Object) As Boolean`
- `GetAssets(a As String) As Object`

NETWORKING OBJECTS

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: The *roDatagramSender* object 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 sends *roDatagramEvent* instances to a message port when UDP packets are received on a specified port.

Object Creation: The *roDatagramReceiver* object is created with a single parameter. The `port` paremeter specifies the port on which to receive UDP packets.

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

Interfaces: *ifIdentity*, *ifMessagePort*

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

The *ifMessagePort* 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(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.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`

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(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.

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

This event object is used to handle requests generated by the *roHttpServer* object.

Interfaces: *ifHttpEvent*, *ifUserData*

The *ifHttpEvent* interface provides the following:

- `GetFailureReason() As String`: Yields additional useful information if a function return indicates an error.
- `GetMethod() As String`: Returns the type of HTTP method that triggered the event on the *roHttpServer* instance.
- `SetResponseBodyString(body As String)`: Sets the response body for an event generated via the `AddGetFromEvent()` or `AddMethodToString()` method on the *roHttpServer* object. This call is ignored with any other event.
- `SetResponseBodyFile(filename As String) As Boolean`: Specifies the name of a file to use as the source response body for an event generated via the `AddGetFromEvent()` or `AddMethodToString()` method on the *roHttpServer* object. This call is ignored with any other event. This function will return False if the file cannot be opened or another failure occurs.

Note: *The specified 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()*. An empty string is returned with any other event.
- `GetRequestBodyFile() As String`: Returns the name of the temporary file created if the event is generated via *roHttpServer.AddGetFromEvent*.This call is ignored with any other event.
- `GetRequestHeader(header_name 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(URI_parameter 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(header As String, value 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`. Adds the specified HTTP header/value pairs to the response. This method expects an *roAssociativeArray* 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, though a subset of headers to might be set before the failure occurs. This function returns True upon success.
- `SendResponse(http_status_code As Integer) As Boolean`: Sends the HTTP response using the specified HTTP status code. To ensure that the response is sent, this function needs to be called once the script has finished handling the event. This function returns False upon failure.
- `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(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.

roHttpServer

This object allows for processing of RESTful HTTP requests from remote URLs to the embedded web server of the BrightSign player. Many of the requests are provided to the script as *roHttpEvent* objects for handling.

Object Creation: The *roHttpServer* object is created with an *roAssociativeArray*.

```
CreateObject("roHttpServer", parameters As roAssociativeArray)
```

Currently, the associative array can contain a single key:value pair:

- `port`: The port number of the embedded web server

Interfaces: *ifHttpServer*, *ifMessagePort*, *ifGetMessagePort*

The *ifHttpServer* interface provides the following:

Note: *Each "Add" handler method described below takes an associative array as its parameter. Values in the associative array specify how the handler behaves. See the table at the end of this section for common key:value pairs.*

- `GetFailureReason() As String`: Yields additional useful information if an *roHttpServer* method fails.
- `AddGetFromString(parameters As roAssociativeArray) As Boolean`: Causes any HTTP GET requests for the specified URL path to be met directly with the contents of the "body" member of the parameter associative array. The MIME type (and potentially the entire character set) should be specified if the request is expected to come from a web browser. The request is handled entirely within the *roHttpServer* method; no events are sent to the message port.
- `AddGetFromFile(parameters As roAssociativeArray) 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. The request is handled entirely within the *roHttpServer* method; no events are sent to the message port.

- `AddGetFromEvent(parameters As roAssociativeArray) As Boolean`: Requests that an event of type *roHttpEvent* be sent to the configured message port. This occurs when an HTTP GET request is made for the specified URL path.
- `AddPostToString(parameters As roAssociativeArray) As Boolean`: Requests that an event of type *roHttpEvent* be sent to the configured message port. This occurs when an HTTP POST request is made for the specified URL path. Use the *roHttpEvent.GetRequestBodyString()* method to retrieve the posted body.
- `AddPostToFile(parameters As roAssociativeArray) As Boolean`: Requests that, when an HTTP POST request is made to the specified URL path, the request body be stored in a temporary file according to the `parameters["destination_directory"]` value in the associative array. When this request is complete, an *roHttpEvent* event is sent to the configured message port. Use the *roHttpEvent.GetRequestBodyFile()* method 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. However, if the player reboots or loses power during the POST process, the file will not be deleted. For this reason, we recommend using a dedicated subdirectory as the "`destination_directory`" and wiping this subdirectory during startup (using `DeleteDirecotry()`) before adding handlers that refer to it.
- `AddPostToFormData(parameters As roAssociativeArray) 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 the *roHttpEvent.GetFormData()* method.
- `AddMethodFromEvent(parameters As roAssociativeArray) As Boolean`: Requests that an event of type *roHttpEvent* be sent to the configured message port. Unlike `AddGetFromEvent()`, this method can support arbitrary HTTP methods. The HTTP method is specified using the `method` member in the associative array.
- `AddMethodToFile(parameters As roAssociativeArray) As Boolean`: Requests that, when an arbitrary HTTP request is made to the specified URL path, the request body be stored in a temporary file according to the `parameters["destination_directory"]` value in the associative array. The HTTP method is specified using the `method` member in the associative array. When the request is complete, an *roHttpEvent* event is sent to

the configured message port. Use the *roHttpEvent.GetRequestBodyFile()* method 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.

- `AddMethodToString(parameters As roAssociativeArray) As Boolean`: Attempts to support an arbitrary HTTP method. The request body is placed in a string and an event is raised. This makes the request body available via the *roHttpEvent.GetRequestBodyString()* method. A response can be sent in the same manner as the `AddGettoEvent` method.
- `SetupDWSLink(parameters As Dynamic) As Boolean`: Generates a tab in the Diagnostic Web Server (DWS) that links to files hosted by the *roHttpServer* instance. The tab will function differently depending on the parameters passed to the method.
    - `title As String`: Generates a tab with `title` that links directly to the base `ip_address:port` of the *roHttpServer* instance.
    - `{title:[title As String], tab:"no"}`: Generates a tab with `title` that links to a DWS page. This page will contain a single link to the base `ip_address:port` of the *roHttpServer* instance.
    - `{title:[title As String], link1:[params As roAssociativeArray], link2:[params As roAssociativeArray], ...,}`: Generates a tab with `title` that links to a DWS page. This page can contain any number of links to files hosted by the *roHttpServer* instance. Each link is configured with an associative array containing the following key:value pairs:
        - `name`: The name of the link on the DWS page
        - `url`: The `url_path` of the file hosted by the *roHttpServer* instance (see the table below for details)

Example:

```
server1.SetupDWSLink("My AWS Link")

server2.SetupDWSLink({ title: "My AWS Link", tab: "no" })

server3.SetupDWSLink({ title: "Link> ", link1: { name: "Name 1", url: "/mylink1.jpg" }, link2: {
name: "Name 2", url: "/mylink2.jpg" } })
```

Associative Array Parameters for "Add" Handler Methods

The following table describes common key:value pairs for "Add" handler methods:

Name	Applies to	Value
`url_path`	`all`	The path for which the handler method will be used
`user_data`	`GetFromEvent(),PostToString(), PostToFile(),MethodToString()`	A user-defined value that can be retrieved by calling *roHttpEvent.GetUserData()*
`method`	`AddMethodFromEvent(), AddMethodToFile()`	The HTTP method associated with the generated *roHttpEvent*. The method type can then be retrieved using *roHttpEvent.GetMethod()*.
`passwords`	`all`	An associative array that contains a mapping between usernames and passwords
`auth`	`all`	The authentication type to use when passwords are set. This value can be either "basic" or "digest". The value defaults to “digest” if not specified.
`realm`	`all`	The authentication realm, which will be displayed by web browsers when prompting for a username and password
`headers`	`GetFromFile`	An associative array that contains arbitrary headers to be included with the automated response
`content_type`	`GetFromFile`	The contents of the "Content-Type" header that is included with the automated response. This may not be set at the same time as the `headers` member. You can set both the MIME type and character set together (e.g. "`text/plain; charset=utf-8`")
`body`	`GetFromString`	The response body
`filename`	`GetFromFile`	The path to the file used for the response body.
`destination_directory`	`PostToFile`	The path to the directory used for the temporary file containing the request body. A random filename will be generated automatically.

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

roMediaServer

The *roMediaServer* object waits for client requests, deals with negotiation, and ultimately generates an *roMediaStreamer* pipeline to fulfill the request. For more information, see the Media Server Developer's Guide on the documentation page. This object currently supports RTSP and HTTP requests. Requests from the client must take the following form:

```
protocol://IP_address:port/media_streamer_pipeline
```

- `protocol`: Either `rtsp` or `http`
- `IP_address:port`: The IP address of the BrightSign player and the port number on which the media server is running.
- `media_streamer_pipeline`: A media streamer pipeline, but without the final destination component (as the destination is implicit in the request from the client).

Object Creation: The *roMediaServer* object is created with no parameters

```
CreateObject("roMediaServer")
```

Interfaces: *ifMediaServer*, *ifIdentity*, *ifUserData*, *ifMessagePort*

The *ifMediaServer* interfaces provides the following:

- `GetFailureReason() As String`: Returns useful information if the `Start()`, `Stop()`, or `Terminate()` methods return False.
- `Start(a As String) As Boolean`: Begins a media server instance. This method can be passed a string that specifies the streaming protocol and the port number of the server:

```
s = CreateObject("roMediaServer")
s.Start("http:port=8080")
```

A number of optional parameters can be added after the `port` parameter using an "&" (ampersand):

- `trace`: Displays a trace of messages in the negotiation with the client. This parameter is useful particularly for debugging RTSP sessions. For example: `"rtsp:port=554&trace"`
- `maxbitrate`: Sets the maximum instantaneous bitrate (in Kbps) of the RTP transfer initiated by RTSP. This parameter has no effect for HTTP. The parameter value 80000 (i.e. 80Mbps) has been found to work well. The default behavior (also achieved by passing a zero value) is to not limit the bitrate at all. For example: `"rtsp:port=554&trace&maxbitrate=80000"`
- `threads`: Sets the maximum number of threads the server is prepared to have running. Each thread handles a single client request. The default value is "5". For example: `"http:port=8080&threads=10"`

- `Stop() As Boolean`: Stops the media server. This method signals all threads to stop, but does not wait for this to happen before destroying the server instance.
- `Terminate() As Boolean`: Stops the media server. This method waits for all threads to stop before destroying the server instance.

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

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.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`

roMediaStreamer

The current implementation of this object allows a player to stream *.ts* files over UDP and RTP. For more information, see the Media Server manual on the documentation page.

Object Creation: The *roMediaStreamer* object is created with no parameters.

```
CreateObject("roMediaStreamer")
```

Interfaces: *ifMediaStreamer*, *ifUserData*, *ifMessagePort*

The *ifMediaStreamer* interface provides the following:

- `GetFailureReason() As String`
- `SetPipeline(pipeline As String) As Boolean`: Specifies a streaming pipeline. The source (a file URI) and destination (an IP address) of the stream are specified in the passed stream. This method replaces the `SetSource()` and `SetDestination()` methods from firmware version 4.7. To stream media as before, use the `filesimple` source designation and the `udpsimple`/`rtpsimple` destination designations:

```
m = CreateObject("roMediaStreamer")
m.SetPipeline("filesimple:///data/clip.ts, udpsimple://239.192.0.0:1234/")
m.Start()
```

- `Initialize() As Boolean`: Progresses the pipeline into the INITIALIZED state. This allocates some resources for the pipeline, but does not begin a stream.
- `Connect() As Boolean`: Progresses the pipeline into the CONNECTED state. This allows the script to create a memory stream without starting it.
- `Start() As Boolean`: Begins streaming.
- `Stop() As Boolean`: Stops the pipeline stream. Some internal pipeline stages may continue running.

- `Disconnect() As Boolean`: Regresses the steam back to the CONNECTED state.
- `Reset() As Boolean`: Resets the pipeline stream. All internal pipeline stages are terminated.
- `Inject(a As Integer) 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.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`

Source Specifications

The string passed to the *roMediaStreamer.SetPepline()* method can have unique parameters that determine the source type and playback behavior.

- **Looping**: By default, a stream from a media file will not loop when it ends. You can specify a looping parameter at the end of the source string as follows: """filesimple:///data/example.mp4?loop". It is also possible to loop the stream using end-of-stream messages from *roMediaStreamerEvent*. However, the slightly longer restart gap that results from using BrightScript may cause problems with the streaming client. This is especially true if you attempt to set a new media file source upon looping the function.

roMediaStreamerEvent

This object is sent by instances of *roMediaStreamer*. It provides information about the current state of an IP stream being sent by the player.

Interfaces: *ifUserData*, *ifMediaStreamerEvent*

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.

The *ifMediaStreamerEvent* interface provides the following:

- `GetEvent() As Integer`: Returns an integer describing the status of an *roMediaStreamer* instance:
  - `0 - EOS_NORMAL`: The end of the stream has been reached without any errors being detected. This signal is not sent if the `loop` parameter is specified using the `roMediaStreamer.SetSource()` method.
  - `1 - EOS_ERROR`: The stream has been aborted prematurely because of an error condition.

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 shown in the following example.

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

Interfaces: *ifPictureStream*, *ifUserData*, *ifMessagePort*

The *ifPictureStream* interface provides the following:

- `GetUrl() As String`

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.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`: 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: The *roNetworkAdvertisement* object is created with an associative array representing network parameters.

```
CreateObject("roNetworkAdvertisement", advertisement As roAssociativeArray) As Object
```

The *roAssociativeArray* can contain 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

Example

```
di = CreateObject("roDeviceInfo")
 props = { name: "My Hoopy Service", type: "_http._tcp", port: 8080, _serial:
di.GetDeviceUniqueId() }
 advert = CreateObject("roNetworkAdvertisement", props)
```

```
...
' Stop advertising
advert = invalid
```

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*

The *ifInt* interface provides the following:

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

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

This object provides various interfaces for configuring the network interfaces on a BrightSign player.

