Embedded Computers

For professionals

FlashcatUSB

SCRIPT ENGINE DOCUMENTATION

Website: www.embeddedcomputers.net
Support email: contact@embeddedcomputers.net
Last updated: February, 2024

FLASHCAT

QUICK LOOKUP

GETTING STARTED

Before you begin, please make sure you are using the newest software/firmware for your device. You can download the newest software from the EmbeddedComputer website.

The software includes many scripts. Advice for beginners, just look at some of those to get an idea of how to write/use them.

A script file is just a plain text document. You can open and edit one using Notepad. The contents of a script file are made up of commands, labels, and sub procedures (called events). When a script file is executed, any line that is not in an event block will be executed. The purpose of a script is to accomplish more complex operations that may not otherwise be performed using the standard GUI. Script files are also ideal for production environments.

Basic Syntax

FlashcatUSB scripts use a very familiar console/script syntax. Each line contains a statement that can be a command to execute, a condition (IF etc.), a flow control statement, or the start of an Event (a function or sub procedure). To put comments into your script file (statements which have no affect), use the # symbol. Any characters proceeding will not be evaluated by the script engine.

The basic flow-control statements are: LABEL, GOTO, EXIT, RETURN.

The GOTO statement can be used to control which line to execute. Generally, the software starts executing at the top of the file and proceeds down. To change the location, you can use the GOTO statement followed by a location that is indicated by using a label. A LABEL is a user-specified word that ends with a colon. For example:

```
var1 = 10
GOTO SKIP_MINVAL
var1 = var1 - 5
SKIP_MINVAL:
Print(var1)              #var1 = 10
```

So as the script executes, when it reaches the GOTO keyword, the engine will then search the document (forwards and backwards), for a label that contains the name “SKIP_MINVAL”. Once found, execution will begin there. Because the script engine will search both ways, you can also use GOTO keywords to create loops.

The EXIT keyword can be used to leave an event or function, or to exit out of a condition segment (IF or FOR block for example). The syntax usage is EXIT WHILE, EXIT FOR, EXIT IF or EXIT SCRIPT. If you run exit script, no matter what event you are in, the entire script will stop executing. For example:

```
If (VarInt = 10)         #This does a compare to see if VarInt is 10
   SomeFunction()
   Var2 = 0x40
   EXIT SCRIPT
   Var3 = 0x40           #This will not be executed
End
```

When the script reaches the exit command, it will then exit out of this IF statement and proceed to execute the next line after the EndIf statement.

Data Types

1. **Boolean**

   A value that is either true or false.

```
var_bool = true        #create a variable and assigns it the data type of bool with a value of true.
verify(false)          #passes the value of false to a function.
```

2. **String**

   A value that consists of readable ASCII characters. To create this type, put quotes around the value.

```
your_name = “Taki Tachibana”        #creates a variable with a string data type
```

3. **Integer**

   A value that represents a 32-bit signed value. By default, all numbers are parsed into an Integer data type. This data type can also contain negative numbers.

4. **UInteger**

   A value that represents a 32-bit unsigned value. There are two methods to create this data type, first, you can append a number with a capital U, or you can use a base-16 hexadecimal string that is 32-bits or less.

```
counter_var = 128U          #creates a variable with 128 value that is unsigned.
counter_var = 0x80          #does the same as above.
```

5. **Data Array**

   This data type represents an array of bytes. Usually, this data type is created or used by an internal function, but it is also possible to create and define it by using hexadecimal pairs separated by semicolons.

```
d_var = 0x80;0x81;0x82;0x83          #creates a binary data variable with 4 bytes.
d_var = memory.read(0x5000,256)      #reads 256 bytes from memory offset 0x5000.
```

   Data arrays can also be partially accessed or spliced; syntax: data_array(index, count)

```
two_bytes = d_var(0,2)          #returns 0x80;0x81;
single_byte = d_var(3)          #returns 0x82;
```

Arithmetic Operators

The following operators are supported:

+ Addition. Adds two integers, combines two strings, or combines two data arrays.

- Subtract. Only compatible with Integer data types.

/	Divide. Only compatible with Integer data types.
*	Multiple. Only compatible with Integer data types.
<<	Shift left. Shits an integer to the left by a number of bits.
>>	Shift right. Shifts an integer to the right by a number of bits.

Logical Operators

==	compares two operands and returns BOOL TRUE if they are exact.
&	logical AND
|	logical OR

Variables

A variable is a name that you assign an object. You can assign a string, data, integers, or boolean values. For example:

```
ThisVar = "Hello World"
```

Will now create a variable named ThisVar whose string value is "Hello World". To create a data array use ";" after each byte:

```
MyData = 0x01;0x02;0x03;0x04;0x05;0x06;
```

If you assign a variable 4 or less bytes, the variable will auto convert to a Integer type instead of a Data type. To create a boolean variable:

```
DoVar = true
```

And to create an integer:

```
VarInt = 470
```

Integer variables are able to be added or subtracted. String and Data variables can be combined.

```
VarInt = 5
VarInt += 10
msgbox(VarInt)        #this will produce the result of 15
```

For strings and data, use the operand "&", for example:

```
VarStr = "Hello "
VarStr = VarStr + "World!"
msgbox(VarStr)        #Will produce "Hello World!"
```

```
MyData = 0x01;0x02;0x03;0x04;0x05;
MyData = MyData + 0x06;0x07;
msgbox(hex(MyData))     #Will produce "0x01020304050607"
```

The hex command converts the data array into a hex string that can be printed.

