MBScript Reference

Rev. 2.4.42

Table of Contents

Purpose of this document

The Anewin MBScript Reference provides comprehensive information regarding the MBScript language. The MBScript language is integrated into the FiveWire application providing programmable control of the product features. This supports automation of tests and analysis of captured data.

MBScript Commands

Overview

The MicroBench MBScript scripting language provides the ability to automate the setup and control of the instrument tool set. It follows the syntactic conventions of the 'C' language with some more modern modifications and extensions. If you are familiar with the 'C' language, MBScript will feel very natural to you.

Reserved Words

!=, ==, >, <, >=, <=, ||, &&, |, &	getFilePath()	sleep(*ms*)
+, -, *, /, %,^, <<, >>	global	split(*src, del, dest[][, first]*)
abortFlag	if-else	strFind(*start, find, src*)
addLibrary *path*	intToChar(*val*)	strFormat(*format, values...*)
array	isArrayDefined(*array[], index*)	*String:* +. ==, !=
array(*index, array[], item, ...*)	isDefined(*var*)	strLen(*src*)
assArray	listSort	strReplace(*index, src, repl*)
break	local	strSub(*start, len, src*)
breakpoint *item*	localDirectory	strSubstitute(*src, old, new*)
carriageReturn	log(), exp(), sqrt()	*suffix:* ++, --
charToInt(*char*)	mbCommand(*tool, cmd*)	switch-case-defalut
closeCommPort()	myDocuments	textWindow(*msg*)
closeMessageWindow()	newLine	textWindowLoc(*msg, x, y*)
closePictureWindow	openCommPort(*baud, port*)	time
closeReadFile(*handle*)	openMessageWindow()	toArray
closeVideoWindow	openPictureWindow	toAssArray
closeWebWindow	openReadFile(*path*)	toHex(*val*)
closeWriteFile(*handle*)	openVideoWindow	toInt(*val*)
cmdLine	openWebWindow	toNum
continue	openWriteFile(*path*)	toStr(*val*)
commRespGet()	ord	trim(*string*)
commRespWait(*tmo*)	osTick	true
commSendLine(*str, del*)	pi	userScriptsDirectory
continueFlag	*prefix:* -,++,--,~	using
date	rand(*bot, top*)	version
do-while	readFileLine(*handle*)	videoMark
eqn: =, +=, -=, *=, /=, |=, &=,^=	readInput()	while
execVarStr	readSelection()	writeFileLine(*handle*)
exit	runProgram(*pgm [, args]*)	writeMessage(*msg*)
false	runScript(*path*)	writeMessageLine(*msg*, *line*)
for	readSelection(*label, list*)	yesNoWindow(*msg*)
foreach	selectWindow	
function	sin(*deg*), cos(*deg*), tan(*deg*)	

Differences from C

Interpreted not Compiled

MBScript source files are simple text files. When the script is run, it is checked for certain kinds of errors and processed to remove non-executable content. It is then interpreted sequentially until the last line is reached or the 'exit' command is executed.

Automatic Variables

Unlike 'C', variables in MBScript do not need to be declared. Variables are created as they are encountered during execution. Two types of variables are supported: numeric and string. During evaluation of the variable, if its contents can be interpreted as a number it will be treated as a numeric variable. Otherwise it will be treated as a string. For example:

```
a=5; // Numeric assignment

a=0x3A; // Hexadecimal numeric assignment

b=a; // b now contains the number 5

b=c; // Terminates with an error, c is not yet defined

b="c"; // Assigns the string "c" to the variable b
```

The '+' sum operation can be applied to numbers or strings producing either a numeric sum or a concatenated string. Both arguments to the sum operator must be of the same type to avoid ambiguity. The functions toStr(<value>) and toNum(<string>) provide conversion between the types to force the desired operation outcome.