Object Creation: The *roNetworkConfiguration* object is created with a single parameter.

```
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 player.
- 1 for the optional internal Wi-Fi.

Note: *Some of the settings are specific to the network interface, while others are used by the BrightSign host for all network interfaces.*

Interfaces: *ifNetworkConfiguration*, *ifWiFiConfiguration*

The *ifNetworkConfiguration* interface provides the following:

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

- `SetupDWS(settings As roAssociativeArray) As Boolean`: Configures the Diagnostic Web Server (DWS). By default, the Diagnostic Web Server is enabled on port 80, with the player serial number as password. Settings for the DWS are specified in an associative array. These properties are written to the registry and persist after reboot:
  - `port`: The port number of the Diagnostic Web Server, located at the IP address of the player. Setting this value to 0 will disable the DWS, while setting it to "default" will make the DWS accessible on the default port (80). Specifying *only* this parameter in the associative array is equivalent to enabling the DWS without password protection.

    - `password`: An obfuscated password for the DWS. This method uses digest access authentication. Specifying this parameter without setting a `port` number will make the DWS accessible on the default port.
    - `open`: An unobfuscated password for the DWS. This method uses digest access authentication. Specifying this parameter without setting a `port` number will make the DWS accessible on the default port.
    - `basic`: A flag indicating whether basic authentication should be used or not. Setting this parameter to True allows the password set with the `open` parameter to be validated using basic authentication, rather than digest access authentication. This option allows for backwards compatibility with older platforms; most, if not all, modern browsers require basic authentication to be disabled in order to communicate with the DWS.

  Note: *The user name is "admin" for all authentication configurations.*
- `EnableLEDs(enable As Boolean) As Boolean`: Enables or disables the Ethernet activity LED (i.e. flashing during link and activity behavior). The Ethernet LED is enabled by default. Changes to this setting do not persist across reboots. This method returns True upon success and False upon failure. Note that this method is not available on HDx10, HDx20, and LSx22 models.
- `GetClientIdentifier() As String`
- `GetProxy() As String`
- `SetClientIdentifier(a As String) As Boolean`
- `SetLoginPassword(password As String)`: Specifies a login password for the SSH connection (if SSH has been enabled in the registry). This method accepts a plain-text password.
- `SetObfuscatedLoginPassword(password As String)`: Specifies a login password for the SSH connection (if SSH has been enabled in the registry). This method accepts a password that has been obfuscated using a shared secret.
- `SetInboundShaperRate(rate As Integer) As Boolean`: Sets the bandwidth limit for inbound traffic in bits per second. For the default bandwidth limit, pass `-1` to the method; for no bandwidth limit, pass `0` (though these two settings are functionally the same). You will need to call `Apply()` for this setting to take effect, and changing this setting at any time will cause the network interface to be taken down and reinitialized.

Note: *Because of overhead on the shaping algorithm, attempting to limit the bandwidth at rates greater than approximately 2Mbit/s will reduce speeds to less than the specified rate.*

- `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. This method returns True on success.
- `SetWiFiPassphrase(passphrase as String) as Boolean`: Configures the passphrase or key for the wireless network. This method accepts a plain-text password. It returns True if the password is successfully set.
- `SetObfuscatedWiFiPassphrase(password As String) As Boolean`: Configures the passphrase or key for the wireless network. This method accepts a password that has been obfuscated using a shared secret. It returns True if the password is 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("")`. You can use URL syntax to specify that the player use an HTTP or HTTPS server to synchronize the clock. The following are valid time server addresses:
  - `http://time.brightsignnetwork.com/`
  - `https://time.brightsignnetwork.com/`
  - `ntp://time.brightsignnetwork.com/`
  - `time.brightsignnetwork.com`

  Note: *The last two addresses are equivalent.*
- `GetTimeServer() As String` (host): Retrieves the time server currently in use.
- `SetTimeServerIntervalSeconds(interval_in_seconds As Integer) As Boolean`: Specifies how often the player should communicate with the time server and adjust its clock via NTP. The default interval is 12 hours; passing a value of `0` specifies the default interval. The minimum interval allowed is 120 seconds.
- `GetTimeServerIntervalSeconds() As Integer`: Returns the current interval for NTP time-server renewal (in seconds).
- `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. Passing an empty string to this method resets the device host name to its automatically generated value.

- `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. The proxy string should be formatted as "http://user:password@hostname:port". It can contain up to four "*" characters; each "*" character can be used to replace 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 player will attempt to use a proxy named "proxy-192.168".
- `SetProxyBypass(hostnames As Array) As Boolean`: Exempts the specified hosts from the proxy setting. The passed array should consist of one or more hostnames. The player will attempt to reach the specified hosts directly rather than using the proxy that has been specified with the `SetProxy()` method. For example, the hostname "example.com" would exempt "example.com", "example.com:80", and "www.example.com" from the proxy setting.
- `GetProxyBypass() As roArray`: Returns an array of hostnames that have been exempted from the proxy setting using the `SetProxyBypass()` method.
- `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* of Strings	Host	The currently active DNS servers
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.
shape_inbound	Integer	Interface	The current bandwidth shaping for inbound traffic determined by the SetInboundShaperRate() method.
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	This value is True if the tests find no problems, or 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* of 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	This value is True if the tests find no problems, or 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* of strings	A complete log of all the tests performed and their results.

- `GetNeighborInformation() As roAssociativeArray`: Retrieves location information from the network infrastructure using the LLDP-MED protocol. The information is returned as an associative array of strings corresponding to civic-address types, which are defined as follows according to the LLDP-MED specification:

CAtype	Label	Description
1	A1	national subdivisions (state, region, province, prefecture)
2	A2	county, parish, gun(JP), district(IN)
3	A3	city, township, shi(JP)
4	A4	city division, borough, city district, ward, chou(JP)
5	A5	neighborhood, block
6	A6	street

CAtype	NENA	PIDF	Description	Examples
0			language	`i-default [3]`
16	PRD	PRD	leading street direction	`N`
17	POD	POD	trailing street suffix	`SW`
18	STS	STS	street suffix	`Ave, Platz`
19	HNO	HNO	house number	`123`
20	HNS	HNS	house number suffix	`A, 1/2`
21	LMK	LMK	landmark or vanity address	`Columbia University`
22	LOC	LOC	additional location information	`South Wing`
23	NAM	NAM	name (residence and office occupant)	`Joe's Barbershop`
24	ZIP	PC	postal/ZIP code	`10027-1234`
25			building (structure)	`Low Library`
26			unit (apartment, suite)	`Apt 42`
27		FLR	floor	`4`

28			room number	`450F`
29			placetype	`office`
30	PCN		postal community name	`Leonia`
31			post office box (P.O Box)	`12345`
32			additional code	`13203000003`
128			script	`Latn`
255			reserved	

The *ifWiFiConfiguration* interface provides the following:

- `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.

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.*

To determine which network was attached or detached, the script needs to call *roNetworkAttached.GetInt* or *roNetworkDetached.GetInt*. These methods provide an index of the network interface that was attached or detached.

Interfaces: *ifMessagePort*, *ifUserData*

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

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.

roNetworkStatistics

This object allows you to monitor and post how much bandwidth the player is using.

Object Creation: The *roNetworkStatistics* object is created with a single parameter.

```
CreateObject("roNetworkStatistics", 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 player.
- 1 for the optional internal Wi-Fi.

Interfaces: *ifNetworkStatistics*

The *ifNetworkStatistics* interface provides the following:

- `GetTotals() As roAssociativeArray`: Yields the total network figures since booting up.
- `GetIncremental() As roAssociativeArray`: Yields the total network figures since booting up. Then, every subsequent time this method is called, it will yield the amount each figure has changed since the previous call.

Note: *If multiple instances of* roNetworkStatistics *are created,* `GetIncremental()` *calls for each instance will track changes independently.*

Both methods return the following statistics as floating point values:

- `tx_carrier_errors`
- `tx_packets`
- `rx_packets`
- `tx_errors`
- `rx_frame_errors`

- tx_bytes
- rx_errors
- tx_collisions
- rx_dropped
- tx_compressed
- rx_multicast
- tx_dropped
- rx_fifo_errors
- rx_bytes
- tx_fifo_errors
- rx_compressed

roPtp

This object can be used to retrieve information about the network PTP state of the player.

Object Creation: This object is created with no additional parameters.

```
ptp = CreateObject("roPtp")
```

Interfaces: *ifUserData*, *ifMessagePort*, *ifPtp*

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.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`: Posts messages of the type *roPtpEvent* to the attached message port.

The *ifPtp* interface provides the following:

- `GetPtpStatus() As roAssociativeArray`: Returns an associative array containing information about the network PTP state of the player:
  - `state`: A string indicating the current PTP state of the player. Values can be "MASTER", "SLAVE", or "UNCALIBRATED".
  - `timestamp`: A value indicating when the PTP state was last changed. This value is measured in seconds since the player booted. This value can be compared against the total uptime of the player, which is retrieved by calling `UpTime(0)`.

roPtpEvent

This event object is generated by the *roPtp* object whenever the PTP status of the player changes.

Interfaces: *ifUserData*, *ifPtpEvent*

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.

The *ifPtpEvent* interface provides the following:

- `GetPtpStatus() As roAssociativeArray`: Returns an associative array containing information about the network PTP state of the player:
  - `state`: A string indicating the current PTP state of the player. Values can be "MASTER", "SLAVE", or "UNCALIBRATED".
  - `timestamp`: A value indicating when the PTP state was last changed. This value is measured in seconds since the player booted. This value can be compared against the total uptime of the player, which is retrieved by calling `UpTime(0)`.

roRssParser, roRssArticle

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

Object Creation: The *roRssParser* and *roRssArticle* objects are 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 an *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(a As Integer) As Boolean`: 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:

Note: *For firmware versions 4.7.x and above, if no alpha value is specified when roTextWidget.SetForegroundColor() is called, the text widget area will appear blank.*

```
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(&hFFD0D0D0)
t.Show()

a = r.GetNextArticle()
while type(a) = "roRssArticle"
       t.PushString(a.GetDescription())
       sleep(1000)
       a = r.GetNextArticle()
end while

while true
       sleep(1000)
end while
```

roRtspStream

This is a simple media-streaming object that is passed to the *roVideoPlayer.PlayFile()* method. Use this object to play UDP, RTP, HLS, and HTTP streams. See the Video Streaming and IP Camera FAQs for more details.

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

```
v = createobject("rovideoplayer")
r = createobject("rortspstream", "http://172.30.1.37/alldigital/1080p/playlist.m3u8")
v.playfile({rtsp:r})
```

Note: *The key in the passed associative array will always be `rtsp`, no matter which streaming protocol is used.*

Interfaces: *ifRtspStream*, *ifMessagePort*, *ifUserData*

The *ifRtspStream* interface provides the following:

- `GetUrl() As String`: Retrieves the currently configured URL.
- `AddHeader(header As String, text As String)`: Adds the specified header and header text to the streaming request. The ":" after the header and the "\r\n" after the header text are supplied automatically by the method. Headers only take effect when a stream is played; you cannot add more headers when a stream is playing (though these headers will be applied if the stream is played again).
- `ClearHeaders()`: Removes headers that have been added using the `AddHeader()` method.

The *ifMessagePort* interface provides the following:

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

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.

roShoutcastStream

Object creation: The *roShoutcastStream* object 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*, *ifMessagePort*, *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 *ifMessagePort* 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*, *ifUserData*, *ifMessagePort*

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 *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.

The *ifMessagePort* interface provides the following:

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

roSnmpEvent

Interfaces: *ifUserData*, *ifString*

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.

The *ifString* interface provides the following:

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

roStreamByteEvent

Interfaces: *ifInt*, *ifUserData*

The *ifInt* interface provides the following:

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

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.

roStreamConnectResultEvent

This event is sent to a message port associated with an *roTCPStream* object when an `AsyncConnectTo()` request has been completed or has failed.

Interfaces: *ifInt*, *ifUserData*

The *ifInt* interface provides the following:

- `GetInt() As Integer`: Returns the result code of the event. If the connection was successfully established, then this method will return 0. If connection failed for any reason, this method will return a non-zero integer.
- `SetInt(a As Integer)`

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.

roStreamEndEvent

Interfaces: *ifInt*, *ifUserData*

The *ifInt* interface provides the following:

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

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.

roStreamLineEvent

Interfaces: *ifUserData*, *ifString*

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.

The *ifString* interface provides the following:

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

roSyncManager

This object provides advanced synchronization capabilities for video walls and other deployments that require closely calibrated interaction among players. *roSyncManager* handles all network traffic for master/slave synchronization, including the network clock. Multiple synchronization groups are allowed on the same local network and even within the same video wall.

Before using *roSyncManager*, you will need to instantiate a synchronization group by setting all players within the group to the same PTP domain value. To do this, use the *roRegistrySection.Write()* method to set the `ptp_domain` key of the "networking" section to a value between 0 and 127. In general, changes to the registry only take effect after a reboot, so the PTP synchronization service will start on each player after it is rebooted.

Example:

```
regSec = CreateObject("roRegistrySection", "networking")
regSec.Write("ptp_domain", "0")
regSec.Flush()

RebootSystem()
```

Object Creation: The *roSyncManager* object is created with an associative array representing a set of parameters.

```
CreateObject("roSyncManager", parameters as roAssociativeArray)
```

The associative array can have the following members:

- `Domain`: A string that is used to distinguish among different *roSyncManager* instances within the same synchronization group (i.e. PTP domain). The default string is "BrightSign". This parameter allows multiple *roSyncManager* instances to operate at the same time.

- `MulticastAddress`: A string specifying to which multicast address synchronization messages are communicated. The default address is "224.0.126.10".
- `MulticastPort`: A string specifying to which multicast port synchronization messages are communicated. The default port is "1539".

Interfaces: *ifMessagePort*, *ifSyncManager*

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`

The *ifSyncManager* provides the following:

- `SetMasterMode(master_mode As Boolean) As Boolean`: Specifies whether the unit is running the master instance of *roSyncManager*.
- `Synchronize(identifier As String, ms_delay As Integer) As Object`: Configures how the master unit will broadcast the time-stamped event to other players. It continues to send out this event every second to allow slave units that are powered on late to catch up. The network message contains the sync ID, as well as the domain and a timestamp. The timestamp is created at the point when this method is called; however, it can be offset by passing a non-zero `ms_delay`, allowing synchronization points to be set slightly in the future and giving the client enough time to switch video files and perform other actions. The event is returned from the call so that the caller can access the timestamp. The `identifier` parameter allows scripts to pass a filename, or some other useful marker, to the slave units as part of the synchronization message.