DATA arrays can also have index arguments specified to read data from the array and use it like an Integer that contains a byte.

```
VAR1 = 0x01;0x02;0x03;
VAR2 = VAR1(1)
Print(VAR2)            #Will print “2”
```

To write a byte to a DATA array, you can also do this:

```
VAR1 = 0x01;0x02;0x03;
VAR1(0) = 0xFF
Print(String.Hex(Data.ToInt(VAR1(0)))) #Outputs “0xFF”
```

Events (Functions)

An Event is similar to a function that you may be familiar with. You can treat it like a “function”, you can create events and then call them like functions, and even return a value. Events can also be assigned to GUI elements, such as buttons. So, when you click a button you created, the engine will run the commands that are in the Event that you assigned to that button.

Events are very useful. You can pass variables to events and retrieve values from events. When you pass a variable or value to an event, the event will create a new variable for each argument passed. These new variables will be named $1, $2, $3, and so on for each variable passed.

Within an Event block, you can exit the block and create a new variable using the **RETURN** keyword.

To create an event or function, use the CreateEvent keyword followed by a parameter specifying the name of the event/function. And to specify the end of the Event, use the EndEvent keyword.

```
EchoToMsgBox("Hello World")
CreateEvent(EchoToMsgBox)
        msgbox($1)
EndEvent
```

This code sample popup a message box saying "Hello World" when executed. You can also use events like functions to parse information an use the event like you would a command function. For example:

```
msgbox(CombineString("Hello"," World"))
CreateEvent(CombineString)
        StrVar = $1 + $2
        Return StrVar
End
```

The output from this will produce a message box that says “Hello World”.

Conditionals

To execute code based on a condition, you can use the IF keyword followed by a statement that can be evaluated to be either true or false. The syntax is IF (STATEMENT). Optionally, you can prefix the statement with the NOT keyword to indicate that the statement should evaluate to the opposite. This is similar to the “IF, ELSE, ENDIF” of other programming languages.

```
If (5 > 2)
        msgbox("This will be executed")
Else
        msgbox("This will not")
End
```

The condition statement (5 > 2) is evaluated and found to be true. You can also use Events that return TRUE or FALSE. If you precede the condition statement with the "not" keyword, whatever the statement is evaluated at, the opposite will happen. You can also use the "!" character for the same effect.

```
If not (GetValue() > 10)
        print ("This will be executed")
End
CreateEvent(GetValue)
        retVar = 5
        return retVar
EndEvent
```

In the above example, you can create a function named GetValue, by specifying it using the CreateEvent keyword. Inside the event block, you can then run commands or other syntax and then use

the Return keyword to return a value to the calling line, in this case, the IF statement that compares the return value to be greater than 10.

You can also use an **IF CONDITION** with the keyword **NOTHING** to check to see if a variable exists. For example

```
If not (ObjectVar==Nothing)
        print("variable exists! ")
Else
        print("Variable does not exist! ")
End
```

Loops (Iterations)

To do repetitive tasks, you can use a FOR loop. This is specified by using the FOR keyword followed by parameters specifying a variable name, starting value, and ending value. Optionally, you can specify a step value (an integer to increase the counter by).

```
For (i = 0 to 9)
        print("We are on loop number: " + Int.ToStr(i))
End
```

This will iterate 50 times (0, 2, 4, 6, ...):

```
For (i = 0 to 98, 2)
        print("We are on loop number: " + Int.ToStr(i))
End
```

A WHILE loop will execute statements in the body while the condition is TRUE. You can also use the NOT modifier to evaluate a statement that is True, until it becomes False.

```
count_down = 10
While (count_down>0)
        print("Current value: " + Int.ToStr(count_down))
        count_down = count_down - 1
End
```

You can even use nested loops like this:

```
For (x = 0 to 9)
        For (y = 0 to 9, 2)
                Print("x,y = " + Int.ToStr(x) + "," + Int.ToStr(y))
        End
End
```

Error Handling

The console application will set the %ERRORLEVEL% environment to -1 if there was an error or to 0 if there was no error. You can then check this variable at the command shell or from a batch file.

```
C:\Users\Admin\Documents\EmbeddedComputers\Source Code\Console\bin\Debug\net5.0>fcusb_console -detect -exit
Welcome to the FlashcatUSB interfacing software, build: 636
Copyright 2021 - Embedded Computers LLC
Running on: Windows (64 bit)
FlashcatUSB Script Engine build: 312
License status: non-commercial use only
MODE not specified (using OPERATION setting instead)
Performing memory Auto-Detect
Waiting to connect to FlashcatUSB
Successfully connected to FlashcatUSB over USB
Connected to FlashcatUSB Pro (PCB 5.x), firmware version: 1.15
Detecting connected Flash device...
Initializing SPI device mode
Attempting to detect SPI device (auto-detect mode)
Successfully opened device in SPI mode
Connected to SPI Flash (RDID:0xEF7021 REMS:0xEF20)
Flash detected: Winbond W25Q01JV (134,217,728 bytes)
Page erase size: 65,536 bytes
Detected SPI Flash on high-speed SPI port
Setting SPI clock to: 8 Mhz
-----------------------------------------------
Application completed
Disconnected from FlashcatUSB Pro device

C:\Users\Admin\Documents\EmbeddedComputers\Source Code\Console\bin\Debug\net5.0>echo %ERRORLEVEL%
0
```

