BrightScript 3.0 Reference

Roku Streaming Player Version 4.1 12/21/2011

Table of Contents

.

.

.

1.0 Introduction

Roku BrightScript is a powerful scripting language that makes it easy and quick to build media and networked applications for embedded devices. The language has integrated support for BrightScript Components, a library of lightweight components. The APIs of the platform (device) the BrightScript is running on are all exposed to BrightScript as BrightScript Components.

This document specifies the syntax of the language. To write useful applications, you should also refer to the BrightScript Component Reference Manual for the device you are targeting code for. This manual is designed for people that have some experience programming software. It is a reference guide, not a tutorial.

BrightScript compiles code into bytecode that is run by an interpreter. This compilation step happens every time a script is loaded and run. There is no separate compile step that results in a binary file being saved. In this way it is similar to JavaScript.

BrightScript statement syntax is not C-like; in this way it is similar to Python or Basic or Ruby or Lua. BrightScript Objects and named entry data structures are Associative Arrays; in this way it is similar to JavaScript or Lua. BrightScript supports dynamic typing (like JavaScript), or declared types (like C or Java). BrightScript uses "interfaces" and "components" for its APIs; similar to ".Net" or Java.

BrightScript is a powerful bytecode interpreted scripting language optimized for embedded devices; in this way it is unique. For example, BrightScript and the BrightScript Component architecture are written in 100% C for speed, efficiency, and portability. BrightScript makes extensive use of the "integer" type (since many embedded processors don't have floating point units). This is different from languages like JavaScript where a number is always a float. BrightScript numbers are only floats when necessary.

If you want to get a quick flavor of BrightScript code, see the Appendix of this manual for the game "snake".

BrightScript is optimized to be the "glue" that connects underlying components for network connectivity, media playback, and UI screens into user friendly applications with minimal programmer effort.

.

2.0 Statement Summary

BrightScript supports the following familiar looking statement types:

If / Then / Else If / Else / End If
For / To / End For / Step / Exit For
For Each / In / End For / Exit For
While / End While / Exit While
Function / End Function / As / Return
Print
Rem (or ')
Goto
Dim
End
Stop

BrightScript is not case sensitive.

Each statement's syntax is documented precisely later in the manual.

Here is an example:

```
Function Main() As Void

    dim cavemen[10]

    cavemen.push("fred")
    cavemen.push("barney")
    cavemen.push("wilma")
    cavemen.push("betty")

    for each caveman in cavemen
        print caveman
    end for

End Function
```

Each line may contain a single statement, or a colon (:) may be used to separate multiple statements on a single line.

```
myname = "fred"
if myname="fred" then yourname = "barney":print yourname
```

.

3.0 Expressions, Variables, and Types

3.1 Identifiers

Identifiers (names of variables, functions, labels, or object member functions or interfaces (appear after a ".")) have the following rules.

- must start with an alphabetic character (a – z)
- may consist of alphabetic characters, numbers, or the symbol "_" (underscore)
- are not case sensitive
- may be of any length
- may not use a "reserved word" as the name (see appendix for list of reserved words).
- if a variable: may end with an optional type designator character ($ for string, % for integer, ! for float, # for double) (function names do not support a type designator character).

For example:

```
a
boy5
super_man$
```

3.2 Types

BrightScript uses dynamic typing. This means that every value also has a type determined at run time. However, BrightScript also supports declared types. This means that a variable can be made to always contain a value of a specific type. If a value is assigned to a variable which has a specific type, the type of the value assigned will be converted to the variables type, if possible. If not possible, a runtime error will result.

The following types are supported in BrightScript:

- **Boolean** – either true or false
- **Integer**– 32 bit signed integer number
- **Float** – the smallest floating point number format supported by the hardware or software
- **Double** - the largest floating point number format supported by the hardware or software. Although Double is an intrinsically understood type, it is implemented internally with the roIntrinsicDouble component. This is generally hidden from the developer.
- **String**. a sequence of ASCII characters. Currently strings are ASCII, not UTF-8. Internally there are two intrinsic string states. The first is for constant strings. For example, `s="astring"`, will create an intrinsic constant string. But once a string is used in an expression, it becomes an "roString". For example: `s=s+"more",` results in s becoming an "roString". If this is followed by an `s2=s`, s2 will be a reference to s, not a copy. This behavior of reference counting strings is new in version 3 of BrightScript.
- **Object** – a reference to a BrightScript Component (native component). Note that if you use the "type()" function, you will not get "Object". Instead you will get the type of object. E.g.: "roList", "roVideoPlayer", etc. Also note that there is no special type for "intrinsic" BrightScript objects (verses a BrightScript Component). Intrinsic BrightScript objects are all built on the BrightScript Component type "roAssociativeArray".
- **Function** – Functions (and Subs, which are functions with void return types) are an intrinsic type. They can be stored in variables and passed to functions.
- **Interface**- An interface in a BrightScript Component. If a "dot operator" is used on an interface type, the member must be static (since there is no object context).
- **Invalid** – the type invalid has only one value – invalid. It is returned in various cases, for example, when indexing an array that has never been set.
- **Dynamic typing** – Unless otherwise specified, a variable is dynamically typed. This means that the type is determined by the value assigned to it at assignment time. For example "1" is an integer, "2.3" is a float, "hello" is a string, etc. If a dynamically typed

.

variable is assigned a new value, its type may change. For example: a=4 creates "a" as integer, then a = "hello", changes the variable "a" to a string.

All variables are dynamically typed, unless: (a) the variable ends in a type designator character, or (b) the "As" keyword is used in a function declaration with the variable.

BrightScript 3 Compatibility

BrightScript 3 has few changes that effect types. These are hidden from the programmer, unless the new optional "3" parameter is passed to "type()". Without this parameter, "type()" will return a BrightScript 2.1 compatible type.

These are the changes:

- Non-constant strings are now stored in an "roString" component and reference counted when assigned (they were copied in version 2.1, and stored in an intrinsic type)
- Double's are now stored in an "roIntrinsicDouble" component. However, their type is still "Double". This change results in a memory use reduction most of the time (except when Doubles are actually used, then a bit more memory is used).
- Container components now store intrinsic typed values, not an auto boxed values. In other words, a=[5] will store an integer 5 in an array, not an roInteger component. This change causes less memory to be used, and execution to be faster.

Here are some examples. Note: ? is a short cut for the "print" statement.

```
BrightScript> ?type(1)
Integer

BrightScript> ?type(1.0)
Float

BrightScript> ?type("hello")
String

BrightScript> ?type(CreateObject("roList"))
roList

BrightScript> ?type(1%)
Integer

BrightScript> b!=1
BrightScript> ?type(b!)
Float

BrightScript> c$="hello"
BrightScript> ?type(c$)
String

BrightScript> d="hello again"
BrightScript> ?type(d)
String

BrightScript> d=d+d
BrightScript> ?type(d)
String

BrightScript> ?type(d, 3)
```

.

```
roString

BrightScript> d=1
BrightScript> ?type(d)
Integer

BrightScript> d=1.0
BrightScript> ?type(d)
Float
```

3.3 Literals (Constants)

Type Boolean: true, false
Type Invalid: invalid
Type String: String in quotes, eg "this is a string"
Type Integer: Hex integer, eg. &HFF, or decimal integer, eg. 255
Type Float: e.g., 2.01 or 1.23456E+30 or 2!
Type Double: eg, 1.23456789D-12, or .2.3#
Type Function , eg: MyFunction
Type Integer: LINE_NUM – the current source line number.

The following rules determine how integers, doubles, and floats are determined:

1. If a constant contains 10 or more digits, or if D is used in the exponent, that number is double precision. Adding a # declaration character also forces a constant to be double precision.

2. If the number is not double-precision, and if it contains a decimal point, then the number is float. If the number is expressed in exponential notation with E preceding the exponent, the number is float.

3. If neither of the above is true of the constant, then it is an integer.

Array "literal"
The Array Operator [ ] can be used to declare an array. It may contain literals (constants), or expressions. E.g.:

```
Myarray = []
Myarray = [ 1, 2, 3]
Myarray = [ x+5, true, 1<>2, ["a","b"]]
```

Associative Array "literal"
The { } operator can be used to define an Associative Array. It can contain literals or expressions. E.g.:

```
aa={ }
aa={key1:"value", key2: 55, key3: 5+3 }
```

Both Arrays and Associative Arrays can also have this form:

```
aa = {
   Myfunc1: aFunction
   Myval1 : "the value"
}
```

Note on Invalid vs. Object

.

Certain functions that return objects can also return invalid (for example, in the case when there is no object to return). In which case, the variable accepting the result must be dynamic, since it may get "invalid" or it may get an "object".

```
l=[]
a$=l.pop()
```

This example will return a type mismatch (a$ is a string, and can not contain "invalid"). Many functions that return objects can return invalid as well

3.4 Type Declaration Characters

A type declaration character may be used at the end of a variable or literal to fix its type. Variables with the same identifier but separate types are separate variables. For example, a, a$, and a% are all independent.

Character	Type	Examples
$	String	A$, ZZ$
%	Integer	A1%, SUM%
!	Single-Precision (float)	B!, N1!
#	Double-Precision (double)	A#, 1/3#, 2#

3.5 Type Conversion (Promotion)

When operations are performed on one or two numbers, the result must be typed as integer, double or single-precision (float). When a +, -, or * operation is performed, the result will have the same degree of precision as the most precise operand. For example, if one operand is integer, and the other double-precision, the result will be double precision. Only when both operands are integers will a result be integer. If the result of an integer *, -, or + operation is outside the integer range, the operation will be done in double precision and the result will be double precision.

Division follows the same rules as +, * and -, except that it is never done at the integer level: when both operators are integers, the operation is done as float with a float result

.

During a compare operation (< , >,=,etc.) the operands are converted to the same type before they are compared. The less precise type will always be converted to the more precise type.

3.6 Effects of Type Conversions on Accuracy

When a number is converted to integer type, it is "rounded down"; i.e., the largest integer, which is not greater than the number is used. (This is the same thing that happens when the INT function is applied to the number.)

When a number is converted from double to single precision, it is "4/5 rounded" (the least significant digit is rounded up if the fractional part > =5. Otherwise, it is left unchanged).

When a single precision number is converted to double precision, only the seven most significant digits will be accurate.

.

3.7 Operators

Operations in the innermost level of parentheses are performed first, and then evaluation proceeds according to the precedence in the following table. Operations on the same precedence are left associative, except for exponentiation, which is right associative.

( ) Function call, or Parentheses
. , [] Array Operator
^ (Exponentiation)
–, + (Negation)
*, /, MOD
+, -
<, >, = , <>, <=, >=
NOT
AND
OR

New In BrightScript 3: MOD

3.8 String Operators

The following operators work with strings
<. >, =, <>, <=, >=, +

3.9 Function References

= , <> work on variables that contain function references and function literals

3.10 Logical and Bitwise Operators

Example:

```
if a=c and not(b>40) then print "success"
```

AND, OR and NOT can be used for logical (Boolean) comparisons, or bit manipulation and bitwise comparisons. If the arguments to these operators are Boolean, then they perform a logical operation. If the arguments are numeric, they perform bitwise operations.

```
x = 1 and 2   ' x is zero
y = true and false  ' y is false
```

When AND and OR are used for logical operations, only the necessary amount of the expression is executed. For example:

```
print true or invalid
```

The above statement will print "true", where as:

```
print false or invalid
```

.

Will cause a runtime error because "invalid" is not a valid operand for OR.

3.11 "dot" Operator

The "." Operator can be used on any BrightScript Component. It also has special meaning when used on any roAssociativeArray, roXMLElement or roXMLList. When used on a BrightScript Component, it refers to an interface or a member function. For example:

```
i=CreateObject("roInt")
i.ifInt.SetInt(5)
i.SetInt(5)
```

"ifInt" is the interface, and "SetInt" is the member function. Every member function of a BrightScript Component is part of an interface. However, specifying the interface with the dot operator is optional. If it is left out, as in the last line of the example above, each interface in the object is searched for the member function. If there is a conflict (a member function with the same name appearing in two interfaces), then the interface should be specified.

When the "." Operator is used on an Associative Array, it is the same as calling the Lookup() or AddReplace() member of the AssociativeArray Object.

```
aa=CreateObject("roAssociativeArray")
aa.newkey="the value"
print aa.newkey
```

The "." Operator's parameters are set at compile time – they are not dynamic (unlike the Lookup() or AddReplace() functons).

The dot operator is always case insensitive. By convention, a statement like:

```
aa.NewKey=55
```

Will actually create the Associative Array entry in all lower case ("newkey").

See the section on XML support for details on using the dot operator on xml objects.

3.12 Array/Function Call Operator

The "[ ]" operator is used to access an Array (any BrightScript Component that has an "ifArray" interface, such as roArray and roList). It can also be used to access an AssociativeArray.

The function call operator "( )" can be used to call a function. When used on a Function literal (or variable containing a function reference), it calls the Function.

Examples:

```
aa=CreateObject("roAssociativeArray")
aa["newkey"]="the value"
print aa["newkey"]

array=CreateObject("roArray", 10, true)
array[2]="two"
print array[2]

fivevar=five
print fivevar()
```

.

```
array[1]=fivevar
print array[1]()   ' print 5

function five() As Integer
   return 5
end function
```

The "[ ]" operator takes expressions that are evaluated at runtime and so is different than a "." Operator in this way. The dot operator takes compile time identifiers.

Arrays in BrightScript are one dimension. Multi-dimension arrays are implemented as arrays of arrays. The "[ ]" operator will automatically map "multi-dimensionality". IE, the following two expressions to fetch "item" are the same:

```
dim array[5,5,5]
item = array[1][2][3]
item = array[1,2,3]
```

(**NOTE: if a multi-dimension array grows beyond its hint size the new entries are not automatically set to roArray**)

3.13 = Operator

"=" is used for both assignment and comparison. Example:

```
a=5
If a=5 then print "a is 5"
```

BrightScript does not support the use of the "="Assignment operator inside an expression (like C does). This is to eliminate the common class of bugs where a programmer meant "comparison", not "assignment".

When an assignment occurs, intrinsic types are copied, but BrightScript Objects are reference counted.

.

4.0 BrightScript Components, Interfaces, and Language Integration

The BrightScript Component architecture and library are separate from BrightScript, but BrightScript requires them.

- All APIs exposed to BrightScript are exposed as BrightScript Components. In other words, if a platform wants to expose APIs to be scripted, the platform must register a new BrightScript Component. The BrightScript Component will most likely be written in C or C++.
- BrightScript has language features that are designed to work with BrightScript Component Interfaces. These include: for each, print, the array operator, dot operator, and intrinsic objects.
- Fundamental BrightScript building blocks are implemented as BrightScript Components. For example: Lists, Vector Arrays, Associative Arrays, and Objects.

