GCSScript

4.0

GCSScript is a script language loosely based on the BASIC language. It has been optimized for the specific task as a Home Automation scripting language, as well as a CGI-style language that can be used as a call-out language by the home automation system to visit web pages and process the information contained -- such as news, weather, and stocks.

Error codes

Error Code	Meaning
0	Unsupported Keyword
1	Syntax
2	Unknown Data Type
3	Divide by 0
4	Line number already used
5	Variable undefined
6	Runtime data/parameter error

Comments

Comments can be written as single line comments denoted by the keyword REM.

Rest of line comments may be C++ Style (//)

Comment Blocks that may span more than one line are C Style (/* .... */)

Labels

GCSScript understands line numbers as labels for your code blocks. These labels can be used in GOTO and GOSUB statements. Labels are always integer numbers and must begin a line.

Data Types

Data Type	Range
Integer	–2,147,483,648 to 2,147,483,647
Float	1.7E +/- 308 (15 digits)
String	0 terminated string, maximum size is 64K

Note: Defining a string can include several escape characters. These include:

Code	Character
\n	Newline (NL / LF)
\r	Carriage Return (CR)
\t	Tab (HT)
\a	Alert (BEL)
\b	Backspace (BS)
\f	Formfeed (FF)
\v	Vertical Tab (VT)
\0	NULL, 0 terminator
\xXX	Hex code, where XX are the 2 digits
\'	Single Quote (')
\"	Double Quote (")
\\	Backslash

Comparisons and Operators

Operator	Meaning	Used in
<	Less than	Comparison
>	Greater than	Comparison
<=	Less than or equal to	Comparison
>=	Greater than or equal to	Comparison
==	Equal to	Comparison
<>	Not equal to	Comparison
AND	Logical AND	Comparison
OR	Logical OR	Comparison
=	Assignment	Assignment
+	Addition	Arithmetic
-	Subtraction	Arithmetic
*	Multiplication	Arithmetic
/	Division	Arithmetic
^	Power	Arithmetic
%	Modulus	Arithmetic (integer only)
&	Bitwise AND	Arithmetic (integer only)
|	Bitwise OR	Arithmetic (integer only)
<<	Bitwise shift left	Arithmetic (integer only)
>>	Bitwise shift right	Arithmetic (integer only)

Note: Strings are limited in functional operators. String comparisons are limited to equality and inequality. String arithmetic is limited to addition.

Type Conversion

Variables in GCSScript are variants. This means that, during execution, the data type will be converted to ensure proper functionality. For example:

```
x = 5.5
y = "The variant x has the value of "
z = y + x     //z will result in a string of "The variant x has the value of 5.5"
PRINT z
```

However, you may encounter times where you need control over the variant type. Three keywords have been added to the language to facilitate this:

MakeInt *var*

Convert the variant *var* to an integer value. If the type is already integer, not conversion is performed.

MakeFloat *var*

Convert the variant *var* to a floating point value. If the type is already float, not conversion is performed.

Make$ *var*

Convert the variant *var* to a string value. If the type is already string, not conversion is performed.

CharToInt *var*

Convert the first character of string variant *var* to an integer. For instance, if the first character was a tab (\t) then the variant would be modified to be integer and the value would be 9.

IntToChar *var*

Convert the integer variant *var* to a one character string containing the ASCII character that that integer represented. For instance, if the integer value is 9, the variant would be modified to string and contain the tab (\t) character.

Programmatic Flow Keywords

IF ... THEN ... ELSE ... ENDIF

The foundation of the BASIC language is comparisons. This is facilitated through the IF .. THEN ... ENDIF clause where ENDIF terminates the clause and code execution jumps to the line following ENDIF, should the condition evaluate to false..