At the script level, you might do your own error checking and then use the EXIT SCRIPT, but since this is recognized as a success, you might also want to indicate an error. In this circumstance, you can set a reserved variable named ERROR to True. For example:

```
IF NOT memory.name=="Winbond W25Q128JV"
        print("ERROR: W25Q128JV DEVICE NOT CONNECTED")
        ERROR = True
```

```
        EXIT SCRIPT
END
```

This small piece of code checks to see if the current memory detected matches a specific device and if not, it will print an error message to the console, set the ERROR variable to true, and then the EXIT SCRIPT command to stop executing. And then when the program quits, the %ERRORLEVEL% will automatically be set to -1.

LIST OF CONSOLE OR SCRIPT COMMANDS

The following is a complete list of commands that are built into the FlashcatUSB script engine. You can execute these either in a script file or from the software's console window. Some commands will output information to the console, others will not. Also note that for the memory commands, if you have initiated more than one memory device, you can access each device by using parameters with an index, for example, memory(0).read will perform the read operation from the first memory device; memory(1).read will do the same from the second device, and so on.

STRING Commands

Command:	string.upper
Parameters:	String
Returns:	String
Description:	Inputs a string and converts it all uppercase.
Examples:	string.upper(“hello”) #ouputs "HELLO”

Command:	string.lower
Parameters:	String
Returns:	String
Description:	Inputs a string and converts it to all lower case.
Examples:	string.lower(“ANIME”) #ouputs "anime"

Command:	string.hex
Parameters:	Integer
Returns:	String
Description:	Converts an Integer value into a hex string
Examples:	string.hex(255) #ouputs "0xFF"

Command:	string.length
Parameters:	String
Returns:	Integer
Description:	Returns the number of bytes the string requires in memory.
Examples:	string.length(“zenzenzense”) #returns 11

Command:	string.toint
Parameters:	String
Returns:	Integer
Description:	Converts a string that contains a numeric value to an integer.
Examples:	var_int = string.toint(“1024”)

Command:	string.fromint
Parameters:	Integer
Returns:	String
Description:	Converts an integer value or variable to a string.
Examples:	var_str = string.fromint(1024)

Command:	string.todata
Parameters:	String
Returns:	Data
Description:	Converts a string into a char array and returns data variable.

INT / UINT Commands

Note: use UINT instead when dealing with UINTEGER data types.

Command:	INT.ToStr
Parameters:	Integer or UInteger
Returns:	String
Description:	Converts an integer to a string
Examples:	value = 3 Print(“value is “ + Int.ToStr(value)) #prints “value is 3” to the console

Command:	INT.ToData32
Parameters:	Integer or UInteger
Returns:	Data
Description:	Converts an integer to a 4-byte data array

Examples:	value = 3 d_arr = INT.ToData32(value) #d_arr = 0x00;0x00;0x00;0x03;

Command:	INT.ToData16
Parameters:	Integer or UInteger
Returns:	Data
Description:	Converts an integer to a 2-byte data array
Examples:	value = 3 d_arr = INT.ToData32(value) #d_arr = 0x00;0x03;

Command:	INT.ToData
Parameters:	Integer or UInteger
Returns:	Data
Description:	Converts an integer to a data array (1 to 4 bytes)
Examples:	value = 3 d_arr = INT.ToData32(value) #d_arr = 0x03;

Command:	INT.Min
Parameters:	Integer or UInteger
Returns:	Integer or UInteger
Description:	Inputs two values and returns the smallest value of the two.
Examples:	min_value = int.min(5,10) #returns 5

DATA Commands

Command:	Data.New
Parameters:	Integer, Data (optional)
Returns:	Data
Description:	Creates a new data array with a specific number of bytes. The first parameter specifies the size of the array. The second parameter will set the initial data to create.

Examples:	DVAR = Data.new(128,0xFF;) #Creates an array with 128 bytes
	DVAR = Data.new(512,0xAA;55) #Alternating bytes

Command:	Data.FromHex
Parameters:	String
Returns:	Data
Description:	Converts a hex string to a data array. Data can be with or without “0x” prefix.
Examples:	Result = Data.FromHex(“FFFFFFFF”)

Command:	Data.compare
Parameters:	Data, Data
Returns:	Bool
Description:	Compares two data arrays and returns true if they both exist and are the same.
Examples:	result = Data.Compare(VAR1,VAR2)

Command:	Data.length
Parameters:	Data
Returns:	Integer
Description:	Returns the size of the data array.
Examples:	size = Data.length(VAR1)

Command:	Data.resize
Parameters:	Data, Integer, Integer (optional)
Description:	Resizes a data array. First parameter is the data array to resize, the second is the offset within the array, and the last (optional) argument is the number of bytes to copy.
Examples:	Data.resize(var1,0,128)