Order of evaluation

Expressions are always evaluated left to right. && and || have a higher precedence. Parenthesis should be used to force correct evaluation if left to right is not what you want. For example:

```
a < 2 && a > 0     // && has precedent

a < 2 + 3 && a > 0 // Incorrect

a < (2 + 3) && !b  // Use parenthesis
```

Associative Arrays

MBScript provides a single array type. Each array includes a name and index. The index can be a number or string. Like variables above, an array element must be assigned before it can be accessed. For example:

```
a[0]=5;
a[1]=10;
a[0]=a[2]; // Produces an error – a[2] has not been assigned yet
b[first] = "first"; // Use a string to index the array b
b[second] = 2;
b[first]=b[second]; // b[first] now contains the number 2

// Define a numericly indexed array starting with zero
n = array(0, myArray[], "item1", "item2", "itemN");
q = myArray[1]; / q now contains item2

// Define associative array with indexes a, b, and c
n = assArray(assoc[], "a:item1", "b:item2", "c:itemN");
q = assoc["c"]; // q now contains "itemN"

// Convert a CSV list to a zero based numeric array
list = "a,b,c";
n = toArray(list, myArray[], ",", 0);
q = myArray[0]; // q now contains "a"

// Convert a compound list to an associative array assoc
//  element separator "," and sub-element separator ";"
//  the zeroth sub-element is the key
n = toAssArray("a;1,b;2,c;3", assoc[], ",",";" 0);
n = toAssArray("a;1,b;2,c;3", assoc[]); // or use defaults , ; 0
q = assoc[b]; // q will equal 2 after assignment
```

MBScript by Example

Instrument Commands

Instrument commands are comprised of a command word and any required parameters separated by white space. Commands are directed to one of the tools: liveLogic, logicAnalyzer, logicSource, waveformSource, protocol, or system.

There are two methods for sending commands to a tool. The first sends one command at a time and supports algorithmic command generation. It supports capturing any command response. The second sends multiple commands to a selected tool but the commands must be constant strings.

```
// First method
mbCommand("system",  "openWindow liveLogic");

// or use
instrument = "liveLogic";
mbCommand("system",  "openWindow " + instrument);

// with return value
toolIsOpen = mbCommand("system",  "isToolOpen liveLogic");
if(toolIsOpen)
{
      // some code
}

// Second method
using system
{
      "openWindow liveLogic";
      "positionWindow liveLogic 100 100";
}
```

Tool commands either return a value or return an empty string. If there is an error in the command a string will be returned that starts with the word "Error".

Some tool commands that return a data array such as data samples.

```
// Get a captured waveform from Live Logic
waveform = mbCommand("liveLogic",  "getWaveform");

// Convert the waveform to an array of samples
numSamples = split(waveform, ",", samples[]);

// Process each sample
foreach(sample in samples[])
{
	// Get the channel and time values from the sample
	split(sample, ";", values[]);
	ch1 = values[0];
	ch2 = values[1];
	time = values[2];

	// Do something with the information
}
```

Descriptions of all of the tool commands are provided at the end of this document along with complete example programs.

String Manipulation

strLen(<string>) returns the length of a string

```
// n is set to 8
n = strLen("FiveWire");

// Show the value in a dialog box
textWindow(n);
```

strFind(<start>, <find>, <string>**)** returns the index of the start of the find string or -1 if not found.

```
// n is set to the location of the second I, 5 in this case
n = strFind(0, "ir", "FiveWire");

// Show the value in a dialog box
textWindow(n);
```

strSub(<*start*>, <*length*>, <*source*>**)** returns a substring from the source string.

```
// Sets s to the string "ir"
s = strSub(5, 2, "FiveWire");

// Show the value in a dialog box
textWindow(s);
```

strSubstitute(<source>, <find>, <new>**)** returns string with find replaced by new.

```
// Replaces W in source string with w setting s to "Fivewire"
s = strSub("FiveWire", "W", "w");

// Show the value in a dialog box
textWindow(s);
```

strReplace(<start>, <source>, <new>**)** returns string with new overwrite at start index.

```
// Returns string "The First"
s = strReplace(4, "The first", "F");

// Show the value in a dialog box
textWindow(s);
```

strFormat(<*format* >[, <*value0*>, <*value1*>, ...]) returns a formatted string

```
// Get the Logic Analyzer data for channel 4
ch = 4;
cmd = strFormat("getWaveform  {0}", toStr(ch));
chData = mbCommand("logicAnalyzer", cmd);

// Other format options
{n:f} where n is the var index and f is the format
{0:d3} decimal 0025
{0:0.00e0} exponential 1.25e-6
{0:X4} hexadecimal A5A5
{0:f4} fixed point 3.5500
{0:r} round up a float to the next higher integer
{0:p} display as percentage
```

trim(<string>**)** returns string with leading and trailing white space removed.

```
// trim reduces length of string from 10 to 8
s = trim(" FiveWire ");
n = strLen(s);
```