4.1 Brief Summary of BrightScript Components

BrightScript Components are light weight components that are implemented in C (or a C compatible language such as C++). C++ templates exist to help C++ programmers implement the key C functions needed to implement a BrightScript Component.

BrightScript Components are robust against version changes. In other words, scripts are generally backwards compatible with Objects that have undergone version improvements.

BrightScript Components keep a reference count and delete themselves when the reference count goes to zero.

A key BrightScript Component concept is the Interface. The term Interface is used here as it is in Java or Microsoft COM. An interface is a known set of member functions that implement a set of logic. In some ways an Interface is like a virtual base class in C++. Any script or C-compatible program can use an object's interface without regard to what type of object it is a part of, as long as it is familiar with a particular interface.

For example, the standard serial interface (RS-232) object implements three interfaces: "ifSerialControl", "ifStreamReceive", and "ifStreamSend". Since the BrightScript "print" statement sends its output to any object that has an "ifStreamSend" interface, it works with the serial object (and others).

4.2 BrightScript statements that work with BrightScript Component Interfaces

For each
The for-each statement works on any object that has an "ifEnum" interface. These include: Array, Associative Array, List, ByteArray, and Message Port.

Print
The print #*object*, "hello" format will print "into" any object that has an "ifStreamSend" interface. These include the TextField and SerialPort objects.

If the expression being printed evaluates to an object that has an "ifEnum" interface, print will print every item that can be enumerated.

.

In addition to printing the values of intrinsic types, "print" will also print any object that exposes one of these interfaces: ifString, ifInt, ifFloat..

Wait
The wait function will work on any object that has an "ifMessagePort" interface.

Array Operator –"[]"
The array operator works on any object that has an "ifArrayGet" or "ifArraySet" interface. This includes Array, Associative Array, ByteArray, and Lists.

Member access operator "."
The "." Operator works on any object that has an "ifAssociativeArray" interface (as well as on any BrightScript Component (when calling a member function)). It also has special meaning when used on roXMLElement or roXMLList.

Expression Parsing
Any expression that is expecting an Integer, Float, Double, Boolean or String, can take an object with the "ifInt", "ifFloat", "ifDouble", "ifBoolean" or "ifString" interface.

4.3 Wrapper Objects and intrinsic type promotion

The intrinsic BrightScript types integer, float, double, string, invalid, boolean and function all have object equivalents. If one of these intrinsic types is passed to a function that expects an Object, the appropriate wrapper object will be created, assigned the correct value, and passed to the function. This is sometimes referred to as "autoboxing".

In versions prior to 3.0 containers made extensive use of autoboxing. This is how, for example, roArray could store integers and strings as well as objects. However, starting with BrightScript 3.0, containers store typed values, and so autoboxing is much less common.

For example:

```
Function Main()
      MyFunA(4)
      MyFunB(4)
      Stop
End Function

Function MyFunA(p as object)
      print "A",p,type(p)
End Function

Function MyFunB(p)
      print "B",p,type(p)
End Function
```

Will Print:

```
A                4              roInt
B        4       Integer
```

4.4 Use of wrapper functions on intrinsic types

If a function that would be available on a wrapper object is used on an intrinsic type, the value will be autoboxed.

.

For example

```
Print 5.tostr()+"th"
Print "5".toint()+5
```

Note that

```
-5.tostr()    'will cause an error.  Use:
(-5).tostr()
```

More examples:

```
if type(5.tostr())<> "String" then stop
if (-5).tostr()<>"-5" then stop
if (1+2).tostr()<>"3" then stop
if 5.tostr()<>"5" then stop
i=-55
if i.tostr()<>"-55" then stop
if 100%.tostr()<>"100" then stop
if (-100%).tostr()<>"-100" then stop
y%=10
if y%.tostr()<>"10" then stop

if "5".toint()<>5 or type("5".toint())<>"Integer" then stop
if "5".tofloat()<>5.0 or type("5".tofloat())<>"Float" then stop
fs="-1.1"
if fs.tofloat()<>-1.1 or fs.toint()<>-1 then stop

if "01234567".left(3)<>"012" then stop
if "01234567".right(4)<>"4567" then stop
if "01234567".mid(3)<>"34567" then stop
if "01234567".mid(3,1)<>"3" then stop
if "01234567".instr("56")<>5 then stop
if "01234567".instr(6,"56")<>-1 then stop
if "01234567".instr(0,"0")<>0 then stop
```

4.5 BrightScript XML Support

BrightScript supports XML via two BrightScript Components, and some dedicated language features. The BrightScript Component roXMLElement provides support for parsing, generating, and containing XML. In addition, the roXMLList object is often used to hold lists of roXMLElement, and implements the BrightScript standard ifList interface as well as the ifXMLList interface. Language features are provided via the dot operator, and the @ operator.

Dot Operator on XML

1. When applied to an roXMLElement, the dot operator returns an roXMLList of children that match the dot operand. If no tags match, an empty list is returned
2. When applied to an roXMLList, the dot operator aggregates the results of performing the dot operator on each roXMLElement in the list.
3. When used on XML, which is technically case sensitive, the dot operator is still case insensitive. If you wish to do a case sensitive XML operation, don't use the dot operator. Use the XML member functions.

Attribute Operator

The @ operator can be used on an roXMLElement to return a named attribute. It is always case insensitive (despite the fact that XML is technically case sensitive). When used on an roXMLList, the @ operator will return a value only if the list contains exactly one element.

For example, if the file "example.xml" contains the following:

.

```
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
  <photos page="1" pages="5" perpage="100" total="500">
    <photo id="3131875696" owner="21963906@N06" secret="f248c84625"
      server="3125" farm="4" title="VNY 16R" ispublic="1" isfriend="0"
      isfamily="0" />
    <photo id="3131137552" owner="8979045@N07" secret="b22cfde7c4"
      server="3078" farm="4" title="hoot" ispublic="1" isfriend="0"
      isfamily="0" />
    <photo id="3131040291" owner="27651538@N06" secret="ae25ff3942"
      server="3286" farm="4" title="172 • 365 :: Someone once told me..."
      ispublic="1" isfriend="0"
  </photos>
</rsp>
```

Then

```
rsp=CreateObject("roXMLElement")
rsp.Parse(ReadAsciiFile("tmp:/example.xml"))

? rsp.photos.photo
```

Will return an roXMLList with three entries.

```
? rsp.photos.photo[0]
```
Will return an roXMLElement reference to the first photo (id="**3131875696").**

```
? rsp.photos
```
Will return an roXMLList reference containing the photos tag.

```
rsp.photos@perpage
```
Will return the string 100.

Use the Text() method to return an element's text.

For example, if the variable booklist contains this roXMLElement:

```
      <booklist>
         <book lang=eng>The Dawn of Man</book>
      </booklist>

Print booklist.book.gettext()
```

Will print "The Dawn of Man".

```
print booklist.book@lang
```

Will print "eng".

example flikr code clip

```
REM
REM Interestingness
REM pass an (optional) page of value 1 - 5 to get 100 photos
REM starting at 0/100/200/300/400
REM
REM returns a list of "Interestingness" photos with 100 entries
REM
```

.

```
Function GetInterestingnessPhotoList(http as Object, page=1 As Integer) As Object

       print "page=";page


http.SetUrl("http://api.flickr.com/services/rest/?method=flickr.interestingness
.getList&api_key=YOURKEYGOESHERE&page="+mid(stri(page),2))

    xml=http.GetToString()

    rsp=CreateObject("roXMLElement")
    if not rsp.Parse(xml) then stop

       return helperPhotoListFromXML(http, rsp.photos.photo)
'rsp.GetBody().Peek().GetBody())


End Function

Function helperPhotoListFromXML(http As Object, xmllist As Object,
          owner=invalid As dynamic) As Object

    photolist=CreateObject("roList")
    for each photo in xmllist
       photolist.Push(newPhotoFromXML(http, photo, owner))
    end for
    return photolist

End Function

REM
REM newPhotoFromXML
REM
REM    Takes an roXMLElement Object that is an <photo> ... </photo>
REM    Returns an brs object of type Photo
REM       photo.GetTitle()
REM       photo.GetID()
REM       photo.GetURL()
REM       photo.GetOwner()
REM

Function newPhotoFromXML(http As Object, xml As Object, owner As dynamic) As Object
    photo = CreateObject("roAssociativeArray")
    photo.http=http
    photo.xml=xml
    photo.owner=owner
    photo.GetTitle=function():return m.xml@title:end function
    photo.GetID=function():return m.xml@id:end function
    photo.GetOwner=pGetOwner
    photo.GetURL=pGetURL
    return photo
End Function

Function pGetOwner() As String
       if m.owner<>invalid return m.owner
       return m.xml@owner
End Function

Function pGetURL() As String
       a=m.xml.GetAttributes()
       url="http://farm"+a.farm+".static.flickr.com/"+a.server+"/"+a.id+"_"+a.se
cret+".jpg"

.
```

```
      return url
End Function
```

4.6 Garbage Collection

BrightScript will automatically free strings when they are no longer used, and it will free objects when their reference count goes to zero. This is done at the time the object or string is no longer used; there is no background garbage collection task. This results in very predictable "garbage collection" -- there are no unexpected stalls in execution.

A "mark and sweep" garbage collection is run after a script executes, or can be manually forced to run via the debug console. A script can force the Garbage Collector to run via the RunGarbageCollector() function. The Garbage Collector's purpose is to clean up objects that refer to themselves or have other circular references (which are not managed by the normal reference counting garbage collection).

```
i=roCreateObject("roInt")
j=i          ' reference incremented
i=invalid    ' reference decremented
j=0          ' roInt just free'd.
```

4.7 Events

Events in BrightScript center around an event loop and the "roMessagePort" BrightScript Component. Any BrightScript Component can be posted to a message port. Typically these will be Objects that are designed to represent an events. For example, the "roTimer" class posts events of type "roTimerEvent".

Example:

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

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

         REM ignore buttons pressed while flashing led above
         while p.GetMessage()<>invalid
         end while
      end while
```

Note that the following two lines:

```
      while true
         msg=wait(0, p)
```

.

Could be replaced with:

```
For each msg in p
```

And then the last "end while" would become a "end for".

4.8 Threading Model

A BrightScript script runs in a single thread. The general rule of thumb is that BrightScript Component calls are synchronous if they return quickly, or asynchronous if they take a long time to complete. For example, class roArray methods are all synchronous. But if "roVideoPlayer" is used to play a video, the play method returns immediately (it is asynchronous). As the video plays, it will post events to a script created message port. Typical events would be "media finished" or "frame x reached".

Whether a BrightScript Component launches a background thread to perform asynchronous operations is a decision made by the Object implementer. Often an object will have synchronous and asynchronous versions of the same method.

This threading model means that the script writer does not have to deal with mutexes and other synchronization objects. It is always single threaded and the message port is polled or waited upon to receive events into the thread. However, BrightScript Component implementers have to think about threading issues. For example, roList and roMessagePort are thread safe internally so that they can be used by multiple threads.

4.9 Scope

BrightScript uses the following scoping rules:

- BrightScript does not support global variables. Except, there is one hard-coded global variable "global" that is an interface to the global BrightScript Component. The global component contains all global library functions.
- Functions declared with the FUNCTION statement are at global scope, unless they are anonymous, in which case they are local scope.
- Local variables exist with function Scope. If a function calls another function, that new function has its own scope.
- Labels exist in function scope.
- Block Statements (like FOR-END FOR or WHILE-END WHILE) do not create a separate scope

4.10 Creating and Using Intrinsic Objects

In most of this manual we use the term "object" to refer to a "BrightScript Component". These are C or C++ components with interfaces and member functions that BrightScript uses directly. Other than a few core objects that BrightScript relies upon (roArray, roAssociativeArray, roInt, etc.) BrightScript Components are platform specific.

You can create "intrinsic" objects in BrightScript itself to use in your scripts. However, to be clear, these are not BrightScript Components. There is currently no way to create a BrightScript Component in BrightScript, or to create intrinsic objects that have interfaces (they only contain member functions, properties, or other objects).

A BrightScript object is built upon an Associative Array. In fact, it is an roAssociativeArray. When a member function is called "from" an AssociativeArray, a "this" pointer is set. The "this" pointer is "m". "m" is accessible inside the function code to access object keys.

.

A "constructor" in BrightScript is a normal function at global scope that creates the AssociativeArray and fills in its member functions and properties. There is nothing "special" about it.

For an example, see the "snake" game at the end of this manual.

4.11 "m" The BrightScript "this pointer"

A BrightScript object is built upon an Associative Array. In fact, it is an roAssociativeArray. When a member function is called "from" an AssociativeArray, a "this" pointer is set. The "this" pointer is "m". "m" is accessible inside the function code to access object keys.

```
Function Main()
      o=ConstructMyObject()
      o.Set("hi!")
      print o.Get()
      print "--------"
      print o
      stop
End Function

Function ConstructMyObject()

      o={
            Set         : function(x):m.Value=x:end function
            Get         : function():return m.Value:end function
            Value : 0
      }

      return o

End Function
```

Output:

```
hi!
--------
value: hi!
get: <bsTypedValue: Function>
set: <bsTypedValue: Function>
```

4.12 Script Libraries

In addition to the platform BrightScript components discussed in Section 4.1, BrightScript 3.0 enables Script Libraries to be used in your scripts. The BrightScript libraries are .brs files, that are provided by the BrightScript platform or you can provide. They can be included in your source files with the new "Library" keyword. Libraries are searched for in pkg:/ folderand then the system library path. On the Roku Streaming Player, the system library path is "common:/LibCore".