Note: *Because synchronization can involve slave units seeking to catch up with the playback of a master unit, we recommend using the more efficient MOV/MP4 container format when synchronizing video files. Transport Stream files (MPEG-TS) are also supported, but they must begin with a presentation timestamp (PTS) of 0. Program Stream files (MPEG-PS) are not supported.*

Currently, there are two objects that can accept synchronization parameters: The *roVideoPlayer* `PlayFile()` call accepts the parameters provided by *ifSyncManagerEvent* messages, while the *roImagePlayer* `DisplayFile()` and `PreloadFile()` calls accept `SyncIsoTimestamp` in an associative array. To synchronize image playback, an *roImagePlayer* object will simply delay the transition thread prior to running the transition. If there is a separate call for `DisplayFile()`, then the transition will be cancelled and the image will be displayed immediately (as with non-synchronized `DisplayFile()` calls).

Example

```
' Create a sync manager with default address and port.
aa1=CreateObject("roAssociativeArray")
aa1.Domain = "BS1"
s=CreateObject("roSyncManager", aa1)
p=CreateObject("roMessagePort")
s.SetPort(p)

' Create a video player - we're going to play a seamlessly looped file
v=CreateObject("roVideoPlayer")
v.SetLoopMode(True)

' THIS SECTION IS ONLY DONE BY THE MASTER
' We're the master unit - send out a synchronize event saying that we're starting.
' playback 1000ms from now
s.SetMasterMode(True)
msg = s.Synchronize("Blah1", 1000)

' THIS SECTION IS ONLY DONE BY THE SLAVE
' We're a slave unit, and we're sitting waiting for a sync message.
```

```
msg=Wait(4000, p)

' EVERYONE DOES THE REST
aa=CreateObject("roAssociativeArray")
aa.Filename = "Text_1.mov"
aa.SyncDomain = msg.GetDomain()
aa.SyncId = msg.GetId()
aa.SyncIsoTimestamp = msg.GetIsoTimestamp()

v.PlayFile(aa)
```

roSyncManagerEvent

These events are generated on slave units in response to *roSyncManager.Synchronize()* calls from the master unit. The *roSyncManager* on each slave unit will handle message duplicates, so the script will receive the sync message only once during normal operations.

If the slave unit is already booted up, then the event will arrive from the first network event generated by *roSyncManager.Synchronzie()*. On the other hand, if the slave unit is booted up while the master is in the middle of playing a video file or displaying an image file, then one of the message resends (generated at one second intervals by the master unit) will trigger the event. The script passes on the data from the event to the `PlayFile()` command of the video player or the `DisplayFile()` command of the image player, which will then determine how far forward in the file it needs to seek.

Interfaces: *ifUserData*, *ifSyncManagerEvent*

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.

The *ifSyncManagerEvent* interface provides the following:

- `GetDomain() As String`: Returns the domain of the sync group, which is specified during creation of the *roSyncManager* object on the master unit.
- `GetId() As String`: Returns the identifier of the event.
- `GetIsoTimestamp() As String`: Returns the timestamp of the event in ISO format.

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, ifSocketInfo, ifInternalGetTCPStream*

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.

The *ifSocketInfo* interface provides the following:

- `GetSourceAddress() As String`: Returns the IP address of the remote end of the TCP connection.

roTCPServer

Interfaces: *ifTCPServerInstance*, *ifUserData*

The *ifTCPServerInstance* interface provides the following:

- `GetFailureReason() As String`: Yields additional useful information if an *roTCPServer* method fails.
- `SetPort(port As Object)`: Sets the message port that will receive events from an *roTCPServer* instance.
- `BindToPort(port As Dynamic) As Boolean`: Prepares to accept incoming TCP connections on the specified port. Passing an integer to this method will specify a standard port number. This method can also accept an index of integer interfaces contained within an associative array, which can contain the following members:
  - `-1`: Any (this is the default value)
  - `0`: Ethernet
  - `1`: WiFi
  - `2`: Modem
  - `32767`: Loopback (i.e. TCP connections can only be established by internal sources)

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`: Returns the user data that has previously been set via `SetUserData()`. It will return Invalid if no data has been set.

roUPnPController

This object establishes and maintains a UPnP Control Point. It must exist for the entirety of UPnP discovery operations. Refer to the UPnP Device Architecture document for more information about UPnP discovery protocols.

Object Creation: The *roUPnPController* object is created without any parameters.

```
CreateObject("roUPnPController")
```

Interfaces: *ifUPnPController*, *ifMessagePort*, *ifUserData*

The *ifUPnPController* interface provides the following:

- `SetDebug(debug As Boolean) As Void`: Enables detailed debugging in the UPnP engine.
- `Search(searchTarget As String, mx As Integer) As Boolean`: Issues a Search request for a UPnP device. The parameters correspond to the ST (Search Target) and MX (Maximum wait time) header values that are sent with a UPnP M-SEARCH command. Responses to the Search request will generate messages in the form of *roUPnPSearchEvent* objects.

  Note: *The most common value for the* `searchTarget` *parameter is "upnp:rootdevice". This allows you to search all root devices, identify which devices you wish to interact with, and then get the* roUPnPDevice *and* UPnPService *instances for these embedded devices and services.*
- `RemoveDevice(udn As String) As Boolean`: Forces removal of the specified device from the Control Point device list. The passed string must include `"uuid:"` prepended to the UDN value.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`: Specifies the port that will receive events generated by the *roUPnPController* instance.

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.

UPnP Controller Operation

The *roUPnPController* object maintains a list of all currently detected UPnP devices accessible via the local network. To maintain this list, the *roUPnPController* object follows these generally accepted control-point practices:

- If an `ssdp:alive` multicast notification is received from a device that is not part of the list, it is queried for its device information and added to the list. However, the `ssdp:alive` message is not intended as the primary means for device discovery; rather, this behavior is intended to keep the list up-to-date and remove devices that disappear without an `ssdp:byebye` notification.
- If an `ssdp:byebye` multicast notification is received form a device that is part of the list, it will be removed from the list.
- The UPnP Controller allows a client to issue a Search request for UPnP devices. All devices on the network are expected to respond directly to the requesting device. If a response is received from a device that is not part of the list, it is queried for its device information and added to the list.
- UPnP devices report a "time-to-live" for notifications. For UPnP NOTIFY and search-response messages, this is contained in the "Cache-Control: max-age" header. Typically, this "time-to-live" is 20 or 30 minutes, though some devices have much shorter time values. Every device is configured to expire after its "time-to-live" is reached, at which point it is removed from the device list. The counter is reset (i.e. the device is renewed) after each receipt of an `ssdp:alive` message.

BrightScript does not allow direct access to its internal DeviceList. Rather, it raises events in the form of *roUPnPSearchEvent* objects when devices are added or removed from the list. These objects can, in turn, be used to retrieve *roUPnPDevice* objects containing all device information.

The controller also raises events whenever it receives a NOTIFY multicast message or a response to an M-SEARCH message (i.e. a response to a controller search request). These events return associative arrays containing headers from the NOTIFY multicast message or from the HTTP response to the M-SEARCH message.

The associative arrays may also contain additional non-header items. For an SSDP multicast message notification (type 0), the associative array will contain an "ssdpType" key, the value of which designates whether it is a NOTIFY or M-SEARCH message. In most cases, it is best to ignore M-SEARCH messages, unless you are implementing a UPnP device (the UPnP controller object does allow this).

During an M-SEARCH request, a "new device" notification (type 2) will only be sent when a device is added to the controller's internal list. Once a device is part of the device list, subsequent M-SEARCH requests will only return type 1 (search response) values for that device. This type 1 response returns an associative array with message headers, but not an *roUPnPDevice* object (which is used to contain a complete set of device information).

Note: *See the roUPnPSearchEvent entry for more information about the messages sent by the UPnP Controller.*

roUPnPSearchEvent

This event object is returned when there is a response to a *roUPnPController.Search()* operation. It is also returned when the *roUPnPController* object receives multicast UDP SSDP messages.

Interfaces: *ifUPnPSearchEvent*, *ifUserData*

The *ifUPnPSearchEvent* interface provides the following:

- `GetObject() As Object`: Returns either an *roUPnPDevice* or an *roAssociativeArray* instance, depending on the value returned from the `GetType()` method.
- `GetType() As Integer`: Returns an integer value indicating the type of response:
  - `0` (Advertisement) – Indicates the receipt of an SSDP multicast message, which can be either a NOTIFY message or an M-SEARCH message. The `GetObject()` method will return an associative array with all SSDP headers and an "ssdpType" key, which can have a value of either "m-search" or "notify".
  - `1` (Search response) – Indicates that the message is a response to an *roUPnPControllor.Search()* request. The `GetObject()` method will return an associative array with all headers from the HTTP M-SEARCH response.
  - `2` (New device added to the device list) – Indicates that the *roUPnPController* object has detected a new device and added it to the device list, which is maintained internally. Device detection can result from receiving either an `ssdp:alive` message or a response to an M-SEARCH message. The `GetObject()` method will return an *roUPnPDevice* instance containing information about the added device. This message type is only delivered once per new device. Once a device is part of the device list, subsequent M-SEARCH requests will only return type 1 (Search response) values for that device.
  - `3` (Device will be deleted from the device list) – Indicates that a device will be deleted from the device list, which is maintained internally. A device is deleted from the list when the player receives an `ssdp:byebye` message from the device, when the device does not send an `ssdp:alive` message within the defined "max-age" interval, or when the device is forcibly removed using the *roUPnPController.RemoveDevice()*

method. The `GetObject()` method will return an *roUPnPDevice* instance containing information about the device to be removed.

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.

roUPnPDevice

This object is returned by the *roUPnPSearchEvent.GetObject()* method under certain conditions.

Interfaces: *ifUPnPDevice*

The *ifUPnPDevice* interface provides the following:

- `GetUUID() As String`
- `GetHeaders() As roAssociativeArray`: Returns an associative array of headers (including vendor-specific extensions) associated with the advertisement or search.
- `GetDeviceInfo() As roAssociativeArray`: Returns an associative array of device metadata from the device XML (applicable to root items only).
- `GetEmbeddedDevices() As roAssociativeArray`: Returns an associative array of embedded *roUPnPDevice* object instances, keyed by device type.
- `GetEmbeddedDevice(deviceType As String) As roUPnPDevice`: Returns an *roUPnPDevice* instance, using the specified `deviceType` as a unique identifier. The `deviceType` parameter must use one of the following formats:
  - `urn:schemas-upnp:device:deviceType:v`: Search for any device of this type. The `deviceType` and version (`v`) are defined by the UPnP Forum working committee.
  - `urn:domain-name:device:deviceType:v`: Search for any device of this type. The `domain-name`, `deviceType`, and version (`v`) are defined by the UPnP vender. Period characters in the domain name must be replaced with hyphens in accordance with RFC2141.
- `GetServices() As roAssociativeArray`: Returns an associative array of embedded *roPnPService* object instances, keyed by type.
- `GetService(serviceType As String) As roUPnPService`: Returns an *roUPnPService* instance of the specified type.

roUPnPService

This object is returned by the `GetServces()` and `GetService()` methods on the *roUPnPDevice* object.

Interfaces: *ifUPnPService*, *ifMessagePort*, *ifUserData*

The *ifUPnPService* interface provides the following:

- `Invoke(actionName As String, params As Object) As Integer`: Invokes an action asynchronously on the UPnP service. This method returns a transaction ID that can be used to match it against the associated *roUPnPActionResult* instance.
- `Subscribe() As Integer`: Subscribes to events on the UPnP service asynchronously.
- `RenewSubscription() As Integer`: Resubscribes to events on the UPnP service asynchronously. This method should only be called after calling `Subscribe()`.
- `GetSID() As String`: Returns the subscription ID. This method should only be called after calling `Subscribe()`.
- `GetTimeout() As Integer`: Returns the service timeout period. This method should only be called after calling `Subscribe()`.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`: Specifies the port that will receive events generated by the *roUPnPService* instance.

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.

roUPnPServiceEvent

This event object is returned whenever a UPnP event message is received (for example, from a `Subscribe()` operation on the *roUPnPService* object). If a UPnP event message contains multiple state variables, separate event objects will be returned for each state variable.

Interfaces: *ifUPnPServiceEvent*, *ifUserData*

The *ifUPnPServiceEvent* interface provides the following:

- `GetUUID() As String`: Returns the subscription ID of the subscription service sending the event. This string matches the value returned by the `GetSID()` method of the *roUPnPService* instance that generated the event.
- `GetVariable() As String`: Returns the name of the UPnP state variable to which the value corresponds.
- `GetValue() As String`: Returns the value of the variable.
- `GetSequence() As Integer`: Returns the incrementing sequence number, which denotes the UPnP message from which the update originated. The sequence number will be the same for multiple variable updates reported in a single UPnP event.

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.

roUPnPActionResult

This object contains the results of an *roUPnPService.Invoke()* call. It is important to match the transaction ID of this object with the value returned by the `Invoke()` method.

Interfaces: *ifUPnPActionResult*, *ifUserData*

The *ifUPnPActionResult* interface returns the following:

- `GetType() As Integer`: Returns the result type, which can be one of the following:
  - `1` – Invoke result
  - `2` – Subscribe result
- `GetID() As Integer`: Returns the transaction ID of the result as an integer. Use it to match the *roUPnPActionResult* instance with the *roUPnPService.Invoke()* call that generated it.
- `GetResult() As Boolean`: Returns True if the originating `Invoke()` call was successful.
- `GetValues() As roAssociativeArray`: Returns an associative array containing the "out" values (if any) of the originating `Invoke()` call.

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.

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(matcher As Object) As Boolean`: Instructs the stream to use the specified matcher. This object returns True if successful. Pass Invalid to this method to stop using the specified matcher.