Typically written as:

```
IF comparison THEN
  .. several lines of code if true
ENDIF
normal code execution here
```

A more powerful version includes the ELSE clause where code between the THEN and the ELSE is executed if the comparison evaluates to true, and the code between the ELSE and ENDIF is executed if the comparison evaluates to false.

```
IF comparison THEN
  .. several lines of code if true
ELSE
  .. several lines of code if false
ENDIF
normal code execution here
```

FOR *var* = *start* TO *end* ... NEXT

The FOR ... TO ... NEXT statement allows for loops in the code. The variable *var* is assigned the start value and the code between the FOR and NEXT is executed once for each time *var* is less than or equal to end. *var* is incremented each iteration.

```
FOR var = start TO end
  .. several lines of code
```

```
NEXT
normal code execution here
```

Goto *label*

Passes code execution to the line following *label.*

```
Goto 10
  .. several lines of code
10
normal code execution here
```

GoSub *label ...* **Return**

Passes code execution to the line following *label.* Execution is returned to the line following the GoSub upon reaching a Return keyword.

```
GoSub 10
normal code execution here
End

10
  .. several lines of code
Return
```

End

Terminate execution of the script. Processing is complete.

Sleep *milliseconds*

Cause script execution to pause for a specific amount of time, in milliseonds.

Input and Output

Depending on your execution environment, the methods below may not be implemented or provide different results. GCSScript may be used as a stand-alone CGI-style scripting application, or internally as a macro language for home automation.

Print *expression, expression, expression*

Used as a debugging tool, Print converts each expression to string, concatenates them and displays the result in a standard Windows MessageBox. Execution will halt until the OK button is pressed. Any processes waiting for completion will also halt.

Log *expression, expression, expression*

Log converts each expression to string, concatenates and time stamps them and writes the result to the log file.

In standalone mode, the log file is created in the same directory. The filename is the same as the script, with .LOG as the extension.

In macro mode, the result is written to the Doberman_Macro.log file.

Output *expression, expression, expression*

Output converts each expression to string, concatenates and writes the result to STDOUT (console) in standalone mode. Results from Output are identical to results from a CGI script, allowing GCSScript to be used to write CGI components.

In macro mode, this keyword is not implemented.

Input$ *var*

Reads from STDIN and returns the result in the variant *var* in string format. Results from Input are identical to post data for a CGI script, allowing GCSScript to be used to write CGI components.

In macro mode, this keyword is not implemented.

String manipulation

String keywords modify the variant string passed in and do not return a value. If you need the original string, be sure to copy it to another variable before using these keywords.

Left$ *var, length*

Truncates the variant *var* to the first *length* characters.

Right$ *var, length*

Modifies the variant *var* to the last *length* characters.

Mid$ *var, start, length*

Modifies the variant *var* to the string of *length* beginning at the *start* character. *Start* is 1 based.

Upper$ *var*

Convert the variant *var* to all upper case.

Lower$ *var*

Convert the variant *var* to all lower case.

Length$ *string, var*

Calculate the length of *string* and return in variant *var*.

Find$ *string, substring, var*

Locate *substring* within *string,* beginning at *var* and return the position in the variant *var.* Position is 1 based. *var* is modified.

FindPast$ *string, substring, var*

Locate *substring* within *string,* beginning at *var* and return the position of the first character past *substring* in the variant *var.* Position is 1 based. *var* is modified.

FindReplace$ *var, substring, replacestring*

Locate all occurrences of *substring* within string variant *var*, and replace with *substring. var* is modified.

ExtractUntil$ *var, start, string, endstring*

Extract a *substring* from *string*, beginning at position *start* and ending at -- but not including -- *endstring*. *var* is modified. start is 1 based.

```
// Extract a hyperlink.
reference = "Get <a href =
\"www.goldencrater.com\">Cool
Software</a>"
FindPast$ reference, "href = \"", start
ExtractUntil$ link, start, reference, "\">"
// link now contains www.goldencrater.com
```

StripHTML$ *var*

Removes html formatting from *var*. Converts
 <p>   > and other codes into ASCII equivalent. Removes all other tags, including hyperlinks. Result is a text only version of the original string variant *var*.

```
reference = "Get <a href =
\"www.goldencrater.com\">Cool
Software</a>"
StripHTML$ reference
// reference now contains Get Cool Software
```

HTMLFromText$ *var*

Escapes the string variant *var* for use as part of an HTML page. all carriage returns are converted the
, spaces converted to   etc.