```
Example
      Library "v30/bslCore.brs"
```

Those interested in the contents of the library files, can view the source on their debug console:

```
BrightScript> bslCore = ReadAsciiFile("common:/LibCore/1.0/bslCore.brs")
BrightScript> print bslCore
```

.

4.12.1 v30/bslCore.brs

This "system" library includes constant name value pairs in associative arrays.

`Object bslBrightScriptErrorCodes()`
Returns an roAssociativeArray with name value pairs of the error name and value.`Object bslGeneralConstants()`

- Returns an roAssociativeArray with name value pairs of the system constants

`Object bslUniversalControlEventCodes()`

- Returns an roAssociativeArray with name value pairs of the remote key code (buttons) constants

`String AsciiToHex(String ascii)`

- Returns a String that is the Hex encode of the passed in ascii string

`String HexToAscii(String hex)`

- Returns a String that is the hex decode ascii text of the hex input string

`Integer HexToInteger(String hex)`

- Returns the integer value of the passed in hex string.

4.12.2 v30/bslDefender.brs

This library includes 2D graphics helper functions on top of the native components.
`Object dfNewBitmapSet(String filename)`
The goal is to enable simple xml descriptions of graphics resources like bitmaps, regions, sprites, animations, and layouts to be used in your games. The library handles parsing, loading, rendering, and animation of sprites sheets (multiple images in a single png file).

- filename is the path to an XML file that contains info about bitmap regions, animation frames, and ExtraInfo metadata (any fields you would like) about resources used in 2d games.
- Returns an roAssociativeArrary with the following name value pairs:
  - ExtraInfo :roAssociativeArray
    - Contains any name value pairs the designer desires.
  - Regions : roAssociativeArray of name, roRegions pairs
    - The roAssociative array returns a set of Regions from Region tag information from the xml.
    - Used to describe regions of a bitmap
    - Bitmap contains parameters: name, filename where filename is a .png, .gif, or .jpg file
    - Region contains parameters: name, x, y, w, h and is a subtag of a bitmap
  - Animations : roAssociativeArray of name, roArray of roRegion pairs.
    - The top level roAssociative array is the animation name with a value of an array of frames that represent the animation
    - Each frame is an roRegion that is represented in the xml by the "use" parameter. The use parameter refers to a previously defined Bitmap.Region returned in the Regions info.
    - Animation frame descriptions can be used by the roSprite and roCompositor components.
  - Backgrounds : roAssociativeArray of name, path pairs.
    - Backgrounds do not create bitmaps at this point
    - Use in conjunction with dfSetBackground() to manage backgrounds

`Void dfDrawMessage(Object dest, Object region)`

- dest is an roScreen/roBitmap/roRegion and region is an roRegion
  greys the entire dest region and draws it the region centered on the drawable dest.

`Boolean dfDrawImage(Object dest, String path, Integer x, Integer y)`

- Returns True if successful

.

- Creates a bitmap out of the image stored in the filename "path" and draws it at position (x,y) of the drawable dest.

```
Object dfSetupDisplayRegions(Object screen, integer topx, integer topy,
integer width, integer height)
```

- Helper function to setup screen scaling with supplied pillar box or letterbox images to fill the entire screen.
- screen is an roScreen
- topx and topy are the coordinates of the upper left hand corner of the main drawing region
- Width and height is the size of the main drawing region
- Returns an associative array containing the following roRegions
  Main: main drawable region
  Left: left region if there is pillar box area on the left
  Right: right region if there is a pillar box area on the right
  Upper: upper region if there is a letterbox area at thetop
  Lower: lower region if there is a letterbox area at the bottom

When using these regions as drawables, your graphics will be translated and clipped to these regions.

```
Object dfSetBackground(String backgroundName, Object backgrounds)
```

- dfSetBackground helps manage the limited video memory. The video memory does not currently run a defragmenter, and is very limited. These constraints make it important that large bitmaps (like backgrounds that fill the entire screen) are only allocated when needed. It is also helpful if you set your first initial background very early in your program, and then immediately replace the background after it is no longer in use. This helper function supports this background management for your application.
- backgroundName is a key for the Backgrounds roAssociative array of backgrounds.
- Backgrounds is an roAssociative array of background name keys and file path string values
- This function creates an roBitmap out of the background image file and returns a region the size of the entire roBitmap.

Example: bslDefender.brs

If spriteMap.xml contains the following:

```
<DefenderBitmapSet>
   <ExtraInfo cellsize="40"/>
   <Bitmap name="Background" filespec="pkg:/images/background.png" />
   <Bitmap name="game-over" filespec="pkg:/images/gameover.png" />
   <Bitmap name="title-screen" filespec="pkg:/images/Splash.gif" />
   <Bitmap name="water_strip" filespec="pkg:/images/water_sprite.png">
      <Region name="a"    x="0" y="" w="40" h="40" t="225" />
      <Region name="b"    x="40" y="0" w="40" h="40" t="225" />
   </Bitmap>
   <Animation name="water">
      <frame use="water_strip.a" />
      <frame use="water_strip.b" />
   </Animation>
</DefenderBitmapSet>

Then

BrightScript> xml = ReadAsciiFile("pkg:/images/map.xml")
BrightScript> bitmapset = dfNewBitmapSet(xml)
BrightScript> cellwidth=app.bitmapset.extrainfo.cellsize.toint()
```

.

```
BrightScript> print bitmapset.regions
Background: <Component: roRegion>
game-over: <Component: roRegion>
title-screen: <Component: roRegion>
water_strip.a: <Component: roRegion>
water_strip.b: <Component: roRegion>
water_strip: <Component: roRegion>
BrightScript> print bitmapset.animations.water
<Component: roRegion>
<Component: roRegion>
BrightScript> dfDrawMessage(screen, bitmapset.regions["game-over"])
BrightScript> REM screen now shows gameover.png image centered on screen
```

5.0 Program Statements

5.1 DIM name (dim1, dim2, …, dimK)

DIM ("dimension") is a statement that provides a short cut to creating roArray objects. It sets variable *name* to type "roArray", and creates Array's of Array's as needed for multi-dimensional arrays. The dimension passed to Dim is the index of the maximum entry to be allocated (the array initial size = dimension+1); the array will be resized larger automatically if needed.

```
Dim array[5]
```

Is the same as:

```
array=CreateObject("roArray",6,true)
```

Note that x[a,b] is the same as x[a][b]

Another example:

```
Dim c[5, 4, 6]

For x = 1 To 5
   For y = 1 To 4
        For z = 1 To 6
           c[x, y, z] = k
           k = k + 1
       End for
   End for
End for

k=0
For x = 1 To 5
    For y = 1 To 4
       For z = 1 To 6
          If c[x, y, z] <> k Then print"error" : Stop
          if c[x][y][z] <> k then print "error":stop
          k = k + 1
          End for
   End for
End for
```

.

5.2 variable = expression

Assigns a variable to a new value.

Examples:

```
a$="a rose is a rose"
b1=1.23
x=x-z1
```

In each case, the variable on the left side of the equals sign is assigned the value of the constant or expression on the right side.

5.3 END

Terminates execution normally.

5.4 STOP

Interrupts execution return a STOP error. Invokes the debugger. Use "cont" at the debug prompt to continue execution, or "step" to single step.

5.5 GOTO label

Transfers program control to the specified line number. GOTO *label* results in an branch. A label is an identifier terminated with a colon, on a line by itself. Example:

```
mylabel:
print "Anthony was here!"
goto mylabel
```

5.6 RETURN expression

Used to return from a function back to the caller. If the function is not of type Void, return can return a value to the caller.

5.7 FOR counter = exp TO exp STEP exp END FOR

Creates an iterative (repetitive) loop so that a sequence of program statements may be executed over and over a specified number of times. The general form is (brackets indicate optional material):

FOR *counter-variable = initial value* TO *final value* [STEP *increment*]
[*program statements*]
END FOR

In the FOR statement, *initial value, final value* and *increment* can be any expression. The first time the FOR statement is executed, these three are evaluated and the values are saved; if the variables are changed by the loop, it will have no effect on the loop's operation. However, the counter variable must not be changed or the loop will not operate normally. The first time the FOR statement is executed the counter is set to the "initial value" and to the type of "initial value".

.

At the top of the loop, the counter is compared with the *final value* specified in the FOR statement. If the counter is greater than the *final value*, the loop is completed and execution continues with the statement following the END FOR statement. (If *increment* was a negative number, loop ends when counter is less than *final value*.) If the counter has not yet exceeded the *final value*, control passes to the first statement after the FOR statement.

When program flow reaches the END FOR statement, the counter is incremented by the amount specified in the STEP *increment*. (If the *increment* has a negative value, then the counter is actually decremented.) If STEP *increment* is not used, an increment of 1 is assumed.

.Example:

```
for i=10 to 1 step -1
   print i
end for
```

"EXIT FOR" is used to exit a FOR block prematurely.

5.8 FOR EACH item IN object

The FOR EACH statement iterates through each item in any object that has an "ifEnum" interface (enumerator). The For block is terminated with a END FOR statement. The variable *item* is set at the top of the loop to the next item in the object. Objects that are intrinsically ordered (like a List) are enumerated in order. Objects that have no intrinsic order (like AssociativeArray) are enumerated in apparent random order. It is okay to delete entries as you enumerate them.

"EXIT FOR" is used to exit a FOR block prematurely.

The following example objects can be enumerated: roList, roArray, roAsscoiativeArray, roMessagePort.

Example:

```
aa={joe: 10, fred: 11, sue:9}
For each n in aa
   Print n;aa[n]
   aa.delete[n]
end for
```

5.9 While expression / Exit While

The While loop executes until expression is false. The "exit while" statement can be used to terminate a while loop prematurely.

Example:

```
k=0
while k<>0
   k=1
   Print "loop once".
end while


while true
   Print "loop once"
   Exit while
End while
```

.

5.10 REM

Instructs the compiler to ignore the rest of the program line. This allows you to insert comments (REMarks) into your program for documentation. An ' (apostrophe) may be used instead of REM.

Examples Program:

```
rem ** this remark introduces the program **
'this too is a remark
```

5.11 IF expression THEN statements [ELSE statements]

There are two forms of the IF THEN ELSE statement. The single line form (this one), and the multi-line or block form (see next section). The IF instructs the Interpreter to test the following *expression*. If the *expression* is true, control will proceed to the statements immediately following the expression. If the expression is False, control will jump to the matching ELSE statement (if there is one) or down to the next program line.

Examples:

```
      if x>127 then print "out of range" : end
      if caveman="fred" then print "flintsone" else print "rubble"
```

NOTE: THEN is optional in the above and similar statements. However, THEN is sometimes required to eliminate an ambiguity. For example:

```
if y=m then m=o 'won't work without THEN.
```

5.12 BLOCK IF, ELSEIF, THEN, ENDIF

The multi-line or block form of IF THEN ELSE is more flexible. It has the form:

```
  If BooleanExpression [ Then ]
  [ Block ]
  [ ElseIfStatement+ ]
  [ ElseStatement ]
  End If

ElseIfStatement ::=
  ElseIf BooleanExpression [ Then ]
  [ Block ]

ElseStatement ::=
  Else
  [ Block ]
```

For example:

```
    vp_msg_loop:
        msg=wait(tiut, p)
        if type(msg)="rovideoevent" then

            if debug then print "video event";msg.getint()
            if lm=0 and msg.getint() = meden then
                if debug then print "videofinished"
```

.

```
            retcode=5
            return
        endif
    else if type(msg)="rogpiobutton" then
        if debug then print "button press";msg
        if esc0 and msg=b0 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)="Invalid" then
        if debug then print "timeout"
        retcode=6
        return
    endif

    goto vp_msg_loop
```

5.13 PRINT [#output_object], [@location], item list

Prints an item or a list of items on the console, or if *output_object* is specified, to an object that has an "`ifStreamSend" interface. . The items may be either strings, number, variables, or expressions. Objects that have an ifInt, ifFloat, or ifString interface may also be printed.

The items to be PRINTed may be separated by commas or semi-colons. If commas are used, the cursor automatically advances to the next print zone before printing the next item. If semi-colons are used, no space is inserted between the items printed.

Positive numbers are printed with a leading blank (instead of a plus sign); all numbers are printed with a trailing blank; and no blanks are inserted before or after strings.

Examples:

```
x=5:print 25; "is equal to"; x ^2
25 is equal to 25

a$="string"
print a$;a$,a$;" ";a$
stringstring string string

print "zone 1","zone 2","zone 3","zone 4"
zone 1          zone 2          zone 3          zone 4
```

Each print zone is 16 char wide. The cursor moves to the next print zone each time a comma is encountered.

```
print "print statement #1 ";
print "print statement #2"
```

Output:

```
print statement #1 print statement #2
```

Semi-colon's can be dropped in some cases. For example, this is legal:

```
Print "this is a five "5"!!"
```

.

A trailing semi-colon over-rides the cursor-return so that the next PRINT begins where the last one left off. If no trailing punctuation is used with `print`, the cursor drops down to the beginning of the next line.

If the console you are printing to has the interface "ifTextField" then @ specifies exactly where printing is to begin. For example:

```
print #m.text_field,@width*(height/2-1)+(width-len(msg$))/2,msg$;
```

Whenever you `print @` on the bottom line of the Display, there is an automatic line-feed, causing everything displayed to move up one line. To suppress this, use a trailing semi-colon at the end of the statement.

TAB (expression)
Moves the cursor to the specified position on the current line (modulo the width of your console if you specify TAB positions greater than the console width). TAB may be used several times in a PRINT list.

Example:
```
print tab(5)"tabbed 5";tab(25)"tabbed 25"
```

No punctuation is required after a TAB modifier. Numerical expressions may be used to specify a TAB position. TAB cannot be used to move the cursor to the left. If the cursor is beyond the specified position, the TAB is ignored.

POS(x)
Returns a number from 0 to window width, indicating the current cursor position on the cursor. Requires a "dummy argument" (any numeric expression).

```
print tab(40) pos(0)   'prints 40 at position 40

print "these" tab(pos(0)+5)"words" tab(pos(0)+5)"are";
print tab(pos(0)+5)"evenly" tab(pos(0)+5)"spaced"
```

5.14 Function([parameter As Type, …]) As Type / End Function

A function is declared using the function statement. In parentheses, one or more optional parameters to be passed may be declared. The return type of the function may also be declared. If the parameter or return type are not declared, they are assumed to be "dynamic"

Intrinsic types are passed by value (a copy is made). Objects are passed by reference.

Parameters can be of type:

- Integer
- Float
- Double
- Boolean
- String
- Object
- Dynamic

In addition to the above types, the return type can be:

- Void

.

Parameters can have default values and expressions.

For example:

```
Function cat(a, b)
        Return a+b   'a, b could be numbers or strings
End Function

Function five() As Integer
    Return 5
End function


Function add(a As Integer, b As Integer) As Integer
   Return a+b
End function

Function add2(a As Integer, b=5 as Integer) As Integer
      Return a+b
End Function

Function add3(a As Integer, b=a+5 as Integer) As Integer
      Return a+b
End Function
```

Functions have their own scope.

The statement "Sub" can be used instead of "function" as a shortcut to a function of Void return Type.

If a function is called from an associative array, then a local variable "m" is set to the AssociatiaveArray that the function is stored in.

For example:

```
sub main()
   obj={
       add: add
       a: 5
       b: 10
       }

   obj.add()
   print obj.result
end sub

function add() As void
    m.result=m.a+m.b
end function
```

If a function is not called from an AssociatiaveArray, then its "m" is set to an AssocicateArray that is global to the module, and persists across calls.

5.15 Anonymous Functions

A function is anonymous if it does not have a name. It can be declared like this;

.