Command:	Data.word
Parameters:	Data, Integer
Returns:	Integer
Description:	Copies a word (4 bytes) of data from a Binary Data at a specific offset and saves it as an Integer.
Examples:	var1 = data.word(data_array, 0)

Command:	Data.hword
Parameters:	Data, Integer
Returns:	Integer
Description:	Copies a half-word (2 bytes) of data from a Data array at a specific offset and saves it as an Integer.
Examples:	Var1 = data.hword(data_array, 0)

Command:	Data.ToStr
Parameters:	Data
Returns:	String
Description:	Converts the data array to hex string.
Examples:	Var1 = Data.ToStr(data_array)

Command:	Data.copy
Parameters:	Data, Integer, Integer (optional)
Returns:	Data
Description:	Copies a section of a Data array and creates a new array. The first parameter is the Data variable, the second is the offset within the array, and the third is the length of data to copy, otherwise it will copy until the end.
Examples:	Var1 = Data.Copy(data_array,0,128) #Copies 128 bytes to a new array

Command:	Data.ToInt
Parameters:	Data, Optional Integer
Returns:	Integer
Description:	Converts a single data element to an integer. Default uses index 0, but optional parameter can be used to specify offset within the data array.
Examples:	i = Data.ToInt(VAR1(0))

Command:	Data.ToUInt
Parameters:	Data
Returns:	UInteger
Description:	Converts a single data element to an unsigned integer. Default uses index 0, but optional parameter can be used to specify offset within the data array.
Examples:	i = Data.ToUInt(VAR1(0))

I/O commands

Command:	IO.open
Parameters:	String (optional), String (optional), String (optional)
Returns:	Data
Description:	Prompts the user for a file and then reads the file from disk and returns a data variable. First optional parameter is the title of the window and the optional second is the standard file filter to use. The third optional argument can specify a sub-directory to start in.
Examples:	MyData = IO.Open("Choose file", "Firmware files (*.bin)|*.bin")

Command:	IO.save
Parameters:	Data, String (optional), String (optional)
Syntax:	Data variable to write, title prompt, default save name
Description:	Prompts the user to save a data variable to the hard drive.
Examples:	IO.Save(MyData,"Where to save?","file_read.bin")

Command:	IO.read
Parameters:	String
Returns:	Data
Description:	Reads a file from the hard drive. The first parameter is the name of the file (in relation to where FlashcatUSB.exe is located).
Examples:	MyData = IO.Read(“Scripts\EEPROM.bin”)

Command:	IO.write
Parameters:	Data, String
Description:	Writes data to the hard drive. The first parameter is a data array variable, the second is the location where you want to save the file. This command does not prompt the user.
Examples:	IO.Write(MyData,“Scripts\EEPROM.bin”)

Command:	IO.delete
Parameters:	String
Returns:	Bool
Description:	Attempts to delete a file. Returns True if the file existed and was deleted.
Examples:	result = IO.Delete(“output.txt”)

Memory commands

Command:	memory(index).name
Parameters:	None
Returns:	String
Description:	Returns the name of the memory device.
Examples:	flash_name = Memory(0).Name

Command:	memory(index).size
Parameters:	None
Returns:	Integer
Description:	Returns the number of bytes in the given memory device. Note: due to the size limitation of the Integer data type. This function can only be called on memory devices that are 2GB (2,147,483,647) or smaller.
Examples:	flash_size = memory(0).size

Command:	memory(index).write
Parameters:	Data, Integer, Optional Integer

Syntax:	Data object to write, flash address offset, optional length
Description:	Writes a data variable to the flash device. Works for both CFI and SPI flash devices, but please note you must have already initiated the flash.
Examples:	memory.write(data1,0,256) #Writes Binary Data to memory

Command:	memory(index).read
Parameters:	UInteger, Integer, Optional Bool
Returns:	Data
Description:	Reads data from the flash device. First parameter is the flash offset, the second is number of bytes to read. The third option disables status updates.
Examples:	dataVar = memory.read(0,512) #Reads 512 bytes

Command:	memory(index).wait
Parameters:	None
Description:	Performs a wait operation (delay while busy) on the memory device. If the memory device has specific function to check a busy status, that operation is used. For example, a SPI device will check the status register-1 for the busy bit. A CFI parallel device will check the RDBY pin or preconfigured register.
Examples:	memory.wait

Command:	memory(index).readstring
Parameters:	UInteger
Returns:	String
Description:	Reads a string from the offset specified on the flash device. Returns nothing if error or string not found.
Examples:	dataStr = memory.readstring(0x5000)

Command:	memory(index).readverify
Parameters:	UInteger, Integer
Returns:	Data
Description:	Similar to read(), this function actually does it twice and compares the result, and if needed verifies all data to ensure that the data read is 100% accurate. Returns nothing if verification failed. This function is preferred over read() where the integrity of the data is vital.
Examples:	dataVar = memory.readverify(0,512) #Reads 512 bytes

Command:	Memory(index).sectorcount
Returns:	Integer
Description:	Returns the number of sectors (sometimes called blocks) of a memory device.