TextFromHTML$ *var*

See **StripHTML$**

PrepXML$ *var*

Removes formatting (tabs, multiple spaces, CR, LF), from *var*. Converts
 <p>   > and other codes into ASCII equivalent. All tags left intact.

File Handling

FileLoad$ *var, filename*

Loads file referenced by *filename* into the string variant *var*. If file does not exist, *var* is empty. You may check the file existence with **FileExists**.

FileSave$ *var, filesname*

Appends contents of string variant *var* to file referenced by *filename*. If file does not exist, it will be created. If you wish to overwrite the file, delete the file first with **FileDelete**.

FileDelete *filename*

Deletes the file referenced by *filename.*

FileExists *filename, var*

Returns an integer value in *var* that indicates file *filename* existence and accessibility. Upon return *var*

Value	Meaning
0	Does not exist
1	Exists but no read or write access
2	Write access only
4	Read access only
6	Both read or write access

Internet Enabled Keywords

FileHTTPGet *filename, url, name, password*

Perform an HTTP GET on *url* with the result being written to the file referenced by *filename.* File contents are overridden. *name* and *password* are used to login and may be empty ("")

```
FileHTTPGet "c:\temp\download.gif" ,
"www.goldencrater.com/image.gif", "", ""
```

HTTPGet$ *var, url, name, password*

Perform an HTTP GET on *url,* loading the result into string variant *var*. *name* and *password* are used to login and may be empty ("")

```
HTTPGet$ page ,
"www.goldencrater.com/index.htm", "", ""
// page now contains the contents of
index.html
```

HTTPPost$ *var, url, name, password, postdata*

Perform an HTTP POST on *url* and posting the supplied *postdata.* The result is loaded into string variant *var*. *name* and *password* are used to login and may be empty ("")

```
HTTPPost$ page ,
"www.somesite.web/notreal.cgi", "", "",
"request data"
```

WebRawRequest$ *var, url, port, header, data*

Opens a socket to *url* on *port* and proceeds to send *header*, two carriage return/line feed combinations, followed by *data.* The socket is then read for the full response. The header is stripped from the response and the rest is set into the string variant *var*.

If header does not include Content-Length, it will be calculated and added to the header.

This keyword allows for invoking SOAP-style requests.

```
// Sample SOAP request to get the weather for Toronto.
// Please note lines are wrapped to fit screen.

// Header
posth = "POST /ccx/GlobalWeather HTTP/1.1\r\nSOAPAction:
\"capeconnect:GlobalWeather:GlobalWeather#getWeatherReport\"\r\nContent-
Type: text/xml; charset=UTF-8"

// Post Data
postd = "<?xml version=\"1.0\" encoding=\"UTF-8\"?><soap:Envelope
xmlns:n=\"capeconnect:GlobalWeather:GlobalWeather\"
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:soapenc=\"http://schemas.xmlsoap.org/soap/encoding/\"
xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"> <soap:Body
soap:encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\">
<n:getWeatherReport> <code xsi:type=\"xs:string\">cyyz</code>
</n:getWeatherReport> </soap:Body> </soap:Envelope>"

// Content-Length will automatically be added to header, since not included.
httprawrequest$ f "live.capescience.com" 80 posth postd
```

SendEMail address, subject, body, attachment

Send an email to *address* using *subject* and *body*. *attachment* can be empty or filename to attach. Email is always sent as text only, html or RTF in the *body* often will not be recognized as such and rendered by the receiving email application as plain text.

Time Manipulation

GetTime *var*

Get the current time, in seconds and place in the integer variant *var*. Time is counted as seconds past midnight, January 1, 1970.

TimeMake *var, years, months, days, hours, minutes, seconds*

Create an integer based time the corresponding passed-in values.

TimeBreak *time, years, months, days, hours, minutes, seconds, dow*

Given *time*, will fill in the integer variants to their corresponding values. *dow* will be of value 0 (Sunday) to 7 (Saturday).

Automation Device query and control

Note: zone and units are 1 based. In X10, A1 would be 1,1. Thermostat base node is 1, zone 1.