```
myfunc=function (a, b)
   Return a+b
end function

print myfunc(1,2)
```

They can be used with AA literals like this:

```
q = {

starring : function(o, e)
      str = e.GetBody()
      print "Starring: " + str
      toks = box(str).tokenize(",")
      for each act in tok
            actx = box(act).trim()
            if actx <> "" then
                  print "Actor: [" + actx + "]"
                o.Actors.Push(actx)
            endif
      end for
      return 0
      end function
}

q.starring(myobj, myxml)
```

.

6.0 Built-In Functions

BrightScript has a small number of built-in, module scope, intrinsic functions. They are the following.

6.1 CreateObject(classname As String, [optional parameters]) As Object

Creates a BrightScript Component of class *classname* specified. Return invalid if the object creation fails. Some Objects have optional parameters in their constructor that are passed after *name*. Example:

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

6.2 Type(variable, [optional version]) As String

Returns the type of a variable and/or object. See the BrightScript Component specification for a list of types. For example:

```
Print type(5)                ‘returns a 2.1 compatible type
Print type("my string", 3)   ‘return a BrightScript 3 type
```

6.3 GetGlobalAA() As Object

Each script has a global Associative Array. It can be fetched with this function.
New in BrightScript 3.

6.4 Box(x as Dynamic) as Object

Box() will return an object version of an intrinsic type, or pass through an object if given one.

```
bo = box("string")
bo = box(bo)  ' no change to bo
```

6.5 Run(filename As String, Optional Args…) As dynamic

Run(filenamearray As Object, Optional Args…) As dynamic

The run command will run a script from a script. Args may be passed to the scripts Main() function, and the called script may return arguments.

If a filename string is passed, that file compiled and run.

If an array of files are passed instead of a single filename, then each file is compiled and they are linked together, than run.

Example:

```
Sub Main()
      Run("pkg:/test.brs")
      BreakIfRunError(LINE_NUM)
      Print Run("test2.brs", "arg 1", "arg 2")
```

.

```
      if Run(["pkg:/file1.brs","pkg:/file2.brs"])<>4 then stop
      BreakIfRunError(LINE_NUM)
      stop
End Sub

Sub BreakIfRunError(ln)
   el=GetLastRunCompileError()
   if el=invalid then
      el=GetLastRunRuntimeError()
      if el=&hFC or el=&hE2 then return
      'FC==ERR_NORMAL_END, E2=ERR_VALUE_RETURN
      print "Runtime Error (line ";ln;"): ";el
      stop
   else
      print "compile error (line ";ln;")"
      for each e in el
         for each i in e
            print i;": ";e[i]
         end for
      end for

      stop
   end if

End Sub
```

6.6 Eval(codesnipit) As Integer

Eval can be used to run a code snippet in the context of the current function. It performs a compile, and then byte code execution. The runtime error is returned – these are the same error codes are returned by GetLastRunTuntimeError().

Eval() can be usefully in two cases. The first is when you need to dynamically generate code at runtime. The other is if you need to execute a statement that could result in a runtime error, but you don't want code execution to stop.

```
Print Eval("1/0")   ' divide by zero error
```

6.7 GetLastRunCompileError() As Object

Returns an roList of compile errors, or invalid if no errors. Each list entry is an roAssociativeArray with the keys: ERRNO, ERRLN, ERRSTR.

6.8 GetLastRunRuntimeError() As Integer

Returns an error code result after the last script Run().

These two are normal:

```
  &hFC==ERR_NORMAL_END
  &hE2==ERR_VALUE_RETURN
```

Here is a code fragment to assign variables to common runtme errors:

```
ERR_USE_OF_UNINIT_VAR = &hE9
ERR_DIV_ZERO = &h14
```

.

```
ERR_TM = &h18
ERR_USE_OF_UNINIT_VAR = &hE9
ERR_RO2 = &hF4
ERR_RO4 = &hEC
ERR_SYNTAX = 2
ERR_WRONG_NUM_PARAM = &hF1
```

.

7.0 Global Utility Functions

BrightScript has a set of standard, module scope, functions. These functions are stored in the global object. If the compiler sees a reference to one of the global functions, it directs the runtime to call the appropriate global object member.

7.1 Sleep(milliseconds As Integer) As Void

This function causes the script to pause for the specified time, without wasting CPU cycles. There are 1000 milliseconds in one second.
Example:

```
sleep(1000)   ' sleep for 1 second
sleep(200)    ' sleep 2/10 of a second
sleep(3000)  ' sleep three seconds
```

7.2 Wait (timeout As Integer, port As Object) As Object

This function waits on objects that are "waitable" (those that have a MessagePort interface). Wait() returns the event object that was posted to the message port. If timeout is zero, "wait" will wait for ever. Otherwise, Wait will return after timeout milliseconds if no messages are received. In this case, Wait returns a type "invalid".

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
```

7.3 GetInterface(object As Object, ifname As String) As Interface

Each BrightScript Component has one or more interfaces. This function returns a value of type "Interface".

Note that generally BrightScript Components allow you to skip the interface specification. In which case, the appropriate interface within the object is used. This works as long as the function names within the interfaces are unique.

7.4 UpTime(dummy As Integer) As Float

Returns the uptime of the system since the last reboot in seconds.

7.5 RebootSystem() As Void

Causes a soft reboot. Works only on the BrightSign platform. The Roku platform has disabled this feature.

7.6 ListDir(path As String) As Object

Returns a List object containing the contents of the directory path specified. All files names are converted to all lowercase For example:

```
BrightScript> l=ListDir("/")
```

.

```
BrightScript> print l
test_movie_3.vob
test_movie_4.vob
test_movie_1.vob
test_movie_2.vob
```

7.7 ReadAsciiFile(filepath As String) As String

This function reads the specified file and returns it as a string. For example:

```
text=ReadAsciiFile("tmp:/config.txt")
```

7.8 WriteAsciiFile(filepath As String, buffer As String) As Boolean

This function reads the specified file and returns it as a string. For example:

```
WriteAsciiFile("tmp:/config.txt", "the text to write")
```

7.9 CopyFile(source As String, destination As String) As Bool

Make a copy of a file.

7.10 MatchFiles(path As String, pattern_in As String) As Object

Search a directory for filenames that match a certain pattern. *Pattern* is a wildmat expression. Returns a List object.

This function checks all the files in the directory specified against the pattern specified and places any matches in the returned roList. The returned list contains only the part of the filename that is matched against the pattern not the full path.

The pattern may contain certain special characters:

A '?' matches any single character.

A '*' matches zero or more arbitrary characters.

The character class '[...]' 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 close bracket. Within the 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 can be negated by specifying '^' as the first character. To match a literal '^' place it elsewhere within the class.

The characters '?', '*' and '[' lose their special meaning if preceded by a single '\'. A single '\' can be matched as '\\'.

Example:

```
l=MatchFiles(".", "*.mpg")
```

.

7.11 DeleteFile(file As String) As Boolean

Delete the specified file.

7.12 DeleteDirectory(dir As String) As Boolean

It is only possible to delete an empty directory.

7.13 CreateDirectory(dir As String) As Boolean

Creates the specified Directory. Only one directory can be created at a time

7.14 FormatDrive(drive As String , fs_type As String) As Boolean

Formats a specified drive using the specified filesystem.

7.15 strtoi(str As String) As Dynamic

return invalid if str is not a string, else the integer value as converted by strtoi.

7.16 RunGarbageCollector() As Object

New in BrightScript 3.0

This function runs the garbage collector. It returns and Associative Array with some statistics regarding the garbage collection.

See the "Garbage Collection" section of this manual for more detail. You don't normally need to call this function.

Example:

```
BrightScript Debugger> a=[]
BrightScript Debugger> a[0]=a
BrightScript Debugger> a=invalid
BrightScript Debugger> print RunGarbageCollector()
COUNT:  3
ORPHANED:  1
ROOT:  2
```

.

8.0 Global String Functions

8.1 UCase(s As String) As String

Converts the string to all upper case.

8.2 LCase(s As String) As String

Converts the string to all lower case.

8.3 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 to occur. Example:

```
print asc("a")
```

8.4 Chr (ch As Integer) As String

Performs the inverse of the ASC function: returns a one-character string whose character has the specified ASCII, or control. Example:

```
print chr(35)  'prints a number-sign #
```

Using CHR, you can assign quote-marks (normally used as string-delimiters) to strings. The ASCII code for quotes - is 34. So A$=CHR(34) assigns the value " to A$.

8.5 Instr(position to start As Integer, text-to-search As String, substring-to-find As String) As Integer

Returns the position of a substring within a string. Returns 0 if the substring is not found. The first position is 1. For example:

```
print instr(1, "this is a test", "is")
```

will print 3

8.6 Left (s As String, n As Integer) As String

Returns the first *n* characters of *s.*

```
print left("timothy", 3)    ' displays tim
```

8.7 Len (s As String) As Integer

Returns the character length of the specified string. Example:

```
print len("timothy")   ' prints 7
```

8.8 Mid (s As String, p As Integer, [n As Integer]) As String

Returns a substring of *s* with length *n* and starting at position *p*. n may be omitted, in which case the string starting at p and ending at the end of the string is returned. The first character in the string is position 1. Also see the ifStringOps interface in roString. Example:

.

```
print mid("timothy", 4,3)     'prints oth
```

8.9 Right (s As String, n As Integer) As String

Returns the last n characters of *string*. Example*:*

```
right$(st$,4) returns the last 4 characters of st$.
```

8.10 Str (value As Float) As String
Stri(value as Integer) As String

Converts a *value* to a string. STR(A), for example, returns a string equal to the character representation of the value of A. For example, if A=58.5, then STR(A) equals the string " 58.5". (Note that a leading blank is inserted before "58.5" to allow for the sign of A). Also see the ifStringOps interface in roString.

8.11 String (n As Integer, character As String) As String
Stringi (n As Integer, character As Integer) As String

Returns a string composed of n *character*-symbols. For example,

```
string(30,"*")
 returns "******************************"
```

8.12 Val (s As String) As Float

Performs the inverse of the STR function: returns the number represented by the characters in a string argument. For example, if A$="12" and B$="34" then VAL(A$+ "."+B$) returns the number 12.34.

.

9.0 Global Math Functions

The following math functions are part of global. Trig functions use or return radians, not degrees.

9.1 Abs (x As Float) As Float

Returns the absolute value of the argument.

9.2 Atn (x As Float) As Float

Returns the arctangent (in radians) of the argument; that is, ATN(X) returns "the angle whose tangent is X". To get arctangent in degrees, multiply ATN(X) by 57.29578.

9.3 Cos (x As Float) As Float

Returns the cosine of the argument (argument must be in radians). To obtain the cosine of X when X is in degrees, use CGS(X*.01745329).

9.4 Csng (x As Integer) As Float

Returns a single-precision float representation of the argument.

9.5 Cdbl(x As Integer) As Float

Also returns a single precision float representation of the argument. Someday may return double.

9.6 Exp (x As Float) As Float

Returns the "natural exponential" of X, that is, *ex.* This is the inverse of the LOG function, so X=EXP(LOG(X)).

9.7 Fix (x as Float) As Integer

Returns a truncated representation of the argument. All digits to the right of the decimal point are simply chopped off, so the resultant value is an integer. For non-negative X, FIX(X)=lNT(X). For negative values of X, FIX(X)=INT(X)+1. For example, FIX(2.2) returns 2, and FIX(-2.2) returns -2.

9.8 Int(x As Float) As Integer

Returns an integer representation of the argument, using the largest whole number that is not greater than the argument.. INT(2.5) returns 2; INT(-2.5) returns -3; and INT(1000101.23) returns 10000101.

9.9 Log(x As Float) As Float

Returns the natural logarithm of the argument, that is, *$\log_e$(argument). This*

is the inverse of the EXP function, so X=LOG(EXP (X)). To find the logarithm of a number to another base b, use the formula $\log_b(X) = \log_e(X)/\log_e(b)$. For example, LOG(32767)/LOG(2) returns the logarithm to base 2 of 32767.

9.10 Rnd(0) As Float
Rnd(range As Integer) As Integer

Generates a pseudo-random number using the current pseudo-random "seed number" (generated internally and not accessible to user). RND may be used to produce random numbers between 0 and 1, or random integers greater than 0, depending on the argument.

.

RND(0) returns a float value between 0 and 1.
RND(integer) returns an integer between 1 and *integer* inclusive . For example, RND(55) returns a pseudo-random integer greater than zero and less than 56.

9.11 Sgn(x As Float) As Integer
Sgn(x As Integer) As Integer

The "sign" function: returns -1 for X negative, 0 for X zero, and +l for X positive.

9.12 Sin(x As Float) As Float

Returns the sine of the argument (argument must be in radians). To obtain the sine of X when X is in degrees, use SIN(X*.01745329).

9.13 Sqr(x As Float) As Float

Returns the square root of the argument. SQR(X) is the same as X^(1/2), only faster.

9.14 Tan(x As Float) As Float

Returns the tangent of the argument (argument must be in radians). To obtain the tangent of X when X is in degrees, use TAN(X*.01745329).

.

10.0 Core BrightScript Components and Interfaces

The following set of core BrightScript Components are used extensively by BrightScript, and are therefore incorporated into the language definition.

- roArray (interfaces: ifArray, ifEnum)
- roAssociativeArray (interfaces: ifAssociativeArray, ifEnum)
- roList (interfaces: ifList, ifArray, ifEnum)
- roByteArray (interfaces: ifArray, ifEnum, ifByteArray)
- roMessagePort (interfaces: ifMessagePort, ifEnum)
- roGlobal (interfaces: ifGlobal; all static member functions)
- roInt (interfaces: ifInt, ifIntOps)
- roFloat (interfaces: ifFloat)
- roString (interfaces: ifString, if StringOps)
- roFunction (interfaces: ifFunction)
- roBoolean (interface: ifBoolean)
- roInvalid (no interfaces)
- roXMLElement (interfaces: ifXMLElement)
- roXMLList (interfaces: ifList, ifXMLList)

10.1 ifList

`ResetIndex() As Boolean`
- Reset current index or position in list to the head element.

`AddTail(tval As Dynamic) As Void`
- Add typed value to tail of list.