Examples:	sectors = memory.sectorcount()

Command:	memory(index).sectorsize
Returns:	Integer
Description:	Returns the number of bytes of a given sector. Some devices have different sector sizes, while others have uniform sizes.
Examples:	sector_size = memory.sectorsize(0)

Command:	memory(index).erasesector
Parameters:	Integer
Returns:	None
Description:	Erases the specified flash sector.
Examples:	Memory(index).EraseSector(0) #Erases the first sector

Command:	memory(index).erasebulk
Parameters:	None
Returns:	Nothing
Description:	Erases the entire flash memory
Examples:	Memory(index).erasebulk()

Command:	memory(index).exist
Parameters:	None
Returns:	Bool
Description:	Returns true if a memory device at a given index has been created.
Examples:	Memory(2).exist()

NAND commands

Command:	nand(index).writepage
Parameters:	Data, Integer, Integer (Optional), Integer (Optional)
Returns:	Nothing
Description:	Performs a low-level NAND memory write operation. The first parameter is the data array, and this should be small enough to fit into the page, spare area, or both. Second is the page index. Third argument is the memory area. Lastly, the fourth argument can be used to specify an offset within the page. Use the following for the memory area: **Memory Area:** 0 = Main page (2048 bytes for example)

	1 = Spare page (64 bytes for example) 2 = All (2112 bytes for example)
Examples:	MAIN_AREA = 0 nand(0).writepage(Buffer, 0, MAIN_AREA)

Command:	nand(index).GetFeature
Parameters:	Integer
Returns:	Integer
Description:	Performs the GetFeature operation which is supported by some NAND devices. The address is generally one byte and the value returned is one byte (as an integer).
Examples:	print(NAND.GetFeature(0x89)) #Returns the Read Retry value

Command:	nand(index).SetFeature
Parameters:	Integer, Integer
Returns:	Nothing
Description:	Writes the value of the Feature table. The first argument is the one byte feature address, and the second value is the byte to write into the register.
Examples:	NAND.SetFeature(0x89,2) #Sets the Read Retry value to 2

TAB commands

Command:	tab.create
Parameters:	String
Returns:	Integer
Description:	Creates a application specific tab. Returns the index of the tab.
Examples:	tab.create("My Device")

Command:	tab.addgroup
Parameters:	String, Integer, Integer, Integer, Integer
Syntax:	Name of group, (X-axis), (Y-axis), Length, Height
Description:	Creates a group box on the tab.
Examples:	tab.addgroup("Feature",10,10,420,140)

Command:	tab.addbox
Parameters:	String, String, Integer, Integer

Description:	Creates a input box on your tab.
Examples:	tab.addbox("BXNAME","default text",30,110)

Command:	tab.addtext
Parameters:	String, String, Integer, Integer
Description:	Creates a text label on your tab.
Examples:	tab.addtext("txtName","What to say",30,110)

Command:	tab.addImage
Parameters:	String, String, Integer, Integer
Description:	Adds a image to your tab from the specified file (in your scripts folder)
Examples:	tab.addimage("ImgName","logo.gif",20,20)

Command:	tab.addButton
Parameters:	Event, String, Integer, Integer
Description:	Adds a button to your tab. The specified event is called when the user clicks on the button.
Examples:	Tab.addbutton(HelloWorld,"Click Me!",20,20)

Command:	tab.addprogress
Parameters:	Integer, Integer, Integer
Description:	Adds a progress bar to your form. This bar will then be automatically updated via internal functions that you call (selected ones that might take time to process). The parameters are x-axis, y-axis, and bar width.
Examples:	Tab.addprogress(20,92,404)

Command:	tab.remove
Parameters:	String
Description:	Removes any previously added object from your tab.
Examples:	Tab.remove("ImgName")

Command:	tab.settext
Parameters:	String, String
Description:	Changes the text of any previously created object
Examples:	tab.settext("txtName","hello world")

Command:	tab.gettext
Parameters:	String
Description:	Gets the text property of a previously created object.

Examples:	res_str = Tab.GetText("txtName") #Returns "hello world"

Command:	tab.buttondisable
Parameters:	String (Optional)
Description:	Disables a button so the user can not click it and run the event. The input is the name of the button, but if omitted, then all user created buttons will be affected.
Examples:	Tab.buttondisable("btName")

Command:	tab.buttonenable
Parameters:	String (Optional)
Description:	Enables the button (if you had it disabled). The input is the name of the button, but if omitted, then all user created buttons will be affected.
Examples:	Tab.ButtonEnable("btName")

SPI commands

Command:	SPI.Clock
Parameters:	Integer
Description:	Used to set the hardware SPI clock (in MHZ). For Classic, compatible speeds are 8, 4, 2, and 1. For Professional they are 5, 8, 10, 12, 15, 20, 24, and 30.
Examples:	SPI.Clock(8)

Command:	SPI.Mode
Parameters:	Integer
Description:	Used to set the SPI device mode. Supported modes 0, 1, 2, 3.
Examples:	SPI.Mode(0)

Command:	SPI.Database
Parameters:	Boolean
Description:	Prints the entire list of supported SPI devices and size in bits. Optional parameter to also display the JEDEC ID.
Examples:	SPI.Database(true)