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:

- `SetSendEol(eol_sequence As String) As Void`: Sets the EOL sequence when writing to the stream. The default value is CR (ASCII value 13). If you need to set this value to a non-printing character, use the `chr()` global function.
- `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(a As Dynamic) As Void`: Writes the specified characters to the stream. This method can support either a string or an *roByteArray*. If the block is a string, any null bytes will terminate the block.
- `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.

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. It reports transfer status using the *roUrlEvent* object.

Object Creation: The *roUrlTransfer* object is created with no parameters.

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

Important: *You must create a separate* roUrlTransfer *instance for each asset you wish to read/write.*

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

The *ifUserData* interface provides the following:

- `SetUserData(user_data As Object)`: Associates an arbitrary object with the *roUrlTransfer* instance that is provided via `roUrlEvent.GetUserData()` when an event is generated.
- `GetUserData() As Object`: Retrieves the arbitrary object set using `SetUserData()`.

The *ifIdentity* interface provides the following:

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

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`: 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.
- `EnableResume(enable As Boolean) As Boolean`: Specifies the file-creation behavior of the `GetToFile()` and `ASyncGetToFile()` methods. If this method is set to False (the default setting), each download will generate a temporary file: if the download is successful, the temporary file will be renamed to the specified filename; if the download fails, the temporary file will be deleted. If this method is set to True, the file with the specified filename will be created regardless of whether the download is successful or not—this allows the download to be resumed by a subsequent `GetToFile()` or `ASyncGetToFile()` call.
- `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 ansynchronous HTTP HEAD request. Events will be sent to the message port associated with the object. If the request could not be issued, the method will return False and will not deliver any events.
- `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 fifteen minutes or more might be appropriate.
- `GetFailureReason As String`: May provide additional information if any of the *roUrlTransfer* methods indicate failure.
- `SetHeaders(a As Object) As Boolean`
- `AsyncGetToObject(type As String) As Boolean`: Begins an asynchronious GET request and uses the contents to create an object of the specified type. Events will be sent to the message port associated with the object. If this method returns False, the request could not be issued and no events will be delievered.
- `AsyncCancel() As Boolean`
- `EnableUnsafeAuthentication(enable 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.
- `EnableUnsafeProxyAuthentication(enable As Boolean) As Boolean`: Supports basic HTTP authentication against proxies if True (which, unlike `EnableUnsafeAuthentication()`, is the default setting). HTTP authentication uses an insecure protocol, which might allow others to easily determine the password. If this method is False, it will refuse to provide passwords via basic HTTP authentication, and any requests requiring this authentication type will fail.

- `EnablePeerVerification(a As Boolean) As Boolean`
- `EnableHostVerification(a As Boolean) As Boolean`
- `SetCertificatesFile(a As String) As Boolean`
- `EnableEncodings(enable As Boolean) As Boolean`: Enables HTTP compression, which communicates to the server that the system can accept any encoding that the *roUrlTransfer* object is capable of decoding by itself. This currently includes "deflate" and "gzip", which allow for transparent compression of responses. Clients of the *roUrlTransfer* instance see only the decoded data and are unaware of the encoding being used.

Note: *HTTP compression is enabled by default in firmware versions 6.0.x and later.*

- `SetUserAndPassword(a As String, b As String) As Boolean`
- `Head() As Object`: Performs a synchronous HTTP HEAD request and returns 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.
- `Escape(unescaped As String) As String`: Converts the provided string to a URL-encoded string. All characters that could be misinterpreted in a URL context are converted to the %XX form.
- `Unescape(a As String) As String`
- `GetUrl() As String`
- `SetProxy(proxy As String) As Boolean`: Sets the name or address of the proxy server that will be used by the *roUrlTransfer* instance. The proxy string should be formatted as "http://user:password@hostname:port". It can contain up to four "*" characters; each "*" character can be used to replace 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 player will attempt to use a proxy named "proxy-192.168".
- `SetProxyBypass(hostnames As Array) As Boolean`: Exempts the specified hosts from the proxy setting. The passed array should consist of one or more hostnames. The player will attempt to reach the specified hosts directly rather than using the proxy that has been specified with the `SetProxy()` method. For example, the hostname "example.com" would exempt "example.com", "example.com:80", and "www.example.com" from the proxy setting.

- `SetTimeout(milliseconds As Integer) As Boolean`: Terminates the transfer if the request takes longer than the specified number milliseconds. Note that this includes the time taken by any name lookups, so setting this value too low will cause undesirable results. Passing 0 to the method disables the timeout. This method returns True upon success and False upon failure. In the event of failure, using the `GetFailureReason()` method may provide more information. If the operation times out, the status return is -28.
- `SetUserAgent(a As String) As Boolean`
- `PutFromString(a As String) As Integer`: Uses the HTTP PUT method to write the supplied string to the current URL and return the response code. Any response body is discarded; use *roUrlTransfer.SyncMethod* to retrieve the response body.
- `PutFromFile(a As String) As Integer`: Uses the HTTP PUT method to write the contents of the specified file to the current URL and return the response code. Any response body is discarded; use *roUrlTransfer.SyncMethod* to retrieve the response body.
- `AsyncPutFromString(a As String) As Boolean`: Uses the HTTP PUT method to write 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. Any response body is discarded; use *roUrlTransfer.AsyncMethod* to retrieve the response body.
- `AsyncPutFromFile(a As String) As Boolean` : Uses the HTTP PUT method to write the contents of the specified file to the current URL. Events of type *roUrlEvent* will be sent to the message port associated with the object. A False return indicates that the request could not be issued and no events will be delivered. Any response body is discarded; use *roUrlTransfer.AsyncMethod* to retrieve the response body.
- `Delete() As Object`: Uses the HTTP DELETE method to delete the resource at the current URL and return the response code. Any response body is discarded; use *roUrlTransfer.SyncMethod* to retrieve the response body.
- `AsyncDelete() As Boolean`: Uses the HTTP DELETE method to delete the resource at 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. Any response body is discarded; use *roUrlTransfer.AsyncMethod* to retrieve the response body.
- `ClearHeaders()`: Removes all headers that would be supplied with an HTTP request.

- `AddHeaders(a As Object) As Boolean`: Adds one or more headers to HTTP requests. Pass headers to this object as an *roAssociativeArray* of name/value pairs. This method returns True upon success and False upon failure. All headers that are added with this method will continue to be sent with HTTP requests until `ClearHeaders()` is called.
- `SyncMethod(a As Object) As Object`: Performs a synchronous HTTP method request using the specified parameters. If the request is started successfully, then the method returns an roUrlEvent object containing the results of the request. This method returns Invalid if the the request could not be started. In this case, the `GetFailureReason()` method may provide more information.
- `SetRelativeLinkPrefix(prefix As String) As Boolean`: Places the specified prefix in front of the URL if the URL is relative. Use this method to easily make file:/// URLs drive agnostic.
- `BindToInterface(interface As Integer) As Boolean`: Ensures that the request only goes out over the specified network interface. By default, the request goes out over the most appropriate network interface (which may depend on the routing metric configured via *roNetworkConfiguration*). Note that if both interfaces are on the same layer 2 network, this method may not always work as expected due to the Linux weak host model. The default behavior can be selected by passing -1 to the method. This method returns False upon failure. In this case, the `GetFailureReason()` method may provide more information.
- `AsyncMethod(parameters As roAssociativeArray) As Boolean`: Begins an asynchronous HTTP method request using the specified parameters (see below). If the request is started successfully, the method returns True and and will deliver an event. If the request could not be started, then the method returns False and will not deliver an event. If this occurs, you may be able to use the `GetFailureReason()` method to get more information.

  The parameters are sepecifed using an *roAssociativeArray* instance that may contain the following members:

Name	Type	Description
`method`	String	An HTTP method. Normal values include "HEAD", "GET", "POST", "PUT", and "DELETE". Other values are supported; however, depending on server behavior, they

		may not work as expected.
request_body_string	String	A string containing the request body.
request_body_file	String	The name of a file that contains the request body
response_body_string	Boolean	If specified and set to True, the response will be stored in a string and provided via the roUrlEvent.GetString() method.
response_body_file	String	The name of the file that will contain the response body. The body is written to a temporary file and then renamed to the specified filename if successful.
response_body_resume_file	String	The name of the file that will contain the response body. For a GET request, a RANGE header is sent based on the current size of the file, which is written in place rather than using a temporary file.
response_body_object	String	Uses the response body to create an object of the specified type. See the entry for AsyncGetToObject() for supported object types.
response_pipe	*roArray*	Use a pipeline of handlers to process the response body as it is received. See below for more details.

The *roArray* response for response_pipe consists of one or more *roAssociativeArray* instances containing a filter description (see below). The last associative array is usually an output filter.

Name	Type	Description
hash	String	Calculate a hash (digest) of the data using the specified algorithm as it passes through the pipeline. Supported hashes include the following: "CRC32", "MD5", "SHA1", "SHA256", "SHA384", "SHA512". The resulting hash can be retrieved as a hexadecimal string using the *roUrlEvent.GetHash()* method.
decompress	String	Decompress the response body using the specified algorithm. Currently, the only supported algorithm is "gzip". It is often easier to use an HTTP Content-Encoding rather than explicitly decompressing the body.
prefix_capture	Integer	Capture the specified number of bytes (between 1 and 16384) from the start of the

		stream and store them separately. The bytes can be retrieved using the *roUrlEvent.GetPrefix()* method, but they cannot be passed on to subsequent filters.
`output file`	String	Output the pipeline to the specified file. The output is written to a temporary file and then renamed to the specified filename if successful.
`output_string`	Boolean	If specified and set to True, the response will be stored in a string and provided via the `roUrlEvent.GetString()` method.

Example: The following code specifies an array of handlers to filter the response body of an HTTP request.

```
url = CreateObject("roUrlTransfer")
pipe = [ { decompress: "gzip"}, { hash: "MD5" }, { output_file: "test.txt" } ]
result = url.AsyncMethod({ method: "GET", response_pipe: pipe })
```

roUrlEvent

This event is generated by the *roUrlTransfer* object.

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).

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`: Retrieves an arbitrary object set via the `SetUserData()` method of the *roUrlTransfer* instance that generated this event.

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.

The following error codes are generated by the system software, and are outside the range of CURL events:

Status	Name	Description
-10001	ERROR_CANCELLED	The transfer request has been cancelled because the *roUrlTransfer* instance is out of scope.
-10002	ERROR_EXCEPTION	The callback threw an exception.

- `GetObject() As Object`: Returns the object associated with the event. Currently, this method can only return an object created in response to an *roUrlTransfer.AsyncGetToObject* request.
- `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 the event came from.
- `GetResponseHeaders() As roAssociativeArray`: Returns an associative array containing all the headers returned by the server for appropriate protocols (such as HTTP).
- `GetHash() As String`: The hash (digest) of the response body, as specified by the `response_pipe{hash:}` parameter of the *roUrlTransfer.AsyncMethod()* method.
- `GetPrefix() As String`: A number of bytes from the start of the response body. The amount of bytes is specified with the `response_pipe{prefix_capture:}` parameter of the *roUrlTransfer.AsyncMethod()* method.

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.

INPUT/OUTPUT OBJECTS

roCecInterface

This object provides access to the HDMI CEC channel.

Object Creation: The *roCecInterface* object is created with no parameters.

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

Interfaces: *IfCecInterface*, *ifUserData*, *ifMessagePort*

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 because they will be replaced with the source address (unless source bit replacement is disabled using the `UseInitiatorAddressFromPacket()` method).
- `UseInitiatorAddressFromPacket(enable As Boolean) As Boolean`: Removes the source address included by system software if passed True. This method allows you to transmit unmodified bytes via CEC.
- `GetLogicalAddress() As Integer`
- `EnableCecDebug(a As String)`

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.

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

roCecRxFrameEvent, roCecTxCompleteEvent

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

roCecRxFrameEvent

Interfaces: *ifCecRxFrameEvent*

The *ifCecRxFrameEvent* interface provides the following;

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

roCecTxCompleteEvent

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: The *roChannelManager* object is created with no parameters.

```
CreateObject("roChannelManager")
```

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

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.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`

The *ifMessagePort* 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 I/O port of the BP200 and BP900 USB button boards, as well as the on-board I/O port and side buttons on the BrightSign player. It also supports "button-up" events. The object is used to configure output levels on the I/O connector and monitor inputs. Typically, LEDs and buttons are attached to the I/O connector on the BrightSign player or the BrightSign Expansion Module.

Object Creation: The *roControlPort* object is created with a single parameter that specifies the port being used.

```
CreateObject("roControlPort", port As String)
```

The `port` parameter can be one of the following:

- `BrightSign`: Specifies the onboard DA-15 connector, as well the SVC (GPIO12) and Reset buttons.
- `Expander-GPIO`: Specifies 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`: Specifies 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.*

- `Touchboard-<n>-GPIO`: Retrieves events from the specified BP200/BP900 button board. Events are handled in the same manner as events from the `BrightSign` port.
- `Touchboard-<n>-LED-SETUP`: Sets various LED output options for the specified BP200/BP900 button board.
- `Touchboard-<n>-LED`: Sets the bits for each button on the specified BP200/BP900 button board. The bits indicate whether the associated LED should be on or off.

Note: *Since multiple BP200/BP900 button boards can be connected to a player simultaneously, the `<n>` value specifies the port enumeration of each board. An unspecified enumeration value is synonymous with a button board with an enumeration value of `0` (e.g. `Touchboard-GPIO` and `Touchboard-0-GPIO` are identical).*

Interfaces: *ifControlPort*, *ifMessagePort*, *ifUserData*, *ifIdentity*

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) As Boolean`: 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.
- `GetIdentity() As Integer`: Returns the identity value that can be used to associate *roControlUp* and *roControlDown* events with this control port.
- `SetOutputState(pin As Integer, level As Boolean) 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. This method can also be used to configure LED output behavior on BP200/B900 button boards; see the **BP200/BP900 Setup** section below for more details.