`AddHead(tval As Dynamic) As Void`
- Add typed value to head of list.

`RemoveIndex() As Dynamic`
- Remove entry at current index or position from list and increment index or position in list.
- Return invalid when end of list reached

`GetIndex() As Dynamic`
- Get entry at current index or position from list and increment index or position in list.
- Return invalid when end of list reached

`RemoveTail() As Dynamic`
- Remove entry at tail of list.

`RemoveHead() As Dynamic`
- Remove entry at head of list.

`GetTail() As Dynamic`
- Get Object at tail of List and keep Object in list.

`GetHead() As Dynamic`
- Get entry at head of list and keep entry in list.

`Count() As Integer`
- Return the number of elements in list.

`Clear() As Void`
- Remove all elements from list.

.

10.2 ifEnum

```
Reset() As Void
```

- Reset position to first element of enumeration

```
Next() As Dynamic
```

- Return typed value at current position and increment position

```
IsNext() As Boolean
```

- Return true if there is a next element

```
IsEmpty() As Boolean
```

- Return true is there is not a next element

10.3 roList

The list object implements the interfaces: ifList, ifArray, ifEnum and therefore can behave like an array that can dynamically add members. The array operator [ ] can be used to access any element in the ordered list.

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]
```

Output:

```
a
b
c
d
c
```

10.4 ifMessagePort

```
GetMessage () As Object
WaitMessage(timeout As Integer) As Object
```

10.5 roInt, roFloat, roString, roBoolean, roFunction, roInvalid

The intrinsic types Integer ("Integer"), Float ("Float"), function ("Function"), Boolean ("Boolean"), Invalid ("Invalid") and String "String" have an object and interface equivalents, with the following interfaces:

10.6 ifInt

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

.

10.7 ifIntOps

```
ToStr() As String
```

Note: This function does not add a leading space to positive integers, as the global str() function does.

10.8 ifFloat

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

10.9 ifString

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

10.10 ifStringOps

```
SetString(s As String, strlen As Integer) As Void
AppendString(s As String, strlen As Integeter) As void
Len() As Integer
GetEntityEncode() As String
Tokenize(delim as String) As Object
Trim() As String
ToInt() As Int
ToFloat() As Float
Left(num_chars) As String
Right(num_chars) As String
Mid(start_index) As String
Mid(start_index, num_chars) As String
Instr(substring) As Integer) As Integer
Instr(start_index, substring) As Integer
```

Note: ifStringOps function indexes start at zero, not one as the global functions do.

Also Note: the function appendstring() modifies itself, which can result in unexpected results. For example:

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

This code does not appear to do an append, but it is working as designed since the append happens to the temporary boxed object. x="string" sets x to the intrinsic type, vs. x=box("string", which work as you might expect.

10.11 ifFunction

```
GetSub() As String
SetSub(value As Function) As Void
```

.

10.12 ifBoolean

```
GetBolean() As Boolean
SetBoolean(value As Boolean) As Void
```

. These are useful in the following situations:

- When an object is needed, instead of an intrinsic value. For example, "roList" maintains a list of objects. If an Integer is added to roList, for example, it will be automatically wrapped in an roInt by the language interpreter. When a function that expects a BrightScript Component as a parameter is passed an int, float, function, or string, BrightScript automatically creates the equivalent BrightScript Component.

- If any object exposes the ifInt, ifFloat, ifBoolean or ifString interfaces, that object can be used in any expression that expects an intrinsic value. For example, in this way an roTouchEvent can be used as an integer whose value is the userid of the roTouchEvent.

Notes:

- If o is an roInt, then the following statements have the following effects
    1. print o ' prints o.GetInt()
    2. i%=o ' assigns the integer i% the value of o.GetInt()
    3. k=o 'presumably k is dynamic typed, so it becomes another reference to the roInt o
    4. o=5 'this is NOT the same as o.SetInt(5). Instead it releases o, and 'changes the type of o to Integer (o is dynamically typed). And assigns it to 5.

Example:

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

Example:

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

Note that an integer value of "5" is converted to type "roInt" automatically,
because list.AddTail() expects an BrightScript Component as its parameter.

10.13 roAssociativeArray

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

This object is created with no parameters:

- `CreateObject("roAssociativeArray")`

.

The ifAssociativeArray interface provides:

`AddReplace(key As String, tval As Dynamic) As Void`
- Add a new entry to the array associating the supplied value with the supplied key string. Only one value may be associated with a key so any existing value is discarded.

`Lookup(key As String) As Dynamic`
- Look for an entry in the array associated with the specified key. If there is no value associated with the key then type "invalid" is returned.

`DoesExist(key As String) As Boolean`
- Look for an entry in the array associated with the specified key. 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`
- Look for an entry in the array associated with the specified key. If there is such an value then it is deleted and true is returned. If not then false is returned.

`Clear() As Void`
- Remove all key/values from the associative array.

`SetModeCaseSensitive() As void`
- Associative Array lookups are case insensitive by default. This call makes all subsequent actions case sensitive.

`LookupCI(key As String) As Dynamic`
- Same as "Lookup" except key comparison is always case insensitive, regardless of case mode.

`Append(aa As Object)`
- Append an AssociativeArray to this one.

Example:

```
aa = CreateObject("roAssociativeArray")

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

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

Produces:

 42
Sign
```

10.14 roArray

An array stores objects in a continuous array of memory location. Since an roArray contains BrightScript Components, and there are object wrappers for most intrinsic data types, each entry of an array can be a different type (or all of the same type).

The roArray class implements the ifArray and ifEnum interfaces.

This object is created with two parameters:

- `CreateObject("roArray", size As Integer, resize As Boolean)`
  - *Size* is the initial number of elements allocated for the array. If *resize* is true, the array will be resized larger if needed to accommodate more elements. If the array is large, this process might be slow.

.

- The "dim" statement may be used instead of CreateObject to create a new array. Dim has the advantage in that it automatically creates arrays of arrays for multi-dimensional arrays.

The ifArray interface provides:

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

`SetEntry(index As Integer, tvalue As Dynamic) As Void`
- Sets an entry of a given index to the passed typed value

`Peek() As Dynamic`
- Returns the last (highest index) array entry without removing it.

`Pop() As Dynamic`
- Returns the last (highest index) array entry and removes it from the array.

`Push(tvalue As Dynamic) As Void`
- Adds a new highest index entry into an array (adds to the end of the array)

`Shift() As Dynamic`
- Removes index zero from the array and shifts every other entry down one. This is like a "pop" from the bottom of the array instead of the top.

`Unshift(tvalue As Dynamic) As Integer`
- Adds a new index zero to the array and shifts ever other index up one to accomidate. This is like a Push to the bottom of the array.

`Delete(index as Integer) As Boolean`
- Deletes the indicated array entry, and shifts down all entries above to fill the hole. The array length is decreased by one.

`Count() As Integer`
- Returns the index of highest entry in the array+1 (the length of the array).

`Clear() As Void`
- Deletes every entry in the array.

`Append(As Object) As Void`
- Appends one roArray to another. If the passed Array contains "holes" (entires that were never set to a value), they are not appended.

TODO: Doc ifArraySet, ifArrayGet.

10.15 roByteArray

The byte array component is used to contain and manipulate an arbitrary array of bytes. It contains functions to convert strings to or from a byte array, as well as to or from ascii hex or ascii base 64. Note that if you are converting a byte array to a string, and the byte array contains a zero, the string conversion will end at that point. roByteArray will autosize to become larger as needed. If you wish to turn off this behavior, then use the SetResize() function. If you read an uninitialized index, "invalid" is returned.

roByteArray supports the ifArray interface, and so can be accessed with the array [] operator. The byte array is always accessed as unsigned bytes when using this interface. roByteArray also supports the ifEnum interface, and so can be used with a "for each" statement. In addition, the ifByteArray interface provides various functions listed below.

ifByteArray Interface Functions:

```
WriteFile(path) As Boolean
WriteFile(path , start_index, length) As Boolean
ReadFile(path) As Boolean
ReadFile(path, start_index, length ) As Boolean
AppendFile(path) As Boolean
```

.

```
   SetResize(minimum_allocation_size, autoresize) As Boolean
   ToHexString() As String
   FromHexString(hexstring) As Void
   ToBase64String() As String
   FromBase64String(base65string) As Void
   ToAsciiString() As String
   FromAsciiString(string)
   GetSignedByte(index) As Integer
   GetSignedLong(index) As Integer
   IsLittleEndianCPU() As Boolean
```

Here are a few examples of roByteArray:

```
ba=CreateObject("roByteArray")
ba.FromAsciiString("leasure.")
if ba.ToBase64String()<>"bGVhc3VyZS4=" then stop


ba=CreateObject("roByteArray")
ba.fromhexstring("00FF1001")
if ba[0]<>0 or ba[1]<>255 or ba[2]<>16 or ba[3]<>1 then stop


ba=CreateObject("roByteArray")
for x=0 to 4000
      ba.push(x)   ' will truncate when x bigger than 255
end for
ba.WriteFile("tmp:/ByteArrayTestFile")

ba2=CreateObject("roByteArray")
ba2.ReadFile("tmp:/ByteArrayTestFile")
if ba.Count()<>ba2.Count() then stop
for x=0 to 4000
      if ba[x]<>ba2[x] then stop
end for

ba2.ReadFile("tmp:/ByteArrayTestFile", 10, 100)
if ba2.count()<>100 then stop
for x=10 to 100
      if ba2[x-10]<>x then stop
end for
```

10.16 roXMLElement

Also see the section "BrightScript XML Support" for details of using the dot and @ shortcuts for many of these member functions.

roXMLElement is used to contain an XML tree. It has one interface: ifXMLElement

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

.

```
GetNamedElementsCi(As String) As Object
Parse(s As String) As Boolean
SetBody(As Object) As Void
AddBodyElement() As Oject
AddElement(As String) As Void
AddElementWithBody(As String, As Object) As Object
AddAttribute(As String, As String) As Void
SetName(As String) As Void
Parse(As String) As Boolean
GenXML(gen_header As String) As String
Clear() As Void
GenXMLHeader()
IsName(As String) Ad Boolean
HasAttribute(As String) As Boolean
```

Example:
```
<tag1>this is some text</tag1>
```

Would parse such that:
  Name = tag1
  Attributes = invalid
  Body = roString containing "this is some text"

```
<emptytag caveman="barney" />
```

Would parse such that:
  Name= emptytag
  Attributes = roAssociatveArray, with one entry {caveman, barney)
  Body = invalid

If the tag contains other tags, body will by of type "roXMLList".

To generate XML, create an roXMLElement, then use the "Set" functions to build it. Then call GenXML().

GenXML() takes one parameter (boolean) that indicates whether the generated xml should have the <?xml …> tag at the top.

AddBody() will create an roXMLList for body, if needed, then add the passed item (which should be an roXMLElement tag).

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
```

Example of generating XML:

```
    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)
```

Another Example

```
    xml = CreateObject("roXMLElement")
    xml.SetName("root")
    subel1 = xml.AddBodyElement()
    subel1.SetName("subelement1")
    subel2 = xml.AddBodyElement()
    subel2.SetName("subelement2")
```

Is the same as:

```
    xml = CreateObject("roXMLElement")
    xml.SetName("root")
    subel1 = xml.AddElement("subelement1")
    subel2 = xml.AddElement("subelement2")
```

10.17 roXMLList

Interfaces:

```
      ifList (documented elsewhere)
```

.

```
ifXMLList
   o  GetAttributes() As Object
   o  GetText() As String
   o  GetChildElements() As Object
   o  GetNamedElements(As String) As Object
   o  GetNamedElementsCi(As String) As Object
   o  Simplify() As Object
```

GetNamedElements() is used to return a new XMLList that contains all roXMLElements that matched the passed in name. This is the same as using the dot operator on an roXMLList.

Simplify will

- Otherwise, Return an roXMLElement if the list contains exactly one
- Otherwise, will return itself

GetAttributes() and GetText() are similar to calling xmllist.Simplify().GetText(), xmllist.Simplify().GetAttributes().

.

11.0 Core BrightScript Components new in v3.0

One of the goals of BrightScript 3.0 is to support the development of 2D games. To this end, BrightScript 3.0 enables drawing to offscreen bitmaps, creating animations, managing sprites, drawing in z-order layers, detecting collisions, and playing sounds.

The following set of core BrightScript Components were added in BrightScript 3.0.

- roScreen (interfaces: ifDraw2D, ifSetMessagePort, ifGetMessagePort, ifScreen)
- ro UniversalControlEvent (interfaces: ifUniversalControlEvent)
- roBitmap (interfaces: ifDraw2D)
- roRegion (interface: ifRegion, ifDraw2D)
- roCompositor (interface: ifCompositor)
- roSprite (interface: ifSprite)
- roAudioResource (interface: ifAudioResource)
- roFont
  roFontRegistry

11.1 roScreen

The roScreen component provides a full screen drawing surface that can be stacked and that you can receive input events from. You will need at least one roScreen component in your 2D game application to draw on and get events from. The origin (0,0) is the top left corner of the screen. The pixels are always RGBA 32 bits. Multiple roScreen components stack, and like other screen components only the top screen is viewable and gets events. An roScreen that is not the top most screen can still be drawn to. Once an roScreen is created, the display stack enters "Game Mode", and other screen components documented in the Component Reference cannot be used. Other screen components cannot be intermixed with roScreens as the roScreen display stack is maintained independently from the main screen component display stack. When the final roScreen component is closed, other screen components can be used again.

When the roScreen constructor is called, an optional double buffer flag, and an optional resolution can be passed. If the screen is double buffered, two buffers are fully allocated when CreateObject() succeeds. If the screen is single buffered only one buffer is allocated and the "front" and "back" buffers in method descriptions below are the same buffer. When a screen is created with a different resolution than the output display, it is scaled automatically to the output display resolution.
To maintain proper aspect ratio, and take care of the different pixel aspect ratio in HD vs SD; there is a fixed set of bitmap resolutions that are allowed to be created for screens:
HD mode screensizes:

1280x720 PAR=1:1 (default for HD)
854x480 PAR=1:1 useful for higher performance HD games, also for 640x480 games
940x480 PAR=1.1:1 used for displaying a RokuSD (720x480) games

SD mode screensizes:

720x480 PAR=1.1:1 (default for SD)
640x480 PAR=1:1 (used for 640x480 games)
854x626 PAR=1:1 (used for 854x480 HD games)

The screen dimensions correspond to the drawable area that applications see. . The dimensions were chosen so that applications do not need to compensate for screen aspect ratio or pixel aspect ratio.