Command:	SPI.GetSR
Parameters:	Integer (optional)
Returns:	Data
Description:	Prints and returns the value of the status register. Optional parameter can set the number of bytes to read (default is 1).
Examples:	SPI.GetSR(1)

Command:	SPI.SetSR
Parameters:	Data
Description:	Writes data to the status register. You can write a single byte, or multiple bytes.
Examples:	SPI.SetSR(0xF0)

Command:	SPI.WriteRead
Parameters:	Data, Integer (optional)
Returns:	Data
Description:	Writes a series of bytes to the SPI bus and returns the exact number of bytes read back. Using this command, you can execute any specific SPI op code. Writes data to the SPI bus and then optionally allows you to read data back. The first parameter is the data to write, it can be a hex string or data variable. The second parameter (which is optional) is the number of bytes to read back. Using this command you can execute any specific SPI op code. The example shows you how to read the SPI Flash's Device ID.
Examples:	chip_id = SPI.WriteRead(0x9F;,3) print(chip_id)

JTAG commands

Command:	JTAG.idcode
Returns	UInteger
Description:	Returns the current IDCODE from the current selected JTAG device. Returns 0 if no JTAG device has been detected.
Examples:	dev_id = JTAG.idcode()

Command:	JTAG.config
Parameters:	String
Description:	Configures the JTAG library to use processor specific instructions for the memory access.
Settings:	“MIPS” – Will load EJTAG extensions “ARM” – Will load ARM extensions
Examples:	JTAG.mode(“MIPS”) #For EJTAG devices

Command:	JTAG.select
Parameters:	integer
Description:	In a multi-device JTAG chain, you can select a specific device on the chain in which to communicate with. Or you can select the last index for all devices.

Examples:	JTAG.select(0) #Select the first device in a multi device chain

Command:	JTAG.print
Description:	Will display information about the current JTAG chain, including number of devices, chain size in bits and each device with its ID CODE.
Examples:	JTAG.print #Will print all devices on the JTAG chain

Command:	JTAG.clear
Description:	This command will clear all devices from the JTAG chain. The purpose is to all you to other commands to rebuild the JTAG chain with custom parameters.
Examples:	JTAG.clear

Command:	JTAG.set
Parameters:	Integer, String
Description:	This command allows you to modify a device in a multi-device chain and specify its BSDL library name. This is useful for devices that do not support IDCODE command for autoconfiguration. Note: the BSDL file needs to be compiled within the software, see source code for examples. The first argument is the index of the chain to modify and the second is the part name of the BSDL library.
Examples:	JTAG.set(2,”XC9572XL”) #Will set index:2 to use the Xilinx XC9572XL library.

Command:	JTAG.add
Parameters:	String
Description:	This command allows you to add a BSDL file definition to the JTAG chain. The parameter is the name of the BSDL device. You can either use a pre-designed BSDL in the software or to a newly created one using the BSDL commands.
Examples:	JTAG.add("BSDL_NAME")

Command:	JTAG.validate
Description:	This command will validate the JTAG chain and devices. For example, if you use the JTAG.clear and JTAG.set or JTAG.add commands, you will need to run this after to properly validate the chain and entire the entire IR bit size is correct.
Examples:	JTAG.validate

Command:	JTAG.Writeword
Parameters:	UInteger, Integer

Description:	Writes a word of data (32-bit) to the JTAG device. First parameter is the 32-bit address and the second parameter is the word to write.
Examples:	JTAG.Writeword(0xFFFF1000,0xFF10)

Command:	JTAG.Readword
Parameters:	UInteger
Returns:	UInteger
Description:	Reads a word (32-bits) from the JTAG device and returns the value.
Examples:	RES = JTAG.Readword(0x1FC00000)

Command:	JTAG.control
Parameters:	UInteger
Returns:	UInteger
Description:	Sends a JTAG control message to the target device. These types of commands are very dependent on the target device. This can be used to stop (0x10000) or start (0x0) the target processor. The result of the command is returned.
Examples:	JTAG.control(0x10000) #Stops the target processor

Command:	JTAG.MemoryInit
Parameters:	String, Integer, Integer (optional)
Description:	Initializes the JTAG memory controller and connects the host to a memory device connected to the target processor. First parameter is the memory type: “CFI”, or “SPI”. The second parameter is the DMA address (for DMA/CFI) or the SPI controller index: 1=Broadcom, 2=Atheros.
Examples:	MemIndex = JTAG.MemoryInit(“CFI”,0x1FC00000) MemIndex = JTAG.MemoryInit(“SPI”,1) #BCM SPI

Command:	JTAG.Debug
Parameters:	Bool (true or flase)
Description:	Writes the JTAG data register with the standard flag to put the target device into debug mode: (PRACC|PROBEN|SETDEV|JTAGBRK)
Examples:	JTAG.Debug(true) #Will send the JTAG debug command

Command:	JTAG.CpuReset
Description:	Writes the JTAG data register with the standard flag to issue a processor reset. This command can have different results depending on the particular processor part: (PRRST|PERRST)
Examples:	JTAG.CpuReset #Will send a CPU reset command

Command:	JTAG.RunSVF
Parameters:	Data
Description:	This command will run a “Serial Vactor Format” file and process and write all of the commands to a connected JTAG device. This can be use to program Xilinx or Lattice CPLDs for example.
Examples:	JTAG.RunSVF(DataVar) #Runs a *.SVF file