- `SetOutputValue(offset As Integer, bit-mask As Integer)`: Configures the BP200/BP900 button board when *roControlPort* object is instantiated with the `Touchboard-<n>-LED-SETUP` or `Touchboard-<n>-LED` parameter. See the **BP200/900 Setup** section below for more details.
- `GetProperties() As roAssociativeArray`: Returns an associative array of values related to the attached BP200/BP900 button board, including `hardware`, `header`, and `revision`. This method can only be used with an *roControlPort* instantiated with the `Touchboard-<n>-GPIO` parameter.
- `SetPulseParams(parameters As roAssociativeArray) As Boolean`: Specifies a period of time, as well as the time slices within that period, for pulsing GPIO LED pins. These properties are applied to all GPIO pins. This method is passed an associative array with the following parameters:
    - `milliseconds`: An integer specifying the time period (in ms) for pulsing
    - `slices`: An integer specifying the number of divisions within the `milliseconds` time period: For example, a 500ms time period with `slices:2` is divided into two 250ms slices.
- `SetPulse(pin As Integer, bit-field As Integer) As Boolean`: Sets the off/on bit field for a particular GPIO pin. Use the `slices` parameter of the `SetPulseParams()` method to determine the number of bits in the bit field. For example, specifying `milliseconds:500`, `slices:2`, and a bit field of `10` will cause the pin to turn on every other 250 millisecond period.
- `RemovePulse(pin As Integer) As Boolean`: Removes the specified GPIO pin from the set of pins affected by the pulse.

The *ifMessagePort* interface provides the following:

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

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.

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

Example: The following code applies timed pulses to a set of GPIO pins.

```
' set up pin 2 and 3 to flash at 2Hz (i.e. on & off twice in a second) in an alternating '
fashion.

gpioPort = CreateObject("roControlPort", "BrightSign")

gpioPort.EnableOutput(2)
gpioPort.SetOutputState(2, true)

gpioPort.EnableOutput(3)
gpioPort.SetOutputState(3, true)

' set up pulse to have two time slices of 250ms each.
gpioPort.SetPulseParams({ milliseconds: 500, slices: 2 })

' pin 2 will have slice 1 on and slice 2 off.
gpioPort.SetPulse(2, &h01)

' pin 3 will have the reverse of pin 2.
gpioPort.SetPulse(3, &h02)


' wait for a bit.
```

```
sleep(10000)

' stop pulsing on pin 2.
gpioPort.RemovePulse(2)
```

BP200/BP900 Setup

To send a configuration to the BP200/BP900 button board, instantiate *roControlPort* with the `Touchboard-<n>-LED-SETUP` parameter and call the `SetOuputValue()` method. This method accepts two integers: the first integer specifies one of three command types (offsets); the second integer is a bit field consisting of 32 bits.

- **Offset 0**: Configures the button board using a bit field that is split into four bytes of eight bits each. Each byte is a separate part of the configuration. In the script, these bytes need to be listed from right to left in hex value (i.e. Byte 1 + Byte 2 + Byte 3 + Byte 4).
    - Byte 1: Specifies the configuration type for the button board. Currently, the only configuration type is for LED output, which is specified with the value `&hA0`.
    - Byte 2: The button number(s) that will be configured. Buttons are numbered beginning from 1. The value is set to 0 (`&h00`) if this command is not required.
    - Byte 3: The LED bit-field configuration. This value specifies how many on/off bits should be used (up to 32 bits) when `SetOutputValue()` is called on a `Touchboard-<n>-LED` instance (see the **BP200/BP900 LED Output** section below for details). Set the value to 0 (`&h00`) if this command is not required (the bit field will be set to eight bits by default).
    - Byte 4: This value is currently always set to 0 (`&h00`).
- **Offset 1**: Disables buttons on the button board according to values in the bit field. Each button is disabled individually by setting bits 0-10: For example, passing the hex value `&h00000008` will disable button 4 only.
- **Offset 2**: Disables LEDs on the button board according to values in the bit field. Each LED is disabled individually by setting bits 0-10: For example, passing the hex value `&h00000080` will disable the LED on button 8 only.

Note: *Disabling a button LED will not automatically disable the button itself (and vice-versa). To disable both the button and the LED, make separate* `SetOutputValue()` *calls for Offset 1 and Offset 2.*

BP200/BP900 LED Output

To control the behavior of individual button LEDs, instantiate *roControlPort* with the `Touchboard-<n>-LED` parameter, then pass per-LED bit fields to the `SetOutputValue()` method. This method accepts two integers: the first integer specifies the button number (0-11), while the second integer uses a bit field to specify the on/off behavior of the button LED. The size of the bit field (up to 32 bits) is determined with the Offset 0 – Byte 3 value described in the section above.

Each bit specifies the on/off behavior of a single cycle, and the BP200/BP900 button boards run at approximately 11Hz. For example, if you want an LED to cycle on every other second, you would set the Offset 0 – Byte 3 value to `&h16` (22 bits) and the bit field itself to `&h3FF800` (`0000000000011111111111`).

Example: The following code sets a BP900 to “twinkle” by turning off each button LED at a different point in the cycle.

```
led=CreateObject("roControlPort", "TouchBoard-0-LED")
led_setup=CreateObject("roControlPort", "TouchBoard-0-LED-SETUP")
led_setup.SetOutputValue(0, &h000B00A0)
led.SetOutputValue(0, &h07fe)
led.SetOutputValue(1, &h07fd)
led.SetOutputValue(2, &h07fb)
led.SetOutputValue(3, &h07f7)
led.SetOutputValue(4, &h07ef)
led.SetOutputValue(5, &h07df)
led.SetOutputValue(6, &h07bf)
led.SetOutputValue(7, &h077f)
led.SetOutputValue(8, &h06ff)
led.SetOutputValue(9, &h05ff)
```

```
led.SetOutputValue(10, &h03ff)
```

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.

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

Interfaces: *ifMessagePort*, *ifGpioControlPort*

The *ifMessagePort* 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`
- `EnableOutput(output_id As Integer) As Boolean`

roGpioButton

Interfaces: *ifInt*

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

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

roIRReceiver

This object supports receiving arbitrary Infrared remote control codes using the NEC and RC5 protocols.

Object Creation: The *roIRReceiver* object is created with an associative array specifying the following:

- `source`: A string value indicating the source of the input.
  - `"IR-in"`: The 3.5mm IR input/output connector (available on 4Kx42 and XDx32 models)
  - `"GPIO"`: Pin 1 of the GPIO connector
  - `"Iguana"`: The Iguanaworks IR transceiver. This source can support both NEC and RC5 encodings simultaneously.
- `encodings`: An array indicating the required encodings.
  - `"NEC"`
  - `"RC5"` (supported on the Iguanaworks IR transceiver only)

```
CreateObject("roIRReceiver", config As roAssociativeArray)
```

NEC codes are expressed in 24 bits:

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

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

The *roIRReceiver* object can generate the following events:

- *roIRDownEvent*: Generates when a button is pressed.
- *roIRRepeatEvent*: Generates when a button repeats.
- *roIRUpEvent* (Iguanaworks IR transceiver only): Generates when a button is released.

Interfaces: *ifUserData*, *ifMessagePort*

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.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`: Specifies the port that will receive events generated by the *roIRReceiver* instance.

roIRDownEvent, roIRRepeatEvent, roIRUpEvent

An IR event object is generated when an IR button input (button press, button repeat, button release) is received by the *roIRReceiver* object. Use these objects to retrieve the message body of the IR input.

Note: *The* roIRUpEvent *object is generated with the Iguanaworks IR transceiver only.*

These event objects provide the following:

- `GetCode() As Integer`: Returns the IR code received by the *roIRReceiver* instance.
- `SetCode(a As Integer)`: Overrides the IR code received by the *roIRReceiver* instance, replacing it with the specified binary code.

Interfaces: *ifUserData, ifReceivedEvent*

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.

The *ifReceivedEvent* interface provides the following:

- `GetEncoding() As String`: Returns the `encodings` setting of the *roIRReceiver* instance. This setting can be one of the following strings:
  - `"NEC"`
  - `"RC5"` (supported on the Iguanaworks IR transceiver only)

roIRTransmitter

This object supports sending arbitrary remote control Infrared remote control codes using the NEC, RC5, or PHC (Pronto Hex Controls) protocols.

Object Creation: The *roIRTransmitter* is created with an associate array specifying the following:

- `destination`: A string value indicating the connector that will be used to output the signal.
  - `"IR-out"`: The 3.5mm IR output connector (available on XDx30 models) or 3.5mm IR input/output connector (available on 4Kx42 and XDx32 models)
  - `"Iguana"`: The Iguanaworks IR transceiver

Note: *System software will not prevent you from generating both an* roIRTransmitter *instance set to* `IR-out` *and an* roIRReceiver *instance set to* `IR-in` *(i.e. configuring the 3.5mm IR connector for input and output at the same time). However, input/output performance will not be reliable.*

```
CreateObject("roIRTransmitter", config as roAssociativeArray)
```

Interfaces: *ifMessagePort*, *ifUserData*, *ifIRTransmitter*

The *ifMessagePort* interface provides the following:

- `SetPort(message_port_object As Object) As Void`

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.

The *ifIRTransmitter* interface provides the following:

- `GetFailureReason() As String`
- `Send(protocol As String, code As Dynamic) As Boolean`: Sends the specified code using the output `destination` set during object creation. The system currently supports two IR transmission protocols: "NEC" and "PHC" (Pronto Hex Code). This method returns True if the code was successfully transmitted, but there is no way to determine from BrightScript if the controlled device actually received it.
- `AsyncSend(protocol As String, code As Dynamic) As Boolean`: Sends the specified code and generates an *roIRTransmitCompleteEvent* object upon completion. The system currently supports two IR transmission protocols: "NEC" and "PHC" (Pronto Hex Code). This method is only supported using the `IR-out` destination.

roIRTransmitCompleteEvent

This event object is generated by the *roIRTransmitter.ASyncSend()* method. It does not return any information other than user data.

Interfaces: *ifUserData*

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.

roIRRemote

This object supports receiving and transmitting arbitrary Infrared remote control codes using the NEC protocol. You can also use this object to send PHC (Pronto Hex Code) commands. The best way to determine the required send values is to capture the codes received by *roIRRemote* when the remote buttons of the device are pressed and then send the same codes.

Important: *The* roIRRemote *object cannot be used to receive input over the 3.5mm IR port on the 4Kx42 and XDx32 series—use the* roIRReceiver *object instead.*

NEC codes are expressed in 24 bits:

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

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

Interfaces: *ifMessagePort*, *ifIRRemote*.

The *ifMessagePort* interface provides the following:

- `SetPort(message_port_object As Object) As Void`

The *ifIRRemote* interface provides the following:

- `Send(protocol as String, code as Dynamic) As Boolean`: Sends the specified code using the IR blaster. The system currently supports two IR transmission protocols: "NEC" and "PHC" (Pronto Hex Code). This method returns True if the code was successfully transmitted, but there is no way to determine from BrightScript if the controlled device actually received it.

Pronto Hex Format

Raw captures of Pronto Hex commands will likely not work with the inbuilt IR blaster, though they should work with Iguanaworks IR transceivers. This is a result of the trailing *off* periods, which are too long to be ecoded properly. Changing the *off* periods to all zeros ("0000") will fix this issue.

Example: The following example sends an "ON" command to a Panasonic television using a single string of Pronto Hex Code. You can also provide Pronto Hex Code as an *roArray* of hex values, which results in less work for the script engine.

```
ir = CreateObject("roIRRemote")

    pronto_hex_Panasonic_on_str = " 0000 0071 0000 0032 0080 003F 0010 0010 0010 0030 0010 0010
0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010
0010 0010 0030 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010
0010 0010 0010 0030 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010
0010 0010 0010 0010 0030 0010 0030 0010 0030 0010 0030 0010 0030 0010 0010 0010 0010 0010 0010
0010 0030 0010 0030 0010 0030 0010 0030 0010 0030 0010 0010 0010 0030 0010 0000"

    ir.Send("PHC", pronto_hex_lg_on_str)
```

roIRRemotePress

Messages of the type *roIRRemotePress* are generated upon key presses from a Roku Soundbridge remote.

Interfaces: *ifInt*

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

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

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. It can also be used to configure the localization of the keyboard.

Interfaces: *ifMessagePort*, *ifUserData*, *ifKeyboardConfig*

The *ifMessagePort* interface provides the following:

- `SetPort(port As Object) As Void`

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.

The *ifKeyboardConfig* interface provides the following:

- `SetLayout(layout As String) As Boolean`: Specifies the localized layout for the attached USB keyboard. This setting takes effect immediately and persists in the registry after a reboot. The following table lists valid keymap parameters (players are set to `us` by default):

`af`	Afghanistan	`es`	Spain	`kh`	Cambodia	`pk`	Pakistan
`al`	Albania	`et`	Ethiopia	`kr`	Korea, Republic of	`pl`	Poland
`am`	Armenia	`fi`	Finland	`kz`	Kazakhstan	`pt`	Portugal
`at`	Austria	`fo`	Faroe Islands	`la`	Laos	`ro`	Romania
`az`	Azerbaijan	`fr`	France	`lk`	Sri Lanka	`rs`	Serbia

ba	Bosnia and Herzegovina	gb	United Kingdom	lt	Lithuania	ru	Russia
bd	Bangladesh	ge	Georgia	lv	Latvia	se	Sweden
be	Belgium	gh	Ghana	ma	Morocco	si	Slovenia
bg	Bulgaria	gn	Guinea	md	Moldova	sk	Slovakia
br	Brazil	gr	Greece	me	Montenegro	sn	Senegal
bt	Bhutan	hr	Croatia	mk	Macedonia	sy	Syria
bw	Botswana	hu	Hungary	ml	Mali	th	Thailand
by	Belarus	ie	Ireland	mm	Myanmar	tj	Tajikistan
ca	Canada	il	Israel	mn	Mongolia	tm	Turkmenistan
cd	Congo (DRC)	in	India	mt	Macao	tr	Turkey
ch	Switzerland	iq	Iraq	mv	Maldives	tw	Taiwan
cm	Cameroon	ir	Iran	ng	Nigeria	tz	Tanzania
cn	China	is	Iceland	nl	Netherlands	ua	Ukraine
cz	Czech Republic	it	Italy	no	Norway	**us**	United States
de	Germany	jp	Japan	np	Nepal	uz	Uzbekistan
dk	Denmark	ke	Kenya	pc	Pitcairn	vn	Vietnam
ee	Estonia	kg	Kyrgyzstan	ph	Philippines	za	South Africa

roKeyboardPress

This is an event object 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 *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 Reference Guide 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`
- `SetWatchdogTimeout(seconds As Integer) As Integer`: Enables a watchdog timeout on the *roMessagePort* instance. The watchdog on *roMessagePort* is disabled by default. Passing a positive integer to this method instructs the watchdog to crash and reboot the player if `GetMessage()` or `WaitMessage()` does not return after the specified number of seconds. Passing zero to this method disables the watchdog again.