It's likely that when porting games from other platforms, the active game area may be smaller and correspond to more traditional dimensions. In this case, the application cansupplyletterbox or pillarbox artwork and use an roRegion to define the active area. The roRegion will translate and

.

clip graphics to the proper area for the game. Similarly, roRegions are used to describe the left and right pillars for an SD game in HD mode, or the upper and lower letterbox regions for an HD game in SD mode.

Games that require more performance should use smaller dimensions.

Games should run in HD and SD mode. The screensizes HD 854x480 paired with SD 854x626 and HD 940x480 paired with SD 720x480 were designed for this purpose. The game creates a single active game roRegion to do all graphics operations in. roRegions for pillar or letter boxes are used to fill the rest of the screensize area depending on if the app is in HD or SD mode. Please refer to the `dfSetupDisplayRegions() function in section 4.12.2 for help in setting up the drawable regions in screen scaling.`

There are some useful rules of thumb to be aware of to get the best performance when rendering your games:

- Alpha enabled regions are expensive to render.

It is a requirement that the destination be alpha enabled in order for non-rectangular sprites to be properly rendered with transparency. However the sprite used for a background would typically have all pixels be fully nontransparent. Since alpha blending is expensive, a quick way to blit the screen in these scenarios is to first disable alpha enable on the screen, manually draw the background, and then enablealpha for the screen before drawing the rest of the sprites.

- Use smaller resolution images wherever possible. Scaling a large image down at run time is expensive with no benefit to the user.

- Rendering text with DrawText() is expensive

Fortunately, many of these calls are redundant and can be eliminated. The static text for a particular level can be drawn on the background once and this newly created background can be used for refreshing the screen. This will eliminate almost all text redraws.

A screen can be created with one of three constructors.

If it is created with no parameters, the screen will be single buffered, and its output resolution will match the current display resolution

- `CreateObject("roScreen")`

If a single parameter is passed, it is a Boolean that indicates if the screen is double buffered or not. See SwapBuffers():

- `CreateObject("roScreen", true)  ‘ double buffered screen`

If three parameters are passed, the second two specify the screen’s resolution. The dimensions must be one of the screen sizes specified above.

- `CreateObject("roScreen", true, 720, 480)  ‘ db & SD res`

roScreen has the following interfaces:

Interface: ifScreen

```
Void SwapBuffers()
```

.

- This function first operates the same as a call to ifDraw2D.Finish(), completing all queued drawing operations on the back buffer (draw surface). If the screen is single buffered, SwapBuffers() returns after this operation as the back buffer is also the front buffer.
- If the screen is double buffered, SwapBuffers swaps the back buffer with the front buffer so it is now visible. The back buffer should be assumed to be in a garbage state after this call is complete. Which means you will need to re-render the entire frame before a subsequent call to SwapBuffers. This call will not return until the back buffer is ready to be drawn on to. Depending on the implementation, it may take up to a single frame period for the new front buffer to become visible.
- This operation is extremely fast and guaranteed not to "tear" the visible image.

Interface: ifDraw2D

The roScreen object implements the ifDraw2D interface. Coordinates (x,y) for this interface are based on an origin (0,0) at the top, left. This is common for 2D drawing APIs, but is different than OpenGL's default coordinate system.

This interface provides:

`Void Clear(Integer rgba)`
- Clear the screen, and fill with the 32 bit rgba color value. The bitmap color depth is always 32 bits, and the alpha will be filled into the bitmap, even when not used. Once AlphaEnable is set to true, the alpha bits will be taken into account when using this bitmap as a source. See SetAlphaEnable() for more information on alpha blending. Note that Clear() is not the same as a DrawRect() for the entire screen. Clear() fills the bitmap with the complete RGBA; it does not perform any alpha blending operations.

`Integer GetWidth()`
- Returns the width of the bitmap.

`Integer GetHeight()`
- Returns the height of the bitmap.

`Object GetByteArray(Integer x, Integer y, Integer width, Integer height)`
- Returns an roByteArray representing the 32 bit RGBA pixel values for the passed in rect region of top left coordinate x,y and height, width.

`Boolean GetAlphaEnable()`
- Returns True if the Alpha blending is enabled.

`Void SetAlphaEnable(Boolean enable)`
- If enable is true, do alpha blending when this bitmap is the destination. The setting of the source bitmap's alphaenable is ignored.
- When turned on, each pixel in the destination bitmap is set by combining the destination and source pixels according to the alpha value in the source bitmap (or rect). The destination alpha is not used. In OpenGL this is referred to as GL_ONE_MINUS_SRC_ALPHA
- By default, alpha blending is off
- Even when alpha blending is off, the alpha value is still present in the bitmap, and must be passed when a function parameter is a color (which are always RGBA).

  Here's an example:

```
Function Main()
        s=CreateObject("roScreen")
        ' Clear to White with alpha fully opaque
        ' but alpha not actually ever used since it is the bottom most plane
        ' alpha is only looked at on "source" planes, not "destination".
        s.Clear(&hFFFFFFFF)

        'AlphaEnable must be enabled in the destination surface to have effect.
        s.SetAlphaEnable(true)
```

.

```
            bm=CreateObject("roBitmap", {width:100, height: 100, alphaenable: false} )
            bm.Clear(&hFFFF)  'blue, fully opaque alpha

            ' draw a blue rect in the upper left corner
            s.DrawObject(0,0, bm)
            s.Finish()

            Sleep(2000)
            s.Clear(&hFFFFFFFF)
            bm.Clear(&hFF00)  'blue, fully transparent alpha

            ' draw a blue rect in the upper left corner
            ' but, since it is transparent, nothing will appear on the screen.
            s.DrawObject(0,0, bm)
            s.Finish()
            Sleep(2000)
        End Function
```

`Boolean DrawRect(Integer x, Integer y, Integer width, Integer height, Integer rgba)`

- Return True if drawn OK.
- Draw the rectangle at position x,y with width and height. Fill with rgba color.

`Boolean DrawLine(Integer xStart, Integer yStart, Integer xEnd, Integer yEnd, Integer rgba)`

- Return True if drawn OK.
- Draw the line from (xStart, yStart) to (xEnd, yEnd) with rgba color.

`Boolean DrawObject(Integer x, Integer y, Object src)`

- Return True if drawn OK.
- Draw the source src at position x,y. src is an roBitMap or an roRegion Object.

`Boolean DrawScaledObject(Int x, Int y, float scaleX, float scaleY, Object src)`

- Return True if drawn OK.
- Draw the source src at position x,y with scaleX and scaleY. src is an roBitMap or an roRegion Object.

`Boolean DrawRotatedObject(Int x, Int y, float theta, Object src)`

- Return True if drawn OK.
- Draw the source src at position x,y rotated by angle theta.. src is an roBitMap or an roRegion Object.
- Theta is currently limited to 0, 90, 180, and 270 degrees.

`Void  Finish()`

Realize the bitmap by finishing all queued draw calls. Until Finish() is called, prior graphics operations may not be user visible. For example, they may be in the graphics display pipeline, or in a server queue. Finish() is synchronous, it does not return until all graphic operations are complete.

`Boolean DrawText(String  text, Integer x, Integer y, Integer rgba, Object font)`

- Return True if drawn OK.
- Text is the string to render at position (x,y) using the passed in roFont object font and color RGBA
- Text will show the background image/color behind it. To have the text erase the background, use a DrawRect() before the DrawText()
- The size, bold, and italic are specified when creating the roFont.
- Text is drawn anti-aliased

Interface: ifSetMessagePort

`Void  SetPort(roMessagePort port)`

Set the message port to be used for all events from the screen.

.

Interface: ifGetMessagePort

```
Object GetPort(Void)
```
Returns the message port currently set for use by the screen.

Events: roUniversalControlEvent

The roScreen object sends the roUniversalControlEvent, which has one method. If an app constrains the events processed to just the roUniversalControlEvent, the app will work with any controller.

```
Integer GetInt(Void)
```
- Returns an integer representing pressed or released keys on the remote. The integers are defined in bslCore.brs: bslUniversalControlEventCodes():

0 – Back Pressed	100 – Back Released
2 – Up Pressed	102 – Up Released
3 – Down Pressed	103 – Down Released
4 – Left Pressed	104 – Left Released
5 – Right Pressed	105 – Right Released
6 – Select Pressed	106 – Select Released
7 – Instant Replay Pressed	107 – Instant Replay
8 – Rewind Pressed	108 – Rewind Released
9 – Fast Forward Pressed	109 – Fast Forward Released
10 – Info Pressed	110 – Info Pressed
13 – Play Pressed	113 – Play

Example Code: roScreen

```
Screen=CreateObject("roScreen")
dfDrawImage(screen, "myphoto.jpg",0,0)
Screen.Finish()
```

Example Code: Alpha Blending on roScreen

```
white=&hFFFFFFFF
screen0=CreateObject("roScreen")
screen0.SetAlphaEnable(true)
screen0.Clear(white)
screen0.DrawRect(100,100, screen0.GetWidth()-200,
screen0.GetHeight()-200, &h80)
‘ &h80 is black with a 50% alpha mix (RGBA)
screen0.finish()
```

11.2 roBitmap

The roBitmap component represents an allocated bitmap surface, and provides an interface (ifDraw2D) to a bitmap surface for drawing. Bitmaps can be used for a variety of purposes, such as

.

for sprites, compositing, or as double buffers.

- roBitmaps are always RGBA 32 bit and can be any arbitrary size, up to a maximum of 2048x2048.. Note that a 2K x2K x 32-bit bitmap uses 16MB of memory, so there are practical memory limitations which would generally compel smaller bitmap sizes.
- Coordinates (x,y) for 2D bitmaps are based on an origin (0,0) at the top, left.
- roBitmap is always offscreen. The top roScreen is the only ifDraw2D surface which is displayed.
- roBitmap represents something that can be drawn onto, as well as something that can be drawn.
- Drawing operations into a roBitmap (or other surface with ifDraw2D interface, such as an roScreen) will clip such that the area of the Image, Bitmap, Text String, Rect, etc. that is in the display area will display, and the rest will clip. X,Y coordinates that specify a location in a bitmap to render to (for example, as used by DrawObject() or DrawText()) may be positive or negative. Negative implies that the start of the object being rendered will clip.The same bitmap cannot be used as a source and a destination in a single DrawObject() call.
- There are limitations when using the onscreen bitmap as a source. For example, Alpha blending may not work. (TODO: VERIFY THIS)

The roBitmap object can be created with a couple of overloaded CreateObject() calls:

- `CreateObject("roBitmap", Object bitmapProps)`
  Where `bitmapProps` is an roAssociativeArray with Integers width, height, and Boolean AlphaEnable parameters. Example:
  `CreateObject("roBitmap", {width:10, height:10, AlphaEnable:false})`

- `CreateObject("roBitmap", String filename)`
  This form will load a PNG or JPEG and create an roBitmap from it.

roBitmap has the following interfaces:

Interface: ifDraw2D

(see roScreen discussion for documentation on this interface)

Example Code: roBitmap

```
'
' Draw three bitmaps as fast as we can
'
Screen=CreateObject("roScreen")
bm1=CreateObject("roBitmap", "myphoto1.jpg")
bm2=CreateObject("roBitmap", "myphoto2.jpg")
bm3=CreateObject("roBitmap", "myphoto3.jpg")
bmarray=[bm1, bm2, bm3]
While true
For each bitmap in bmarray
      Screen.DrawObject(0,0, bitmap)
      Screen.Finish()
End for
End While
```

Example Code: Double Buffering with roBitmap

.

```
screen1=CreateObject("roScreen")
off=createobject("roBitmap", {width:1280, height:720, AlphaEnable:
false})
off.Clear(white)
dfDrawImage(off, "myimage.png",50,50)
off.DrawRect(150,150, 200, 200, &hFF)  ' black, alpha: all source
screen1.drawObject(0,0,off)
Screen1.Finish()
```

11.3 roRegion

The roRegion component is used to represent a subsection of a bitmap. The region is specified with an x,y, width, and height as well as a time field for use with animated sprites and a wrap field which causes the region to wrap during scrolling. The roRegion is a common parameter used by the drawing functions of roBitmap. Wrap and Time are used by roCompositor. roRegion is also used to specify a pretranslation (x,y) for the draw, rotate, and scale operation. The pretranslation is normally used to specify the center of the region. The scaling operation is controlled by the scalemode specified in the region.

This object is created with parameters to initialize the x,y coordinates, width, height. If time and wrap are desired, use the SetTime() and SetWrap().

- `CreateObject("roRegion", Object bitmap, Integer x, Integer y, Integer width, Integer height)`

The roRegion component implements the following interfaces:

Interface: ifRegion

`Integer GetX()`
- Returns the x coordinate of the region.

`Integer GetY()`
- Returns the y coordinate of the region.

`Integer GetWidth()`
- Returns the width of the region.

`Integer GetHeight()`
- Returns the height of the region.

`Integer GetTime()`
- Returns the time or "frame hold time" in milliseconds.

`Boolean GetWrap()`
- Returns true if the region will wrap to the other side of the screen when scrolled.

`Object GetBitmap()`
- Returns the roBitmap object of the bitmap this region refers to. A region is always a section of a bitmap.

`Integer GetPretranslationX()`
- Returns the pretranslation x value.

`Integer GetPretranslationY()`
- Returns the pretranslation y value.

`Integer GetScaleMode()`
- Returns the scaling mode.

.

`Void Set(Object srcRegion)`

- Takes an roRegion object as input
- Initializes the fields of this region to the values of the fields in the srcRegion.

`Void SetTime(Integer time)`

- Set the time or "frame hold time" in milliseconds.

`Void SetPretranslation(Integer x, Integer y)`

- Set the pretranslation for drawobject,drawrotatedobject, drawscaledobject

`Void SetScaleMode(Integer mode)`

- Set the scalemode used for DrawScaledObject
- 0 = fast scaling operation (may have jaggies)
- 1 = smooth scaling operation (may be slow)

`Void SetWrap(Boolean wrap)`

- If wrap is specified as true, a region that is partial displayed (for example, the right half falls "off" the roCompositor drawto surface), then the region "wraps" to the other side of the roCompositor surface.

`Void OffSet(Integer x, Integer y, Integer w, Integer w, Integer h)`

- Adds the passed parameters x,y, w, and h to the values of those roRegion fields
- Respects the wrap setting when adjusting the fields by the offsets.

`Object Copy()`

- Returns a newly created copy of the region as a new roRegion object.

`Integer GetCollisionType()`

- Returns the value of collision type variable.
- The default value is 0.