GetDevice *address, protocol, zone, unit, var*

Given string variant *address* (or empty for "LocalDbrServer",) the *protocol*, the *zone* and *unit*, the variant *var* will be filled with the current state of that device.

zone, *unit* are 1 based.

In standalone mode, this keyword is not implemented.

GetDeviceByName devicename, *var*

Given string variant *devicename,* the variant *var* will be filled with the current state of that device.

In standalone mode, this keyword is not implemented.

SetDevice *address, protocol, zone, unit, value*

Given string variant *address* (or empty for "LocalDbrServer",) the *protocol*, the *zone* and *unit*, the device state will be set to *value*.

zone, unit are 1 based.

In standalone mode, this keyword is not implemented.

SetDeviceByName *devicename, value*

Given string variant devicename, the device state will be set to *value*.

In standalone mode, this keyword is not implemented.

GetHVAC *address, protocol, zone, param, var*

Given string variant *address* (or empty for "LocalDbrServer",) the *protocol*, and the *zone*, the variant *var* will be filled with the current state of the HVAC parameter *param*. *zone* is 1 based.

In standalone mode, this keyword is not implemented.

Parameter	Index
Heat Setpoint (stage 0)	0
Heat Setpoint (stage 1)	1
Heat Setpoint (stage 2)	2
Heat Setpoint (stage 3)	3
Cool Setpoint (stage 0)	4
Cool Setpoint (stage 1)	5
Cool Setpoint (stage 2)	6
Cool Setpoint (stage 3)	7
Current Temperature	8
Humidity Setpoint	9
Current Humidity	10

Fan	11
Damper	12
Setback	13
Mode	14

Mode: 0 - Off, 1 - Auto, 2 - Heat, 3 - Cool

GetHVACByName *devicename, var*

Given string variant *devicename,* the variant *var* will be filled with the current state of the HVAC parameter *param*.

In standalone mode, this keyword is not implemented.

SetHVAC *address, protocol, zone, param, value*

Given string variant *address* (or empty for "LocalDbrServer",) the *protocol*, and the *zone,* the HVAC parameter *param* will be set to *value*. *zone* is 1 based.

Sending a string *value* will set that text to be displayed on the thermostat, if equipped.

In standalone mode, this keyword is not implemented.

SetHVACByName *devicename, value*

Given string variant *devicename,* the HVAC parameter *param* will be set to *value*.

Sending a string *value* will set that text to be displayed on the thermostat, if equipped.

In standalone mode, this keyword is not implemented.

Miscellaneous

PlaySound *filename*

Plays the WAV sound referenced by *filename.* If *filename* is empty and a sound is playing, the playback will be stopped.

Returns immediately.

LoopSound *filename*

Repeatedly plays the WAV sound referenced by *filename.* If *filename* is empty and a sound is playing, the playback will be stopped.

Returns immediately.

ShellExec *filename, parameters, executionpath*

Call the registered shell "Open" method on *filename*, passing in *parameters*. Execution will take place in *executionpath.*

Returns immediately.

RestartServer

Causes the epAssist or Doberman Server to shutdown and restart. The hardware/operating system does not restart. This is useful during remote configuration changes.

ToCelsius *value*

Converts *value* from Fahrenheit to Celsius.

ToFahrenheit *value*

Converts *value* from Celsius to Fahrenheit.

Pre-created variables

In Standalone mode, several variables are pre-defined in order to assist in building CGI and other scripts.

CGIQuery = the contents of the environment variable **QUERY_STRING**

CGIIP = the contents of the environment variable **REMOTE_ADDR**

ArgA = the first command line value passed in to program execution beyond the script filename (argv[2])

ArgB = the second command line value passed in to program execution beyond the script filename (argv[3])

ArgC = the third command line value passed in to program execution beyond the script filename (argv[4])

HTTPRawHeader = the header returned by the call to **HTTPRawRequest$**

In Macro mode, several variables are pre-defined in order to assist in building CGI and other scripts.

HTTPRawHeader = the header returned by the call to **HTTPRawRequest$**