Command:	JTAG.RunXSVF
Parameters:	Data
Description:	This command will run a compact (binary) “Serial Vactor Format” file and process and write all of the commands to a connected JTAG device. This can be use to program Xilinx CPLDs for example.
Examples:	JTAG.RunXSVF(DataVar) #Runs a *.XSVF file

Command:	JTAG.ExitState
Parameters:	Bool
Description:	This command will will enable or disable the go to test-logic-reset after a SVF file has been executed. Default behavior is enabled.
Examples:	JTAG.ExitState(False)

Command:	JTAG.ShiftDR
Parameters:	Data, Integer, Bool (optional)
Returns:	Data
Description:	Selects the DR register and then shifts data into it. First parameter is the data array, second parameter is the number of bits to shift in. The last optional parameter specifies to leave the DR-resister, the default is yes. The TDO data is shifted in and is returned.
Examples:	JTAG.ShiftDR(DATA,32)

Command:	JTAG.ShiftIR
Parameters:	Data
Returns:	Data
Description:	Selects the IR register and then shifts data into it. First parameter is the data array.
Examples:	JTAG.ShiftIR(DATA)

Command:	JTAG.ShiftOut
Parameters:	Data, Integer, Bool (optional)
Returns:	Data

Description:	Shifts data out at the current state. The first parameter is the data to shift, the second is the number of bits, the last parameter (optional) is to set the last TMS bit to exit the state (default is no)
Examples:	JTAG.ShiftOut(DATA,32,False)

Command:	JTAG.TapReset
Parameters:	None
Description:	Resets the test-access-port state machine to test-logic-reset.
Examples:	JTAG.TapReset()

Command:	JTAG.State
Parameters:	String
Description:	Changes the current state of the TAP. The parameter is a string. Valid inputs are: “RunTestIdle”, ”Select_DR”, ”Capture_DR”, ”Shift_DR”, ”Exit1_DR”, ”Pause_DR”, ”Exit2_DR”, ”Update_DR”, ”Select_IR”, ”Capture_IR”, ”Shift_IR”, “Exit1_IR”, “Pause_IR”, “Exit2_IR”, “Update_IR”.
Examples:	JTAG.State(“RunTestIdle”)

Command:	JTAG.GrayCode
Parameters:	Integer, Bool (optional)
Returns:	UInteger
Description:	Returns the gray code (8-bit table) for a specific index. Optional parameter is used to specify if the table reverse should be used.
Examples:	JTAG.GrayCode(2, true) #Returns 0xC0

BSDL Library

Command:	BSDL.new
Parameters:	String, String, String (optional)
Returns:	Integer
Description:	This command allows you to create a new BSDL definition entry. This command should then be followed by the BSDL. Parameter to specify the library parameters. This command returns an Integer that indicates the index within the library that this entry exists. The first parameter is the manufacture name and the second is the part name. The third parameter is the optional package type. If you add multiple devices of the same part, you should use this to distinguish them.
Examples:	lib_index = BSDL.new(“Xilinx”, “X95XXX”, “QFP44”)

Command:	BSDL.find

Parameters:	String, String (optional)
Returns:	Integer
Description:	If you need to find the index of a BSDL definition, this command can be used. The parameter is the name of the library to look up. If no library is found, this command will return -1. Optionally, you can specify the package type (if there are more than one package type in the database).
Examples:	lib_index = BSDL.find(“X95XXX”)

Command:	BSDL.paramater
Parameters:	String, UInteger (or positive Integer)
Returns:	Integer
Description:	This command will specify (or override) a parameter of a BSDL definition. The library is accessed using its index, which is derived from either creating a new definition or using the BSDL.Find command. The two main parameters are the name of the parameter to modify, and its value. **Valid Parameter Names:** ID_JEDEC ID_MASK IR_LEN BS_LEN IDCODE BYPASS INTEST EXTEST SAMPLE CLAMP HIGHZ PRELOAD USERCODE DISVAL

Examples:	index = BSDL.New("X95100LA") #create the definition or find one
	BSDL(index).parameter("IR_LEN", 10)
	BSDL(index).parameter("BS_LEN", 521)
	BSDL(index).parameter("EXTEST", 0x00)
	BSDL(index).parameter("SAMPLE", 0x01)

Boundary Scan Programmer commands

Command:	BoundaryScan.Setup
Parameters:	None
Description:	First command needed, sets up initial values.
Examples:	BoundaryScan.Setup()

Command:	BoundaryScan.Init
Parameters:	Integer (Optional)
Description:	After all ADx and DQx and control pins have been specified (from the AddPin command), this command initializes the engine. Optional parameter can be used to specify: AUTO (0), X8 over X16 bus (1).
Examples:	BoundaryScan.Init()

Command:	BoundaryScan.AddPin

Parameters:	String, Integer, Integer, Integer (optional)
Description:	Adds a pin to BSR association. This is how you define and map which pins from a Flash device should go to which pin on the target device. Parameter list: pin name, output/bidir index, control index, input index (opt) Valid pin names: DQx, ADx, WE#, OE#, CE#, WP#, RESET#, BYTE# Note: AD, DQ, WE and OE pins are mandatory, and the others are optional if the board does not route them to thse host controller. If the IO cell is bidir and the control cell is next index, then you can omit the other 2 parameters.
Examples:	BoundaryScan.AddPin(”DQ1”, 331)