`Void SetCollisionType(Integer collisiontype)`

- Sets the type of region to be used for collision tests with this sprite.
- Type 0– Use the entire defined region of the sprite. Type 0 is the default.
- Type 1 – Use the defined rectangular region specified by the SetCollisionRectangle() method.
- Type 2 – Use a circular region specified by the SetCollisionCircle() method.

`Void SetCollisionRectangle(Integer xOffset, yOffset, XWidth, XHeight)`

- Sets the collision rectangle used for type 1 collision tests. The offset is added to the (x,y) position of the sprite and the resulting rectangle is used for collision tests.

`Void SetCollisionCircle(Integer xOffset, yOffset, Radius)`

- Sets the collision circle used for type 2 collision tests. The offset is added to the (x,y) position of the sprite and a circle is created around that point for collision tests.

Designing a single app for both SD and HD

There are two ways to design a single app for SD and HD.

1. Create an roScreen, determine the dimensions, and then make appropriate changes to your app such that it works properly in HD and in SD mode. This would include using different artwork, different animations, and different movement maximums.
2. Choose an aspect ratio that your game will work best in; SD or HD. Determine if the Roku is running in SD or HD mode. Open up the appropriate screen for that combination. Create an roRegion that defines the region you will do all your graphics in. Optionally create the roRegions to apply some graphics to the pillar or letter boxed area. Start your game.

See the simple2d app for examples of creating different screens and drawing an animation in the proper area.

11.4 roCompositor

The roCompositor allows the composition and animation of multiple roBitmaps and roRegions. The roCompositor can create and manage roSprites in a z-ordered list. The sprites can be of arbitrary size and can be thought of as planes. The compositor can manage collision detection between the sprites, support scrolling the sprite bitmap source, and support animated sprites (multi-frame sprites with frame-flipping animation). You may have multiple roCompositor components, and they

.

can composite onto the same or separate bitmaps. That said, the most common scenario is to have a single roCompositor

Interface: ifCompositor

`Void SetDrawTo(Object destBitmap, Integer rgbaBackground)`
- Set the destBitmap (roBitmap or roScreen) and the background color.

`Void Draw()`
- Draw any dirty sprites (that is, whatever is new or has changed since the last Draw). No compositor or sprite operations will be reflected on the display until Draw() is called.
- After calling Draw(), you must call Finish() (if single buffered) or SwapBuffers() (if double buffered) before the changes will be user visible.

`Void  DrawAll()`
- Redraw all sprites even if not dirty.
- After calling Draw(), you must call Finish() (if single buffered) or SwapBuffers() (if double buffered) before the changes will be user visible.

`Object NewSprite(Integer x, Integer y, Object region, Integer z)`
- Returns an roSprite object
- Create a new sprite, using an roRegion to define the sprite's bitmap. Position the sprite at coordinate x,y.
- If z is provided, position the sprite in front of all other sprites with equal or lower z value. Negative z values are not rendered are displayed on the screen.

`Object NewAnimatedSprite(Integer x, Integer y, Object regionArray, Integer z)`
- Returns an roSprite object.
- Create a new sprite that consists of a sequence of frames to be animated. The frames are defined by the regionArray which is an roArray of roRegions
- Position the sprite at coordinate x,y.
- If z is provided, position the sprite in front of all other sprites with equal or lower z value

`Void  AnimationTick(Integer duration)`
- Duration is the number of ms since the last call.
- Moves all animated sprites.
- Sprites will not animate unless you call this function regularly.

`Void  ChangeMatchingRegions(Object oldRegion, Object newRegion)`
- Global search and replace of Sprite roRegions.
- Replaces regions that match oldRegion with newRegion

Example Code: Scrolling a bitmap

```
function main()
         black=&hFF                              'RGBA
         screen=CreateObject("roScreen")
         compositor=CreateObject("roCompositor")
         compositor.SetDrawTo(screen, black)
         bigbm=CreateObject("roBitmap","pkg:/assets/VeryBigPng.png")
         region=CreateObject("roRegion", bigbm, 0, 0, 1280, 720)
         region.SetWrap(true)
         view_sprite=compositor.NewSprite(0, 0, region)
         compositor.draw()

          msgport = CreateObject("roMessagePort")
          screen.SetMessagePort(msgport)

         codes = bslUniversalControlEventCodes()
         while true
        msg=wait(0, msgport)   ' wait for a button
        if type(msg)="roUniversalControlEvent" then
           if msg.GetInt()=codes.BUTTON_UP_PRESSED then
                  Zip(view_sprite, compositor, 0,-4)  'up
           else if msg.GetInt()=codes.BUTTON_DOWN_PRESSED then
```

.

```
                Zip(view_sprite, compositor, 0,+4)  ' down
            else if msg.GetInt()=codes.BUTTON_RIGHT_PRESSED then
                Zip(view_sprite, compositor, +4,0)  ' right
            else if msg.GetInt()=codes.BUTTON_LEFT_PRESSED then
                Zip(view_sprite, compositor, -4, 0)  ' left
            else if msg.GetInt() = codes.BUTTON_BACK_PRESSED ‘ back button
                exit while
            end if
          end if
        end while
end function

function Zip(view_sprite, compositor, xd, yd)
        for x=1 to 60
                view_sprite.OffsetRegion(xd, yd, 0, 0)
                compositor.draw()
        end for
end function
```

11.6 roSprite

The roSprite object cannot be created directly with a CreateObject() call. It must be associated with a managing roCompositor object. This association is implicitly created by creating an roSprite object with the roCompositor methods NewSprite() or NewAnimatedSprite().

Interface: ifSprite

`Integer GetX()`
- Returns the x coordinate of the sprite.

`Integer GetY()`
- Returns the y coordinate of the sprite.

`Integer GetZ()`
- Returns the z value of the sprite.
- The default z value is 0.

`Boolean GetDrawableFlag()`
- Returns the value of the Drawable Flag.
- The default value is true.

`Integer GetMemberFlags()`
- Returns the value of member flags variable.
- The default value is 1.

`Integer GetCollidableFlags()`
- Returns the value of collidable flags variable.
- The default value is 1.

`Object GetRegion()`
- Returns an roRegion object that specifies the region of a bitmap that is the sprite’s display graphic.

`Void SetZ(Integer z)`
- Sets the z value of the sprite. This will place the sprite in the list such that it will be drawing in front of any other sprite of lesser or equal z value.

`Void SetDrawableFlag(bool enable)`
- Sets whether this sprite is drawable or just used for collision tests. An undrawable sprite can be used to define a region in the background that needs collision testing. It can also be used as an auxiliary collision region for a more complex sprite defined in another sprite. The default value of true is set when a sprite is created.

`Void MoveTo(Integer x, Integer y)`
- Move the sprite to coordinate x,y.

`Void MoveOffset(Integer xOffset, yOffset)`
- Move the sprite to the current position plus the xOffset and yOffset.

.

`Void SetRegion(Object region)`
- Set the region of the sprite to the passed in region roRegion object. If one already is set, it is replaced.

`Void SetData(TVAL data)`
- Associate user defined data with the sprite. The TVAL (typed value) can be any type including intrinsic types or objects.

`TVAL GetData()`
- Returns any user data associated with the sprite. Set via SetData().
- Returns invalid if there is no user data associated with this sprite.

`Void Remove()`
- Remove the sprite from the managing roComposite object and delete the sprite.

`Object CheckCollision()`
- Returns the first roSprite that this sprite collides with. The collision area is the entire sprite's bounding box, and the sprites must actually be overlapped to detect a collision. That is, if a fast moving sprite moves "through" another sprite w/o actually overlapping when this call is made, no collision is detected.

`Object CheckMultipleCollisions()`
- Like CheckCollision but returns an array of all colliding sprites. If there are no collisions return invalid.

`Void SetMemberFlags(Integer flags)`
- Sets flags to define the sprite membership.These flags are used with CollidableFlags to define what sprites are allowed to collide.
- Enables "levels" of collision detection, as only sprites with a member flag bit that matches a collidable flag bit will be checked for collisions.

`Void SetCollidableFlags(Integer flags)`
- Sets bits to determine what sprites will be checked for collisions. The sprites that are checked must have the corresponding bits sets in their MemberFlags.

`Void OffsetRegion(Integer x, Integer y, Integer width, Integer height)`
- Calls Region.Offset() on this Sprite's region. Adjusts the part of an roRegion's bitmap that is being displayed as the sprite. Wrap is taken into consideration.

11.7 roAudioResource

The roAudioResouce allows .wav files to be cached to memory and quickly played at any time. The filename must be a local file and cannot be an url. You may download Urls to the application's "tmp:" file system and pass filenames of the form "tmp:/assetname" to the constructor. The roAudioResource does not currently support mixing of sounds. So when you play a sound effect, any background music is paused while the sound effect plays and then resumes after the sound effect ends

This object is created with a filename parameter that is a path to the sound resource file:
- `CreateObject("roAudioResource", String filename)`

The roAudioResource implements the ifAudioResource interface:

Interface: ifAudioResource

`Integer Trigger(Integer volume)`
- This method triggers the start of the audio resource sound playback
- The volume is a number between 0 and 100 (percentage off full volume).

`Boolean IsPlaying()`
- True if this audio resource is currently playing

`Object GetMetaData()`

.

- Returns an roAssociative array containing the following meta data parameters about the sound resource:
  - Length (# samples)
  - SamplesPerSecond
  - NumChannels
  - BitsPerSample

11.8 roFontRegistry

The roFontRegistry object allows you to discover system fonts and to register TrueType and OpenType fonts packaged with your application. Font files can quickly get very large, so be conscious of the size of the font files you include with your application. You should be able to find very good font files that are 50k or less. The font is available only for as long as the roFontRegistry object that registered it is not deleted.
Interface: ifFontRegistry

The roFontRegistry object implements the ifFontRegistry interface. This interface provides:

`Boolean Register(String path)`
- Register a font file (.ttf or .otf) as an available font Path is a valid filesystem path name (e.g. pkg:/myfont.ttf)

`roArray GetFamilies(Void)`
- Returns an array of strings that represent the font family names for all font files registered with the roFontRegistry

`Object GetFont(String Family, integer size, Boolean bold, Boolean italic)`

- Returns a valid roFont object that can be used in the DrawText() ifDraw2d interface method.
- Family is a registered family name as returned by GetFamilies()
- Size, bold, italic are additional font characteristics that the .ttf or .otf files must support.
- Size is in "pixels", not "points".

`Object GetDefaultFont(integer size, Boolean bold, Boolean italic)`
- Returns the system default font with the specified size, boldness, and italicness

`Object GetDefaultFont()`
- Returns the system default font in its default size

`Integer GetDefaultFontSize()`
Returns the default size for "normal" text, in pixels

11.9 roFont

roFont represents a particular font, from a font-family (eg. Arial), with a particular pixel size (e.g 20), and a particular boldness or italicness.

`Integer GetOneLineHeight()`
Returns the number of pixels from one line to the next when drawing with this font.

`Integer GetOneLineWidth(String text, Integer MaxWidth)`
Returns the width in pixels for this particular string, whe rendered with this font. Each glyph and the needed spacing between glyphs is measured. The returned number of pixels will be no larger than MaxWidth. MaxWidth is generally the amount of pixels available for rendering on this line.

Example: roFont

```
screen=CreateObject("roScreen")
```

.

```
white = &hFFFFFFFF
blue =  &h0000FFFF
font_registery = CreateObject("roFontRegistry")
font_regular = font_registery.GetDefaultFont()

screen.DrawText("Line One String", 100, 50, white, font_regular)
screen.DrawText("Next Line", 100, 50+font_regular.GetOneLineHeight(), blue, font_regular)
```

.

Appendix A – BrightScript Debug Console

When a script is running, if a runtime error is encountered, or the "stop" statement is encountered, the BrightScript debug console is entered. Access to the console is device specific. For example, on a Roku player you telnet to port 8085. On BrightSigns, it is the main serial port (connect it to a PC with a null-modem cable and use a terminal program).

Use the "help" command to get a list of commands you can use. For example, "var" will list all in scope variables and their types and values. The Scope is set to the function that was running when the error was encountered.

One of the most powerful things you can do at the debug console is to type in any BrighScript statement. It will be compiled and execute in the current context.

Typically the default device (or window) for the "print" statement, and the debug console, are the same.

Insert "stop" statements in your code to set a break point, and use "c", "s", "t" to continue or step execution.

As of this writing, the following debugger commands are available:

```
bt          Print backtrace of call function context frames
classes     List public classes
cont or c   Continue Script Execution
da          Show disassembly and bytecode for this function
down or d   Move down the function context chain one
exit        Exit shell
gc          Run garbage collector
hash        Internal hash tables histogram
last        Show last line that executed
next        Show the next line to execute
ld          Show line data (source records)
list        List current function
bsc         List BrightScript Component instances
stats       Shows statistics
step or s   Step one program statement
t           Step one statement and show each executed opcode
up or u     Move up the function context chain one
var         Display local variables and their types/values
print,?,p   Print variable value or expression
```

Since BrightScript v3.0:

```
t           Step one Op Code at a time
da          Show Disassembly of current function
```

.