Note: *The watchdog timeout will not trigger while waiting on the BrightScript debugger prompt.*

- `DeferWatchdog(a As Integer)`: Defers the watchdog timeout set by the `SetWatchdogTimeout()` method. Passing an integer to this method defers the timeout for the specified number of seconds.
- `DeferWatchdog()`: Defers the watchdog timeout by the amount of seconds set in the `SetWatchdogTimeout()` method.

Note: *Calls to either `DeferWatchdog()` method cannot cause the watchdog to trigger earlier than it already will. For example, calling `DeferWatchdog(100)` followed by `DeferWatchdog(10)` will still cause the watchdog to trigger after 100 seconds.*

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

This object is used to send *roSequenceMatchEvent* events when the specified byte sequence patterns are matched. Once a pattern has been matched and the event has been posted, all contributing bytes are discarded. As a result, if one pattern is a prefix of another pattern, then the second, longer pattern will never be matched by the object.

Interfaces: *ifStreamReceiveObserver, ifSequenceMatcher*

The *ifSequenceMatcher* interface provides the following:

- `SetPort(a As Object)`: Specifies the message port where *roSequenceMatchEvent* objects will be posted.
- `Add(pattern As Object, user_data As Object) As Boolean`: Adds a pattern to be matched by the *roSequenceMatcher* object instance. The pattern should be specified as an object that is convertible to a byte sequence (e.g. *roByteArray*, *roString*). For the user data, pass the object that should be returned if the specified pattern is matched.

Example

```
Function FromHex(hex as String) as Object
    bytes = CreateObject("roByteArray")
    bytes.FromHexString(hex)
    return bytes
End Function

Sub Main()
    serial = CreateObject("roSerialPort", 1, 115200)
    mp = CreateObject("roMessagePort")

    button1_seq = FromHex("080a01040001e000")
```

```
    button2_seq = FromHex("080e01040001e000")

    matcher = CreateObject("roSequenceMatcher")
    matcher.SetMessagePort(mp)
    matcher.Add(button1_seq, { name: "button1" })
    matcher.Add(button2_seq, { name: "button2" })
    matcher.Add("flibbet", { name: "flibbet" })
    matcher.Add("flobbet", { name: "flobbet" })

    if not serial.SetMatcher(matcher) then
       stop
    end if

    finished = false
    while not finished
       ev = mp.WaitMessage(10000)
       if ev = invalid then
           finished = true
       else if type(ev) = "roSequenceMatchEvent" then
           print "Got button: "; ev.GetUserData().name
       else
           print "Unexpected event: "; type(ev)
       end if
    end while
End Sub
```

roSequenceMatchEvent

This object is generated whenever *roSequenceMatcher* matches a specified byte sequence pattern.

Interfaces: *ifUserData*

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.

roSerialPort

This object controls the RS-232 serial port, allowing you to receive input and send responses.

Object Creation: The *roSerialPort* object is created with two parameters.

```
CreateObject(roSerialPort, port As Integer, baud_rate As Integer)
```

- `port`: The port enumeration of the serial device. Most standard RS-232 serial devices enumerate on port 0. If you are connecting a USB-serial device (such as a GPS receiver), it will enumerate on port 2.
- `baud_rate`: The baud rate for serial communication. The serial port supports the following baud rates: 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200.

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:

- `SetSendEol(eol_sequence As String) As Void`: Sets the EOL sequence when writing to the stream. The default value is CR (ASCII value 13). If you need to set this value to a non-printing character, use the `chr()` global function.
- `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(a As Dynamic) As Void`: Writes the specified characters to the stream. This method can support either a string or an *roByteArray*. If the block is a string, any null bytes will terminate the block.
- `Flush()`

The *ifStreamReceive* interface provides the following:

- `SetLineEventPort(port As Object) As Void`
- `SetByteEventPort(port As Object) As Void`
- `SetReceiveEol(eol_sequence As String)`: Sets the EOL sequence to detect when receiving the stream. The default value is CR (ASCII value 13). If you need to set this value to a non-printing character, use the `chr()` global function.
- `SetMatcher(matcher As Object) As Boolean`: Instructs the stream to use the specified matcher. This method returns True if successful. Pass Invalid to this method to stop using the specified matcher.

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: 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400.
- `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)`
- `SetFlowControl(enalbe As Boolean) As Boolean`: Enables or disable RTS/CTS handshaking over the serial port. This feature is currently only available on 4Kx42, XDx32, and HDx22 models.

- `SetInverted(inverted As Boolean) As Boolean`: Inverts the signals on the player serial port. This allows the player to communicate with most PCs that use -12v to 12v signaling. Passing True to the method enables inversion, whereas passing False disables it.
- `SendBreak(duration_in_ms As Integer) as Boolean`: Sends a serial break or sets the serial break condition. This method returns True upon success and False upon failure.
    a. `duration_in_ms = -1`: Sends a continuous break.
    b. `duration_in_ms = 0`: Clears the break state.
    c. `duration_in_ms >= 100`: Sets the break condition for the specified period of milliseconds (note that this integer is only accurate to the tenth of a second).

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

roDeviceCustomization

This object provides miscellaneous device configuration and customization methods.

Interfaces: *ifFailureReason, ifDeviceCustomization*

The *ifFailureReason* interface provides the following:

- `GetFailureReason() As String`: Returns helpful information if one of the *ifDeviceCustomization* methods fail.

The *ifDeviceCustomization* interface provides the following:

- `WriteSplashScreen(filename As String) As Boolean`: Removes the default splash screen (or a previously set splash screen) and replaces it with the specified image file. The image file must use a supported format. This method returns True upon success and False upon failure.
- `FactoryReset(confirm As String) As Boolean`: Applies a factory reset to the player. This method must be passed the string "confirm" to work; otherwise, it will return False and do nothing. If successful, this method will reboot without a return value. The following steps will be carried out during a factory reset:
  1. All files are wiped from the `BOOT:` drive (including custom splash screens and autorun scripts).
  2. All values are wiped from the registry.
  3. The RTC is reset (if the player has an RTC).
  4. The `FLASH:` drive is wiped.

roDeviceInfo

This object provides information about the device hardware, firmware, and features.

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, "HD1020" or "XD230").
- `GetVersion() As String`: Returns the version number of 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 comparable numeric form: `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 comparable numeric form: `major*65536 + minor*256 + build`
- `FirmwareIsAtLeast(version As String) As Boolean`: Returns True if the BrightSign firmware version on the device is less than or equal to the version number represented by the passed string (e.g. "6.0.1").
- `BootFirmwareIsAtLeast(version As String) As Boolean`: Returns True if the BrightSign boot firmware version on the device is less than or equal to the version number represented by the passed string (e.g. "4.4.22").
- `GetDeviceUptime() As Integer`: Returns the number of seconds that the device has been running since the last power cycle or reboot.
- `GetLoadStatistics(parameters As roAssociativeArray) As String`: Provides current performance information related to the Linux kernel. This method accepts an associative array with a single key/value pair formatted as `{item: <parameter>}`; it will then return a string containing information associated with that parameter. The following are recognized parameters:

    - `"loadavg"`: Provides information about system performance. The first three columns measure CPU and I/O utilization over the past 1, 5, and 10 minutes, respectively. The fourth column displays the number of currently running processes and the total number of processes. The last column displays the ID of the most recently used process.
    - `"meminfo"`: Displays physical and swap memory usage.
    - `"slabinfo"`: Provides information about memory usage at the slab level.
    - `"stat"`: Provides overall statistics about the system (e.g. the number of page faults since the system booted).
    - `"vmstat"`: Displays detailed virtual memory statistics from the kernel.
    - `"zoneinfo"`: Provides overall statistics about the system, broken down by system Node.
    - `"interrupts"`: Displays which interrupts are in use and how many of each type there have been.
    - `"version"`: Provides the kernel version.
- `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(feature As String) As Boolean`: Returns True if the player feature, which is passed as a case-insensitive string parameter, is present on the current device and firmware. The possible features that can be queried from the script are listed below:

  `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": A second audio output
- "audio3": A third audio output
- "ethernet"
- "usb"
- "serial port 0": The first RS-232 serial port
- "serial port 1": A second RS-232 serial port
- "serial port 2": A third RS-232 serial port
- "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: The *roResourceManager* object is created with a single `filename` parameter that specifies 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.

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

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 values 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:

- `SetSendEol(eol_sequence As String) As Void`: Sets the EOL sequence when writing to the stream. The default value is CR+LF. If you need to set this value to a non-printing character, use the `chr()` global function.
- `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(a As Dynamic) As Void`: Writes the specified characters to the stream. This method can support either a string or an *roByteArray*. If the block is a string, any null bytes will terminate the block.
- `Flush()`

The *ifSystemLog* interface provides the following:

- `ReadLog() As Object`

DATE AND TIME OBJECTS

roDateTime

This object is used to represent an instant in time.

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`: Returns the current *roDateTime* value as an ISO-8601 basic formatted string. Hyphens for date and colons for time are omitted, and a comma is used to separate seconds from milliseconds: For example, the ISO-8601 standard "2014-05-29T12:30:00.100" would be formatted as "20140529T123000,100".
- `FromIsoString(date-time As String) As Boolean`: Sets the value of the *roDateTime* object using an ISO-8601 basic formatted string. Hyphens for date and colons for time are omitted, and either a period or comma can be used to separate seconds from milliseconds: The ISO-8601 standard "2014-05-29T12:30:00.100" could, for example, be formatted as either "20140529T123000,100" or "20140529T123000.100". This method will return False (indicating that it has not affected changes to the *roDateTime* object) if the string is formatted incorrectly or if the date passed is outside the range of January 1, 1970 and December 31, 2100.
- `ToSecondsSinceEpoch() As Integer`: Returns the number of seconds that have elapsed since midnight on January 1, 1970, as represented by the *roDateTime* instance (not the system time).
- `FromSecondsSinceEpoch(seconds As Integer) As Boolean`: Populates the *roDateTime* instance with the specified number of seconds since midnight on January 1, 1970.
- `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".

roNetworkTimeEvent

This event object is generated by the *roSystemTime* object.

Interfaces: *ifUserData*, *ifNetworkTimeEvent*

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.

The *ifNetworkTimeEvent* interface provides the following:

- `WasSuccessful() As Boolean`: Returns True if the last attempt to set the clock via the network (i.e. NTP or HTTP) was successful.
- `GetFailureReason() As String`: Returns a description of the error if the last attempt to set the clock via the network failed.

roSystemTime

This object provides the ability to read and write the time stored in the real-time clock (RTC). It can also be used to read and write the time-zone setting.

Note: *Dates up to January 1, 2038 are supported.*

Interfaces: *ifUserData*, *ifMessagePort*, *ifSystemTime*

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.

The *ifMessagePort* interface provides the following:

- `SetPort(port As roMessagePort)`: Posts messages of the type *roNetworkTimeEvent* to the attached message port.

The *ifSystemTime* interface provides the following:

- `GetLocalDateTime() As roDateTime`: Returns the current time from the RTC (modulated using the current time zone) as an *roDateTime* instance.
- `GetUtcDateTime() As roDateTime`: Returns the current time from the RTC (modulated using the UTC/GMT time zone) as an *roDateTime* instance.
- `GetZoneDateTime(timezone_name As String) As Object`: Returns the current time from the RTC (modulated using the specified time zone) as an *roDateTime* instance. Supported time zones are listed below.
- `SetLocalDateTime(local_DateTime As roDateTime) As Boolean`: Specifies a new time for the RTC using the current time zone.

- `SetUtcDateTime(utc_DateTime As roDateTime) As Boolean`: Specifies a new time for the RTC using the UTC/GMT time zone.
- `GetTimeZone() As String`: Returns the current time-zone setting of the player. A `POSIX:` value is appended to the beginning of the string if the time zone has been set using the POSIX format.
- `SetTimeZone(zone_name As String) As Boolean`: Specifies a new time-zone setting for the player (supported time zones are listed below). Alternatively, a POSIX formatted time zone can be applied by appending a `POSIX:` value to the beginning of the string.
- `IsValid() As Boolean`: Returns True if the system time is set to a valid value. The time can be set from the RTC or with NTP.
- `GetLastNetworkTimeResult() As roAssociativeArray`: Returns an associative array containing information about the last attempt to set the time via the network:

  Note: *In this associative array, "timestamp" refers to the number of seconds since the player booted. This value can be compared against the total uptime of the player, which is retrieved by calling* `UpTime(0)`.

  - `success_timestamp`: A value indicating when the clock was last set successfully via the network. This value is zero if the clock has never been set successfully via the network.
  - `attempt_timestamp`: A value indicating when the last attempt was made to set the clock via the network. This value is zero if no attempt has been made yet.
  - `failure_reason`: If the last attempt to set the clock via the network failed, this string will contain an error message. If the last attempt was successful, this string will be empty.

Example: The following code specifies a POSIX-formatted time zone:

```
t = CreateObject("roSystemTime")
t.SetTimeZone("POSIX:GMT-0BST-1,M3.5.0/1:00,M10.5.0/2:00")
```

The following are supported system time zones (this list does not apply to POSIX-formatted 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: Australian Central Standard Time (CST) with Daylight Saving Time
- ACST1: Darwin, Australia/Darwin, and Australian Central Standard Time (CST) without Daylight Saving Time
- AEST: Australian Eastern Time with Daylight Saving 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

This object allows the script to trigger events at a specific date/time or during specified intervals. Events are triggered by delivering *roTimerEvent* objects to the specified message port.