Command:	BoundaryScan.SetBSR
Parameters:	Integer, Integer, Bool
Description:	Sets a bit in the boundary scan register to OUTPUT and to logic high or low. The first paramer is the index of the output cell bit, the second is the control bit and the third parameter is true for logic high or false for logic low. The purpose of this command is to set specific bits on a board that would be required to enable flash memory access.
Examples:	BoundaryScan.SetBSR(369,370,True) #Sets IO at 369 to output and HIGH

Command:	BoundaryScan.WriteBSR
Parameters:	None
Description:	Used is combination to SetBSR, calling this command will write out the BSR using the JTAG EXTEST command.
Examples:	BoundaryScan.WriteBSR()

Command:	BoundaryScan.Detect
Parameters:	None
Description:	Initializes and attempts to detect a CFI compatible NOR memory as specified using the Setup and AddPin commands.
Examples:	BoundaryScan.Detect()

Parallel port commands

The follow commands are used to manually manipulate the parallel port (DQ0-DQ15 and ADDR) when using the Parallel NOR mode.

Command:	Parallel.Command
Parameters:	UInteger, UInteger
Description:	Performs a data command to the parallel port. First argument is the address, the second argument is the 8 or 16-bit data to write. A pulse on the WE# occurs with this command.

Examples:	#Perform READ IDENT command:
	Parallel.Command(0x5555, 0xAA)
	Parallel.Command(0x2AAA, 0x55)
	Parallel.Command(0x5555, 0x90)
	ID1 = Parallel.Read(0x0)
	ID2 = Parallel.Read(0x2)
	print("Flash ID is: " + string.hex(ID1) + " " + string.hex(ID2))

Command:	Parallel.Write
Parameters:	UInteger, UInteger
Description:	Similar to the Parallel.Command, this does a data write command, but the address is parameter is logical to the Flash device. So this is useful for writing commands that will need to have their address pins set according to the hardware address.

Examples:	#Perform UNLOCK BLOCK on address 0x10000
	Parallel.Write(0x10000, 0x50)
	Parallel.Write(0x10000, 0x60)
	Parallel.Write(0x10000, 0xD0)

Command:	Parallel.Read
Parameters:	UInteger
Description:	This performs a read operation. The only parameter is the address to read from.
Examples:	Parallel.Read(0x10)

Miscellaneous commands

Command:	Writeline (or print)
Parameters:	Any data type
Description:	Displays a message to the console. You can input an Integer, String, or Data.
Examples:	writeline("this is only a test")

Command:	Msgbox
Parameters:	String
Description:	Displays a message to the user using a pop-up box.
Examples:	msgbox("Hello World!")

Command:	Status
Parameters:	String
Description:	This sets the status text (the bottom bar of the software).
Examples:	status("script is complete")

Command:	Refresh

Parameters:	None
Description:	Updates any of the connected memory device's hex editors. Useful if you have modified any of the screen contents in a script file.
Examples:	status()

Command:	Sleep
Parameters:	Integer
Description:	Waits the specified amount of time (in milliseconds), useful only in scripts.
Examples:	Sleep(1000) #Waits 1 second

Command:	Verify
Parameters:	Bool
Description:	Used to enable or disable the flash programming verification process.
Examples:	Verify(true)

Command:	Mode
Parameters:	None
Returns:	String
Description:	Returns the current operation mode: SPI, JTAG, EXTIO, I2C, etc.
Examples:	mode() #Returns "JTAG"

Command:	ask
Parameters:	String
Returns:	Bool
Description:	Asks the user a yes or no question and returns that value. You can use this in an if statement to make conditional sections.
Examples:	ask("Continue script?")

Command:	endian
Parameters:	String
Description:	Allows the endian mode to be changed. The input can be a string to set the mode: “MSB” or “LSB”. Some JTAG devices will need to have this setting specified. “MSB” “LSB16” “LSB8” are valid options.
Examples:	endian(“LSB”)

Command:	abort
Description:	Aborts any script that is running. Use this in the console to quit a script that may not be working as intended.
Examples:	abort

Command:	crc32
Parameters:	Data
Returns:	Integer
Description:	Computes a standard CRC-32 checksum for a given data variable. The command crc16 is also supported for legacy uses.

Examples:	data_var = memory.read(0,memory.size) #Reads all data from a device
	c32_value = crc32(data_var) #Generates checksum
	print(string.hex(c32_value)) #Prints the hex value

Command:	cint
Parameters:	UInteger
Returns:	Integer
Description:	Converts an Integer Data Type to an Unsigned Data Type

Examples:	var1 = 40U #creates a variable with an UInteger Data Type
	var2 = cint(var1) #converts that variable to Integer Data Type

Command:	cuint
Parameters:	Integer
Returns:	UInteger
Description:	Converts an Unsigned Data Type to an Integer Data Type

Examples:	var1 = 40 #creates a variable an Integer Data Type
	var2 = cuint(var1) #converts that variable to an UInteger Data Type