Appendix B – Example Script - Snake

Snake.brs

```
---------------------------------------
Library "v30/bslDefender.brs"

'
' The game of Snake
' Demonstrates BrightScript programming concepts
' August 24, 2010


function Main()

    app=newSnakeApp()
    dfDrawMessage(app.screen, app.bitmapset.regions["title-screen"])
    app.screen.swapbuffers()
    while true
        msg=wait(0, app.msgport)
        if type(msg)="roUniversalControlEvent" then exit while
    end while

    while true
        app.GameReset()
        app.EventLoop()
        app.GameOverSound.Trigger(100)
        if app.GameOver() then ExitWhile
    end while

end function


' ******************************************************
' ******************************************************
' ***************                    *******************
' ***************         APP        *******************
' ***************                    *******************
' ******************************************************
' ******************************************************


'
' newSnakeApp() is regular Function of module scope.
' The object container is a BrightScript Component of type roAssocitiveArray (AA).
' The AA is used to hold member data and member functions.
'

Function newSnakeApp()
    app={ }       ' Create a BrightScript roAssociativeArray Component
    app.GameReset=appGameReset
    app.EventLoop=appEventLoop
    app.GameOver=appGameOver

    app.screen=CreateObject("roScreen", true)  ' true := use double buffer
    if type(app.screen)<>"roScreen" then
        print "Unable to create roscreen."
        stop   ' stop exits to the debugger
    endif

    app.screen.SetAlphaEnable(true)

    app.bitmapset=dfNewBitmapSet(ReadAsciiFile("pkg:/snake_assets/sprite.small.map.xml"))
    if (app.bitmapset=invalid) then stop

    app.cellwidth=app.bitmapset.extrainfo.cellsize.toint()' each cell on game in pixels width
    app.cellheight = app.cellwidth
    app.msgport = CreateObject("roMessagePort")
    app.screen.SetPort(app.msgport)

.
```

```
    app.StartX=int(app.screen.GetWidth()/2)
    app.StartY=int(app.screen.GetHeight()/2)

    app.TurnSound=CreateObject("roAudioResource", "pkg:/snake_assets/cartoon002.wav")
    app.GameOverSound=CreateObject("roAudioResource", "pkg:/snake_assets/cartoon008.wav")

    return app

End Function

Function appEventLoop() As Void

    tick_count=0
    codes = bslUniversalControlEventCodes()

    while true
     msg=wait(1, m.msgport)   ' wait for a button,
      if type(msg)="roUniversalControlEvent" then
       if msg.GetInt()=codes.BUTTON_UP_PRESSED and m.snake.Turn(m,0,-1) then return  ' N
       if msg.GetInt()=codes.BUTTON_DOWN_PRESSED and m.snake.Turn(m,0,1) then return ' S
       if msg.GetInt()=codes.BUTTON_RIGHT_PRESSED and m.snake.Turn(m,1,0) then return' E
       if msg.GetInt()=codes.BUTTON_LEFT_PRESSED  and m.snake.Turn(m,-1,0) then return  ' W
       else  'here if time out happened, move snake forward
            tick_count=tick_count+1
            if tick_count mod 3 = 0 then
                if m.snake.MakeLonger(m) then return
            else if tick_count mod 2 = 0 then
                if m.snake.MoveForward(m) then return
            end if
            m.compositor.AnimationTick(1)
        end if
       m.compositor.DrawAll()
       m.screen.SwapBuffers()
   end while

End Function

Sub appGameReset()

    m.compositor=CreateObject("roCompositor")
    m.compositor.SetDrawTo(m.screen, 0) ' 0 means "no background color".  Use &hFF for black.

    width=int(m.screen.GetWidth()/m.cellwidth)
    height=int(m.screen.GetHeight()/m.cellheight)
    water=m.bitmapset.animations.water
    for x=0 to width-1
        m.compositor.NewAnimatedSprite(x*m.cellwidth, 0, water)
        m.compositor.NewAnimatedSprite(x*m.cellwidth, m.cellheight, water )
        m.compositor.NewAnimatedSprite(x*m.cellwidth,  (height-2)*m.cellheight, water )
        m.compositor.NewAnimatedSprite(x*m.cellwidth,  (height-1)*m.cellheight, water )
    end for

    for y=1 to height-2
        m.compositor.NewAnimatedSprite(0, y*m.cellheight, water )
        m.compositor.NewAnimatedSprite(m.cellwidth, y*m.cellheight, water )
        m.compositor.NewAnimatedSprite(m.cellwidth*2, y*m.cellheight, water )
        m.compositor.NewAnimatedSprite((width-3)*m.cellwidth,  y*m.cellheight, water )
        m.compositor.NewAnimatedSprite((width-2)*m.cellwidth,  y*m.cellheight, water )
        m.compositor.NewAnimatedSprite((width-1)*m.cellwidth,  y*m.cellheight, water )
    end for

    m.compositor.NewSprite(0, 0, m.bitmapset.Regions.Background).SetCollidableFlag(false)

    m.snake=newSnake(m, m.StartX, m.StartY)

    m.compositor.Draw()

End Sub

Function appGameOver()
    codes = bslUniversalControlEventCodes()

.
```

```
    m.compositor.DrawAll()
    dfDrawMessage(m.screen, m.bitmapset.regions["game-over"])
    m.screen.SwapBuffers()

    while true
        msg=wait(0, m.msgport)
        if type(msg)="roUniversalControlEvent"
            if msg.GetInt()=codes.BUTTON_SELECT_PRESSED return false else return true
        end if
    end while

End Function


' ******************************************************
' ******************************************************
' ******************              ***********************
' ****************** SNAKE OBJECT ***********************
' ******************              ***********************
' ******************************************************
' ******************************************************

'
' construct a new snake BrightScript object
'
Function newSnake(app, x, y)

    snake = {   ' Use AA Operator { }

        Turn : snkTurn
        MoveForward : snkMoveForward
        MakeLonger : snkMakeLonger

        dx : 1      ' default snake direction
        dy : 0      ' default snake direction

        DirectionName : function(xdelta, ydelta, base)
            if xdelta = 1 then
               dir="East"
            else if xdelta = -1 then
               dir="West"
            else if ydelta = 1 then
               dir = "South"
            else
               dir = "North"
            end if
            return base+dir
        end function

        RegionName : function(xdelta, ydelta, base)
            if xdelta = 1 then
               dir="East"
            else if xdelta = -1 then
               dir="West"
            else if ydelta = 1 then
               dir = "South"
            else
               dir = "North"
            end if
            return "snake."+base+dir
        end function



    }


    snake.tongue=app.compositor.NewAnimatedSprite(x,        y,
app.bitmapset.animations[snake.DirectionName( 1, 0, "tongue-")] )


.
```

```
           head=app.compositor.NewSprite(x-app.cellwidth,y,
app.bitmapset.regions[snake.RegionName( 1, 0, "head-")] )
           body=app.compositor.NewSprite(x-2*app.cellwidth, y,
app.bitmapset.regions[snake.RegionName( 1, 0, "body-")] )
    snake.tail= app.compositor.NewSprite(x-3*app.cellwidth, y,
app.bitmapset.regions[snake.RegionName( 1, 0, "butt-")] )

    snake.tail.SetData( {dx: 1, dy: 0, next: body,         previous: invalid} )
           body.SetData( {dx: 1, dy: 0, next: head,         previous: snake.tail} )
           head.SetData( {dx: 1, dy: 0, next: snake.tongue,  previous: body} )
    snake.tongue.SetData( {dx: 1, dy: 0, next: invalid,      previous: head} )

    snake.tongue.SetCollidableFlag(false)

    return snake

End Function


Function snkMoveForward(app)
    sprite=m.tail
    m.tail=m.tail.GetData().next
    m.tail.GetData().previous=invalid
    sprite.Remove()
    m.tail.SetRegion(app.bitmapset.regions[m.RegionName(m.tail.GetData().dx,
m.tail.GetData().dy, "butt-")])
    return m.MakeLonger(app) ' This isnt actually making the snake longer, its just the 2nd
half of MoveForward()
End Function

Function snkMakeLonger(app)
    newbody_x=m.tongue.GetX()-m.dx*app.cellwidth
    newbody_y=m.tongue.GetY()-m.dy*app.cellheight
    newbody=app.compositor.NewSprite(newbody_x, newbody_y,
app.bitmapset.regions[m.RegionName( m.dx, m.dy, "body-")] )

    m.tongue.MoveOffset(m.dx*app.cellwidth, m.dy*app.cellheight)
    head=m.tongue.GetData().previous
    head.MoveOffset(m.dx*app.cellwidth, m.dy*app.cellheight)

    body=head.GetData().previous
    head.GetData().previous=newbody
    body.GetData().next=newbody
    newbody.SetData( {dx: m.dx, dy: m.dy, next: head , previous: body} )

    return head.CheckCollision()<>invalid

End Function

Function snkTurn(app, newdx, newdy)

    if newdx=m.dx and newdy=m.dy then return false   ' already heading this way

    tongue_x=m.tongue.GetX()+newdx*app.cellwidth*2
    tongue_y=m.tongue.GetY()+newdy*app.cellheight*2

    head_x = tongue_x - newdx*app.cellwidth
    head_y = tongue_y - newdy*app.cellwidth

    corner_x = head_x - newdx*app.cellwidth
    corner_y = head_y - newdy*app.cellwidth

    prior_dx=m.dx
    prior_dy=m.dy

    m.dx=newdx
    m.dy=newdy

    newtongue=app.compositor.NewAnimatedSprite(tongue_x, tongue_y,
app.bitmapset.animations[m.DirectionName( newdx, newdy, "tongue-")] )


.
```

```
    newhead=app.compositor.NewSprite(head_x, head_y, app.bitmapset.regions[m.RegionName(
newdx, newdy, "head-")] )
    newcorner=m.tongue
    newbody=newcorner.GetData().previous

    newtongue.SetCollidableFlag(false)


     newcorner.SetData( {dx: newdx, dy: newdy, next: newhead,  previous:
newcorner.GetData().previous} )
       newhead.SetData( {dx: newdx, dy: newdy, next: newtongue,  previous: newcorner} )
     newtongue.SetData( {dx: newdx, dy: newdy, next: invalid,  previous: newhead} )

    m.tongue=newtongue

    if newhead.CheckCollision()<>invalid then return true

   ' fixup the last segment render
   ' (there is a tongue which turns into a corner, and a head which turns into body)

    newbody.SetRegion(app.bitmapset.regions[m.RegionName(newbody.GetData().dx,
newbody.GetData().dy, "body-")])

    if m.dy=-1 then ' turned north
        if prior_dx=-1 then ' was west-bound
            newcorner.SetRegion(app.bitmapset.regions[m.RegionName(0, -1, "corner-")])
        else
            newcorner.SetRegion(app.bitmapset.regions[m.RegionName(-1, 0, "corner-")])
        end if

    else if m.dy=1  ' turned south
        if prior_dx=-1 then ' was west-bound
            newcorner.SetRegion(app.bitmapset.regions[m.RegionName(1, 0, "corner-")])
        else
            newcorner.SetRegion(app.bitmapset.regions[m.RegionName(0, 1, "corner-")])
        end if

    else if m.dx=-1 then ' turned west / left
        if prior_dy=-1 then ' was north-bound
            newcorner.SetRegion(app.bitmapset.regions[m.RegionName(0, 1, "corner-")])
        else
            newcorner.SetRegion(app.bitmapset.regions[m.RegionName(-1, 0, "corner-")])
        end if
    else if m.dx=1  ' turned east / right
        if prior_dy=-1 then   ' was north-bound
            newcorner.SetRegion(app.bitmapset.regions[m.RegionName(1, 0, "corner-")])
        else
            newcorner.SetRegion(app.bitmapset.regions[m.RegionName(0, -1, "corner-")])
        end if
    end if

    app.TurnSound.Trigger(100)

    return false

End Function
```

sprite.small.map.xml

```
<DefenderBitmapSet>
    <ExtraInfo cellsize="40"/>

    <Bitmap name="Background" filespec="pkg:/snake_assets/snake.map_w-dirt.png" />

    <Bitmap name="game-over" filespec=" pkg:/snake_assets/snake.gameover.png" />

    <Bitmap name="title-screen" filespec=" pkg:/snake_assets/snake.title.3.png" />
```

.

```
<Bitmap name="snake" filespec=" pkg:/snake_assets/snake.body_sprite.png">

    <Region name="butt-North"          x="00" y="00" w="40" h="40" />
    <Region name="butt-East"           x="40" y="00" w="40" h="40" />
    <Region name="butt-South"          x="80" y="00" w="40" h="40" />
    <Region name="butt-West"           x="120" y="00" w="40" h="40" />

    <Region name="body-North"          x="00" y="40" w="40" h="40" />
    <Region name="body-East"           x="40" y="40" w="40" h="40" />
    <Region name="body-South"          x="80" y="40" w="40" h="40" />
    <Region name="body-West"           x="120" y="40" w="40" h="40" />

    <Region name="corner-North"        x="00" y="80" w="40" h="40" />
    <Region name="corner-East"         x="40" y="80" w="40" h="40" />
    <Region name="corner-South"        x="80" y="80" w="40" h="40" />
    <Region name="corner-West"         x="120" y="80" w="40" h="40" />

    <Region name="head-North"          x="00" y="120" w="40" h="40" />
    <Region name="head-East"           x="40" y="120" w="40" h="40" />
    <Region name="head-South"          x="80" y="120" w="40" h="40" />
    <Region name="head-West"           x="120" y="120" w="40" h="40" />

    <Region name="tongue-North"        x="00" y="160" w="40" h="40" t="10"/>
    <Region name="tongue-East"         x="40" y="160" w="40" h="40" t="10"/>
    <Region name="tongue-South"        x="80" y="160" w="40" h="40" t="10"/>
    <Region name="tongue-West"         x="120" y="160" w="40" h="40" t="10"/>
</Bitmap>

<Bitmap name="water_strip" filespec=" pkg:/snake_assets/snake.water_sprite.png">
    <Region name="a"        x="0" y="" w="40" h="40" t="25" />
    <Region name="b"        x="40" y="0" w="40" h="40" t="25" />
    <Region name="c"        x="80" y="0" w="40" h="40" t="25" />
    <Region name="d"        x="120" y="0" w="40" h="40" t="25" />
    <Region name="e"        x="160" y="0" w="40" h="40" t="25" />
    <Region name="f"        x="200" y="0" w="40" h="40" t="25" />
    <Region name="g"        x="240" y="0" w="40" h="40" t="25" />
    <Region name="h"        x="280" y="0" w="40" h="40" t="25" />
</Bitmap>

<Bitmap name="empty" shape="rect" color="0" w="40" h="40" t="40" />

<Animation name="water">
    <frame use="water_strip.a" />
    <frame use="water_strip.b" />
    <frame use="water_strip.c" />
    <frame use="water_strip.d" />
    <frame use="water_strip.e" />
    <frame use="water_strip.f" />
    <frame use="water_strip.g" />
    <frame use="water_strip.h" />
</Animation>

<Animation name="tongue-North">
    <frame use="snake.tongue-North" />
    <frame use="empty" />
</Animation>

<Animation name="tongue-East">
    <frame use="snake.tongue-East" />
    <frame use="empty" />
</Animation>

<Animation name="tongue-South">
    <frame use="snake.tongue-South" />
    <frame use="empty" />
</Animation>

<Animation name="tongue-West">
    <frame use="snake.tongue-West" />
    <frame use="empty" />
```

.

```
    </Animation>

</DefenderBitmapSet>
```

Appendix C - Reserved Words

END, LET, PRINT, IF, ELSE, ENDIF, THEN, GOTO, FOR, TO, STEP,
NEXT, RETURN, DIM, NOT, AND, OR, TAB, STOP, OBJFUN, TYPE,
TRUE, FALSE, CREATEOBJECT, WHILE, ENDWHILE, EXITWHILE, EACH, EXIT,
INVALID, ENDFUNCTION, ENDSUB, ELSEIF, REM, LINE_NUM, SUB, FUNCTION

.