Interfaces: *ifTimer*, *ifIdentity*, *ifUserData*, *ifMessagePort*

The *ifTimer* interface provides the following:

- `SetTime(hour As Integer, minute As Integer, second As Integer, millisecond As Integer) As Void`: Sets the time for 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 time occurs.
- `SetTime(a As Integer, b As Integer, c As Integer)`
- `SetDate(year As Integer, month As Integer, day As Integer) As Void`: Sets the date for 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.
- `SetDayOfWeek(day_of_week As Integer) As Void`: Sets the day of week for 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.

Note: *It is possible, using a combination of* `day` *and* `day_of_week` *parameters, 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()`: Stops the timer.

- `SetElapsed(seconds As Integer, milliseconds As Integer)`: Configures a timer to trigger once the specified time period has elapsed. Unlike the absolute timer methods above, changes to the system clock will not affect the period of the `SetElapsed()` timer.

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

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.

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

Example: This code uses `SetElapsed()` to create a timer that triggers every 30 seconds.

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

    timer.SetElapsed(30, 0)

    print "Start at "; Uptime(0)
    timer.Start()

    while true
```

```
       ev = mp.WaitMessage(0)
       if type(ev) = "roTimerEvent" then
           print "Timer event received at "; Uptime(0)
           timer.Start()
       else
           print "Another event arrived: "; type(ev)
       end if
    end while
End Sub
```

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

This event object is generated by the *roTimer* object.

Interfaces: *ifSourceIdentity*, *ifUserData*

The *ifSourceIdentity* interface provides the following.

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

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.

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

roRtspStreamEvent

This object is no longer used to return events related to RTSP streams. The *roVideoPlayer* object now returns events related to an associated *roRtspStream*.

Interfaces: *ifInt*, *ifUserData*

The *ifInt* interface provides the following:

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

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.

roSyncPool

We recommend using *roAssetPool* instead.

Object Creation: The *roSyncPool* object is created with a single parameter that specifies the file path where the pool is located.

```
CreateObject("roSyncPool", pool_path As String)
```

Example:

```
pool = CreateObject ("roSyncPool", "SD:/pool")
```

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

The *ifSyncPool* interface provides the following:

- `ValidateFiles(sync_spec As roSyncSpec, directory As String, options_array As roAssociativeArray) As Object`: Validates the files in the specified directory against the hashes in the specified sync spec. Files that are not in the sync spec are ignored. The options array can currently contain the following optional parameters:
    - DelteCorrupt (Boolean): Automatically delete files that do not match the sync spec. The method will return an associative array that maps each fileneame to an explanation of why it is corrupt. The array only contains corrupt files, so the success is reported by the method returning an empty associative array.
- `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
- SetProxyBypass(a As Array) As Boolean
- SetFileProgressIntervalSeconds(a As Integer) As Boolean
- QueryFiles(a As Object) As Object
- SetFileRetryCount(a As Integer) As Boolean
- SetRelativeLinkPrefix(prefix As String) As Boolean
- BindToInterface(interface As Integer) As Boolean
- EnableUnsafeAuthentication(a As Boolean)
- EnableUnsafeProxyAuthentication(enable As Boolean) As Boolean
- EnableEncodings(enable As Boolean) As Boolean
- SetMaximumPoolSizeMegabytes(maximum_size As Integer) 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.

4.6.x, 4.4.x, 3.10.x

April 29, 2013

1.1 Revised listing and description of Australian time zones in the *roSystemTime* entry.

1.2 Added description of `ifSerialPort.SendBreak()` [implemented in 4.6.14] to the *roSerialPort* entry.

June 11, 2013

2.1 Added a description of the `GetDiagnosticInfo()` method [implemented in version 4.5.11] to the *roTouchScreen* entry.

2.2 Added a description of the `AddGetFromString()` method [implemented in version 4.6.14] to the *roHttpServer* entry.

July 11, 2013

3.1 Added descriptions for all instances of `ifStreamSend.SendBlock()`.

3.2 Added descriptions of `EnableUnsafeProxyAuthentication()` to the *roUrlTransfer* and *roAssetFetcher* entries.

3.3 Added a description of `EnableUnsafeAuthentication()` to the *roAssetFetcher* entry.

July 17, 2013

4.1 Expanded object creation entry for *roAssetCollection*.

4.2 Added a description of objection creation parameters for the *roAssetPoolFiles, roAssetFetcher* and *roAssetRealizer* entries.

July 22, 2013

5.1 Added descriptions of the `ProtectAssets()` and `UnprotectAssets()` methods to the *roAssetPool* entry.

5.2 Added a full description of methods and their parameters to the *roAssetPoolFiles* entry.

5.3 Added a full description of methods and their parameters to the *roAssetRealizer* entry.

August 16, 2013

6.1 Removed the description of `ReadLog()` from the entry for *roSystemLog*.

6.2 Added the following method descriptions to the *roUrlTransfer* entry: `ClearHeaders()`, `AddHeaders()`, `PutFromString()`, `PutFromFile()`, `AsyncPutFromString()`, `AsyncPutFromFile()`, `Delete()`, `AsyncDelete()`.

4.7.x

August 8, 2013

1.1 Added entires for the *roDiskMonitor* and *roDiskErrorEvent* objects.

September 10, 2013

2.1 Added a description for the `roGlobal.EjectDrive()` method.

2.2 Added descriptions of the *roHdmiInputChanged* event object, as well as the `GetHdmiInputStatus()` method, to the *roVideoMode* entry.

October 1, 2013

3.1 Added descriptions for the *roAudioPlayer.SetAudioDelay()* and *roAudioPlayer.SetVideoDelay()* methods (this applies to *roVideoPlayer* as well). Added a description for the related *roAudioOutput.SetAudioDelay()* method as well.

3.2 Added an entry and comprehensive description for *roNetworkStatistics*.

3.3 Added an entry for the *roMediaStreamer* and *roMediaStreamerEvent* objects.

3.4 Expanded the entry for *roIRRemote* to include support for the Pronto Hex Code (PHC) protocol.

3.5 Included additional explanation and an example for *roStorageHotplug*.
3.6 Added descriptions for the *roVideoPlayer.AdjustVideoColor()* and *roVideoMode.AdjustGraphicsColor()* methods.
3.7 Added descriptions for *roVideoMode.GetVideoResX / GetVideoResY* and *roVideoMode.GetOutputResX / GetOutputResY*. Also revised description of *roVideoMode.GetResX / GetResY*.
3.8 Updated and corrected the list of supported baud rates for *roSerialPort.SetBaudRate()* and *roTouchScreen.SetBaudRate()* methods.
3.9 Added a description for the *roSerialPort.SetInverted* method.
3.10 Added descriptions for the *roDateTime.ToSecondsSinceEpoch* and *roDateTime.FromSecondsSinceEpoch* methods.
3.11 Added a description for the *roNetworkConfiguration.SetInboundShaperRate()* method.
3.12 Added an example script to the *roNetworkAdvertisement* entry.
3.13 Expanded description for *roUrlTransfer.SetTimeout()*.
3.14 Revised the descriptions for the *ro*File.Flush()* and *ro*File.AsyncFlush()* methods.
3.15 Added a description for the *roSyncPool.ValidateFiles()* method.

October 17, 2013

1.9 Revised the description of alpha values in *roTextWidget* to reflect the fact that they affect both text and canvas color in 4.7.

October 25, 2013

5.1 Revised the *roMediaStreamer* entry to reflect new functionality.
5.2 Added a description of the `SetMaximumPoolSizeMegabytes()` method to the *roAssetPool* and *roSyncPool* entries.
5.3 Added the *ifUserData* interface to the *roAssetRealizer* object.
5.4 Added entries for the new `SetWatchdogTimout()` and `DeferWatchdog()` methods in the *roMessagePort* section.
5.5 Added entry for new *roTimer.SetElapsed()* method, as well as an example showing how to use *roTimer.SetElapsed().*

November 18, 2013

6.1 Added descriptions for most *roGlobal* methods.

6.2 Expanded the description of the *roNetworkConfiguration.SetHostName()* method.

December 12, 2013

7.1 Added a description for the *roNetworkConfiguration.SetupDWS()* method.

January 28, 2014

8.1 Added a description for the *roHtmlWidget.SetUserAgent()* method. Also added an example of the standard user-agent string reported by WebKit.

8.2 Added a list of standard URL syntax iterations that can be used with *roNetworkConfiguration. SetTimeServer()*.

8.3 Added descriptions for the *roSequenceMatcher* and *roSequenceMatchEvent* objects.

8.4 Add a description of the `SetMatcher()` method to the *roTCPStream* and *roSerialPort* objects.

February 13, 2014

9.1 Added an entry for the new *roTCPConnectEvent.GetSourceAddress()* method (implemented in 4.7.144).

9.2 Revised the description for *roTCPServer.BindToPort* to reflect new functionality (implemented in 4.7.144).

9.3 Added entries and descriptions for the new *roSyncManager* and *roSyncManagerEvent* objects.

9.4 Added a description for the UserString parameter that can be passed to *roAudioPlayerMx*.

9.5 Added a description for the *roAudioEventMx* object.

March 9, 2014

10.1 Added documentation for the *roSqliteDatabase, roSqliteEvent*, and *roSqliteStatement* objects.

10.2 Revised the *roRssParser* script example so that the call to *roTextWidget.EnableForegroundColor()* includes an alpha value.

10.3 Added more descriptive parameters to several *roHtmlWidget* methods.

March 25, 2014

11.1 Added documentation for the new *roVideoMode.Screenshot()* method.

11.2 Clarified some of the information in the entry for *roImagePlayer*.

11.3 Added more descriptive parameter names to several *roHtmlWidget* methods.

April 10, 2014

12.1 Revised the interface listings for *roNetworkConfiguration* and added the *ifMessagePort* and *ifUserData* interfaces.

12.2 Split the *roReadFile, roWriteFile, roReadWriteFile, roAppendFile* section into different sections for each object, and created a new "File Objects" chapter specifically for these objects. Added object descriptions to object entry.

12.3 Added additional information to the *roCecRxFrameEvent* and *roCecTxCompleteEvent* entry (including a description for the objects.

12.4 Modernized some of the information in the *roDeviceInfo* entry.

12.5 Updated several method entries for *roVideoMode*. Also added an entry for the `SaveEdids()` method.

May 12, 2014

13.1 Revised example scripts in the *roImageWidget* entry so that they actually use *roImageWidget* objects.

13.2 Made the parameters of some *roRegex* methods more specific.

13.3 Added descriptions for the `PreloadFile()`, `DisplayFile()`, and `SetTransitionTime()` methods in the *roImagePlayer* entry.

13.4 Added entires for the `Hide()` and `Show()` methods in the *roImagePlayer* entry.

13.5 Added return types and descriptions to *ifSendStream* methods for *roTextField*, *roSerialPort*, *roSystemLog*, and *roTCPStream*.

13.6 Added descriptions for *ifWidget* methods in the *roClockWidget* entry.

May 29, 2014

14.1 Removed legacy information involving mouse cursor bitmaps from the *roTouchScreen* entry. Added a more thorough and up-to-date description for the `SetCursorBitmap()` method.

14.2 Added a deprication notice to the `SetTempDirectory()` method in the entry for *roSqliteDatabase*.
14.3 Added descriptions for most *roSyncSpec* methods.
14.4 Added descriptions for the `ToIsoString()` and `FromIsoString()` methods in the roDateTime entry.
14.5 Added the default transition duration to the description of *roImagePlayer.SetTransitionDuration()*.
14.6 Changed the supported interface in the *roSyncManager* entry from *ifCecInterface* to *ifSyncManager*.

4.8.x

June 26, 2014

1.1 Added a description of the new `SetTransform()` method to the *roVideoPlayer* entry.
1.2 Revised the object creation description for *roTextWidget* to include the new scrolling ticker capabilities. Also added descriptions for the new `SetStringSource()` and `SetAnimationSpeed()` methods.
1.3 Added descriptions for new *roHtmlWidget* methods: `SetLocalStorageDir()`, `SetLocalStorageQuota()`, `SetWebDatabaseDir()`, `SetWebDatabaseQuota()`, `FlushCachedResources()`, `SetHWZDefault()`, `AllowLocalJavaScript()`, and `AllowExternalJavaScript()`.
1.4 Removed the outdated list of video modes in the *roVideoMode* entry in favor of a link to the Supported Resolutions FAQ.
1.5 Added a description of the new `SetGraphicsZOrder()` to the *roVideoMode* entry.
1.6 Added a description of the `ToFront()` and `ToBack()` methods to the *roVideoPlayer* entry.
1.7 Added a description of the new `GetAssetList()` method to the *roAssetCollection* entry.
1.8 Added entries for the new *roBlockCipher* and *roPassKey* methods.
1.9 Added descriptions for the new `SetRectangle()` method to the following objects: *roCanvasWidget*, *roHtmlWidget*, *roTextWidget*, *roClockWidget, roImagePlayer*.
1.10 Added a description for the new `Seek()` method to the *roVideoPlayer*/*roAudioPlayer* objects.
1.11 Added a description for the new `SetFade()` method to the *roVideoPlayer* object.
1.12 Added a description for the new `SetKeyingValue()` method (for setting luma and chroma keys) to *roVideoPlayer* object.

1.13 Revised the *roMediaStreamer* entry to reflect changes and additions to the XD media server.

July 2, 2014

2.1 Clarified the language of the integer list for *roImagePlayer.SetDefaultTransition()*.

July 21, 2014

3.1 Added a description for the `RestartScript()` method to the *roGlobal* entry.

3.2 Added an entry for the *roMediaStreamer* object.

3.3 Expanded the description of *roVideoMode.SetGraphicsZOrder()* to clarify how ordering the graphics plane works in conjunction with ordering the video planes.

August 15, 2014

4.1 Removed the entry for *roTimer.SetIdentity()* because attempting to call it results in an assertion failure.

4.2 Documented the return values for the `GetInt()` method in the *roStreamConnectResultEvent* entry.

4.3 Corrected the *roDeviceInfo.HasFeature()* method parameters for the RS-232 serial port.

4.4 Moved the *roRtspStreamEvent* object entry into the Legacy Objects section.

August 22, 2014

5.1 Corrected the entry for *roAssociativeArray.Lookup()*. It now describes the correct return value for the method.

5.2 Split up the object creation examples for *roImagePlayer* and *roImageWidget*. They are now located in their respective object entries and have more accurate descriptions.

September 15, 2014

6.1 Added a description for the new `SetLayout()` method to the *roKeyboard, roKeyboardPress* entry.

5.0.x

October 1, 2014

1.1 Added a note that *roSqliteDatabase.SetTempDirectory()* was removed in versions 5.0.x.

1.2 Added a caveat about optional and mandatory parameters that are included in associative arrays that are used with *roAssetCollection* instances.

1.3 Clarified the entry for *roAssetRealizer.Realize()*.

1.4 Added new language and an example to *roVideoInput*, clarifying how to use it to display HDMI Input.

1.5 Added a deprication notice to *roVideoPlayer.PlayFileEx()*.

1.6 Added descriptions for the PlayFile() methods in the *roVideoPlayer* entry.

1.7 Added entries for the new *roIRReceiver* and *roIRTransmitter* objects.

Octeober 9, 2014

2.1 Expanded the *roTextWidget.SetAnimationSpeed()* entry to described the different measurements for text modes 0 and 3.

2.2 Clarified when *roHtmlWidget.SetWebDatabaseDir()* should be used.

November 10, 2014

3.1 Added a description for the *roAssetPool.Validate()* method.

3.2 Added a description for the *roHtmlWidget.SetUserStyleSheet()* method.

5.1.x

January 9, 2015

1.1 Listed events generated by the *roIRReceiver* object.

1.2 Documented alternate values that can be used to rotate an *roTextWidget* instance.

1.3 Added a description for the new *roNetworkConfiguration.GetNeighborInformation()* method.

1.4 Added a description for the new *roTextWidget.SetMultiscreen()* method.

1.5 Added descriptions for the new `GetFilePlayability()` and `GetProbePlayability()` methods to the *roVideoPlayer* entry.

1.6 Added descriptions for the new `SetTimeServerIntervalSeconds()` and `GetTimeServerInteralSeconds()` methods to the *roNetworkConfiguration* entry.

1.7 Added a note to the *roUrlTransfer* entry that you must create a separate instance of *roUrlTransfer* for each asset you wish to upload/download.

January 15, 2015

2.1 Removed the `AllowLocalJavaScript()` and `AllowExternalJavaScript()` methods from the *roHtmlWidget* entry. Replaced them with the new `AllowJavaScriptUrls()`.

February 16, 2015

3.1 Added documentation for the `PostJSMessage()` method to the *roHtmlWidget* entry.

3.2 Added an introduction to the *roHttpServer* and *roHttpEvent* methods.

3.3 Added documentation for the `AddMethodFromEvent()` and `AddMethodToFile()` methods in the *roHttpServer* entry.

3.4 Added documentation for the `GetMethod()` method in the *roHttpEvent* entry.

3.5 Added documentation for POSIX support to the *roSystemTime* entry. Also added method descriptions to the entry.

3.6 Added documentation for the `SetPulseParams()`, `SetPulse()`, and `RemovePulse()` to the *roControlPort* entry.

3.7 Fixed a typo with the `auth_password` parameter in the associative-array description for *roAssetCollection*.

3.8 Documented the new `SyncIsoTimeStamp` parameter for the `Pause()` and `Resume()` methods in the *roVideoPlayer* entry.
3.9 Expanded the *roControlPort* entry to include information about configuring BP200/BP900 boards, as well as GPIO LEDs.
3.10 Documented the `RunGarbageCollector()` method in the *roGlobal* entry.
3.11 Documented the `SetPlaybackSpeed()` method in the *roVideoPlayer* entry.
3.12 Documented the new `response_pipe` parameters (implemented in 5.1.37) for the *roUrlTransfer.AsyncMethod()* method.
3.13 Documented the `GetHash()`, `GetPrefix()`, `GetUserData()`, and `GetUserData()` methods in the *roUrlEvent* entry.

February 16, 2015

4.1 Documented additional video file guidelines for *roSyncManager*.

February 16, 2015

5.1 Documented the methods offered by the *ifUserData* interface in various sections.
5.2 Documented the *roIRDownEvent*, *roIRRepeatEvent*, and *roIRUpEvent* objects.

March 22, 2015

6.1 Added descriptions for some methods that are part of the *ifStringOps* interface.
6.2 Revised the method descriptions for some *roGlobal* methods.
6.3 Expanded the documentation for the `ReserveMegabytes()` and `SetMaximumPoolSizeMegabytes()` methods in the *roAssetPool* entry.
6.4 Added documentation for codes that can be returned by the *roAssetFetcherEvent.GetResponseCode()* method.

April 2, 2015

7.1 Added a description for the *roAssetFetcherEvent.GetEvent()* method.

7.2 Added a description for the *roHtmlWidget.StartInspectorServer()*

April 8, 2015

8.1 Added a description of the `SetTransform()` method to the *roImagePlayer* entry.
8.2 Added descriptions for the `CreateTestHole()` and `GetRectangle()` methods to the *roImagePlayer* entry.
8.3 Removed methods that are not actually offered by the *roAssetFetcherEvent* object.
8.4 Documented various methods offered by *roAssetFetcher*, *roAssetFetcherEvent*, and *roAssetFetcherProgressEvent*.
8.5 Added documentation for the `EnableResume()` method to the *roUrlTransfer* entry.
8.6 Documented various methods offered by the *roByteArray.ifByteArray* interface.
8.7 Provided further description fo the *roSqliteEvent.GetSqlResult()* method.
8.8 Documented the *roHttpServer.SetupDWSLink()* method.
8.9 Clarified the operation of the *roTextField.SetAnimationSpeed()* method when it is used with mode 3 (scrolling ticker).

April 16, 2015

9.1 Included additional information about using the *roSyncManager* object.

April 23, 2015

10.1 Included descriptions for the `SetPreferredVideo()`, `SetPreferredAudio()`, and `SetPreferredCaptions()` methods in the *roVideoPlayer* entry.

May 11, 2015

11.1 Added video profile support to the description of the *roVideoMode.SetMode().*
11.2 Added descriptions for the `SetHmacKey()` and `SetObfuscatedHmacKey()` methods in the *roHashGenerator()* entry.
11.3 Added descriptions for the `SetObfuscatedWiFiPassphrase()` and `SetObfuscatedLoginPassowrd()` methods in the *roNetworkConfiguration* entry.

6.0.x

July 24, 2015

1.1 Documented the new `GetBufferByteArray()` and `GetBufferMetadata()` methods in the *roImageBuffer* entry.
1.2 Documented the new `SetFlowControl()` method in the *roSerialPort* entry.
1.3 Added an entry for the new *roDeviceCustomization* object.
1.4 Documented the new `FactoryReset()` method in the *roDeviceCustomization* entry.
1.5 Documented the new `WriteSplashScreen()` method in the *roDeviceCustomization* entry.
1.6 Documented the inclusion of the `EnableEncodings()` method with the *roAssetFetcher* and *roSyncPool* objects.
1.7 Revised the documentation for the `EnableEncodings()` method in the *roUrlTransfer* entry.
1.8 Documented the new `SetProxyBypass()` method in the *roNetworkConfiguration*, *roUrlTransfer*, and *roAssetFetcher* entries.
1.9 Documented the new `GetProxyBypass()` method in the *roNetworkConfiguration* entry.
1.10 Added an entry for the *roHdmiInputChanged* and *roHdmiOutputChanged* event objects.
1.11 Documented the new `GetHdmiOutputStatus()` method in the *roVideoMode* entry.
1.12 Revised the description for the *roAudioPlayer.SetVolume()* method (this also applies to the *roVideoPlayer* implementation of the *ifAudioControl* interface).
1.13 Documented the new `GetStreamInfo()` and `GetStreamStatistics()` methods for the *roVideoPlayer* object.
1.14 Revised the documentation for the *roNetworkConfiguration.SetupDWS()* entry.
1.15 Documented the new *roNetworkConfiguration.EnableLeds()* method.
1.16 Documented the new UPnP functionality, which includes the *roUPnPController*, *roUPnPSearchEvent*, *roUPnPDevice*, *roUPnPService*, *roUPnPServiceEvent*, *roUPnPActionResult* objects.
1.17 Documented the new *roVideoPlayer.GetPlaybackPosition()* method.
1.18 Revised documentation for the *roVideoMode.SetMode()* method
1.19 Documented the new `GetActiveMode()` and `GetConfiguredMode()` methods for the *roVideoMode* object.
1.20 Documented the new *roTextWidget.SetSeparator()* method.

1.21 Documented the new *roTouchScreen.SetCalibrationRanges()* method.
1.22 Documented the new *roDeviceInfo.GetLoadStatistics()* method.
1.23 Documented the new `FirmwareIsAtLeast()` and `BootFirmwareIsAtLeast()` methods for the *roDeviceInfo* entry.
1.24 Documented the new *roAudioOutput.SetTone()* method.
1.25 Removed the *ifIdentity* interface from the *roMediaStreamer* entry.
1.26 Removed the *ifMessagePort* and *ifUserData* interfaces from the *roNetworkConfiguration* entry.
1.27 Documented the addition of the *ifUserData* interface to the following objects; *roVideoPlayer*, *roAudioPlayer*, *roKeyboard*, *roMediaServer*, *roCecInterface*, *roMimeStream*, *roRtspStream*, *roRtspStreamEvent*, *roSnmpAgent*, *roSnmpEvent*, *roStorageHotplug*, *roStorageAttached*, *roStorageDetached*, *roMediaStreamer*, *roTimer*, *roTimerEvent*, *roMimeStreamEvent*.
1.28 Removed the *ifStringOps* interface from the *roStorageAttached*, *roStorageDetached*, and *roSnmpEvent* objects.
1.29 Removed the *ifIntOps* from the *roHdmiInputChanged*, *roHdmiOutputChanged*, *roIRRemotePress*, *roGpioButton*, *roTouchCalibrationEvent*, *roNetworkAttached*, and *roNetworkDetached* objects.
1.30 Removed the `SetTempDirectory()` method from the ro*SQLiteDatabase* entry.

August 8, 2015

2.1 Expanded the description of the *roHttpServer.AddPostToFile()* method.
2.2 Revised the description of the *roVideoMode.GetEdidIdentity()* method.
2.3 Documented the new `GetDuration()` method in the *roVideoPlayer* entry.
2.4 Documented the new `SetMosaic()`, `SetDecoder()`, and `SetMaxResolution()` methods in the *roVideoPlayer* entry.
2.5 Added the *ifUserData* and *ifIdentity* interfaces to the *roControlPort* entry.
2.6 Added the *ifUserData* interface to the *roVideoMode* entry.
2.7 Corrected several entries that listed the *ifSetMessagePort* interfeace rather than the *ifMessagePort* interface.
2.8 Documented the new file decryption capabilities of the *roVideoPlayer.PlayFile()* method.
2.9 Added an entry for the new *roNetworkTimeEvent* object.

2.10 Documented the new `GetLastNetworkTimeResult()` method in the *roSystemTime* entry.
2.11 Documented the new *roPtp* and *roPtpEvent* objects.
2.12 Revised the *roAssetPool* and *roAssetCollection* entries to include support for the MD5 algorithm.

August 28, 2015

3.1 Added additional parameters to the return value of *roVideoMode.GetHdmiOutputStatus()*.
3.2 Added additional parameters to the return value fo *roVideoMode.GetConfiguredMode()*.
3.3 Added additional parameters to the return value of *roVideoPlayer.GetStreamInfo()*.
3.4 Documented the `AddHeader()` and `ClearHeaders()` methods in the *roRtspStream()* entry.
3.5 Documented changes to the *roVideoPlayer.SetLoopMode()* method.
3.6 Added *ifUserData* and *ifMessagePort* interfaces to the *roIRTransmitter* entry.
3.7 Added the new `AsyncSend()` method to the *roIRTransmitter* entry.
3.8 Added the new *roIRTransmitCompleteEvent* object.
3.9 Added the new `GetTxHDCPStatus()` method to the *roVideoMode* entry.
3.10 Revised the `SetProxyBypass()` documentation in the *roAssetFetcher* and *roUrlTransfer* entries.
3.11 Revised the documentation for the `SetPcmAudioOutputs()` and `SetCompressedAudioOutputs()` methods in the *roAudioPlayer* entry.
3.12 Documented changes to the return value types of the `GetHDMIInputStatus()` and `GetHDMIOutputStatus()` methods in the *roVideoMode* entry.

September 24, 2015

4.1 Revised the description of the *roRtspStream* object.
4.2 Revised the description of the *roVideoPlayer.PlayFile()* method.
4.3 Added a description for the `ForceGpuRasterization()` method to the *roHtmlWidget* entry.
4.4 Added a description for the `SetImageSizeThreshold()` method to the *roVideoMode* entry.
4.5 Added descriptions for the `Hide()` and `Show()` methods to the *roVideoPlayer* entry.
4.6 Cleaned up various out of date information at the end of the *roVideoPlayer* entry.

4.7 Added a description for the `UseInitiatorAddressFromPacket()` method to the *roCecInterface* entry.

4.8 Revised the **Mutiple Video Screen Matrix** section in the *roVideoPlayer()* entry to include information about portrait-mode video.

November 18, 2015

5.1 Documented a change in behavior to the *roHtmlWidget.SetLocalStorageDir()* method (change occurred in the 6.0 release).

5.2 Documented the `OverlayImage()` method in the *roImageWidget* entry.

5.3 Updated the `SetSendEol()` and `SetReceiveEol()` method descriptions in various entries to include further information.

5.4 Documented the `ForceHDCPOn()`, `DisableHDCPRepeater()`, and `GetFPS()` methods in the *roVideoMode* entry.

5.5 Documented file decryption support for image display methods on the *roImagePlayer*, *roImageWidget*, *roVideoPlayer*, *roCanvasWidget*, *roClockWidget*, and *roTickerWidget* objects.

5.6 Documented changes to the `:auto` and `:preferred` parameters for the *roVideoMode.SetMode()* method.

December 11, 2015

6.1 Improved documentation for the *roTouchScreen* and *roTouchEvent* objects.

6.2 Revised the `SetFade()` and `AddEvent()` methods, along with the Video Timecode Events section, in the *roVideoPlayer* entry.

6.3 Updated the list of event values for *roVideoEvent*/*roAudioEvent*.