Predefined variables/constants

true has the value 1and **false** has the value 0

```
// Use a flag to control a loop
n=3;
flag = true;
do
{
      // Do something useful
      n--;
      flag = n > 0;
      textWindow(toStr(flag) + " " + toStr(n));
}
while(flag);

textWindow("Done");
```

date has the string value of the current date

time has the string value of the current time

```
// Get the date and time and display it
textWindow(date + ", " + time);
```

osTick has the value of the system tick in milliseconds

```
// Timed delay of 2000 ms
textWindow("Start");
futureTime = osTick + 2000;
while(osTick < futureTime)
      ;
textWindow("Done");
```

pi returns the constant 3.14159…

```
// The value of pi
textWindow(pi);
```

newLine returns the new line character

carriageReturn returns the carriage return character

cmdLine returns the user config string from the script decoder

```
// The command line entered when configuring the scrip decoder
textWindow(cmdLine);
```

version has the numeric value of the MBScript version

```
// Check the current version of the MBScript interpreter
if(version < 1.0)
      textWindow("Need to upgrade");
else
      textWindow("Up to date");
```

localDirectory returns the complete path to the directory from which the script was started

```
// Get the directory path of the executing script
textWindow(scriptDirectory);
```

userScriptsDirectory returns the complete path to the userScripts directory

```
// Get the userScripts director path
textWindow(userScriptsDirectory);
```

myDocuments returns the complete path to my documents

```
// Get the directory path of My Documents
textWindow(myDocuments);
```

abortFlag has the value of the abort flag cleared at startup and set if the message window 'Abort' button is pressed

```
// Keep executing until the user aborts the program
seconds = 0;
openMessageWindow();
while(!abortFlag)
{
	sleep(1000);
	seconds++;
	writeMessageLine(toStr(seconds) + " seconds", 0);
}
closeMessageWindow();
```

Variable scope

Variables declared outside of any curly braces {} are global to the script. Curly braces define a block of code and variables declared within that block.

global

Forces the variable to exist in global scope. If the variable does not exist in global scope a new variable will be created in global scope.

```
// Using global
function setGlob(value)
{
	global glob = value;
}

if(isDefined(glob))
	textWindow(glob);
else
	textWindow("glob not defined");

setGlob(5);

if(isDefined(glob))
	textWindow(glob);
else
	textWindow("glob not defined");
```

local

If the named variable does not exist in the current scope but does exist in a wider scope, the wider scope variable will be used. The local prefix requires the variable to exist in the current scope. If it does not, a new variable will be created in the current scope even if the same variable name exists in a wider scope.

The local prefix can also be used to declare a global array before attempting to assign values to the array

```
// Using local
val = false;
function useLocal()
{
	local val = true;
}
useLocal();
if(val)
	textWindow("global val was changed");
else
	textWindow("global val was not changed");
```

```
// Retrieve LA acquisition
samples = mbCommand("logicAnalyzer", "getWaveform 210");

// Break into a list of samples
n = listToArray(samples, ",", sample[], 0);

// Declare data and time arrays in this scope
local data[];
local time[];

// Break each sample into its data and time components
for(i=0;i<n;i++)
{
	listToArray(sample[i], ";", values[], 0);
	data[i] = values[0];
	time[i] = values[1];
}

// Display the data and time components
openMessageWindow();
for(i=0;i<n;i++)
{
	msg = toStr(data[i]) + " : " + toStr(time[i]);
	writeMessage(msg);
}

// Wait for the user to terminate via abort button
while(!abortFlag)
	;
closeMessageWindow();
exit()
```

Flow Control

if else statements

```
// Keep executing until the user aborts the program
upToDate = true;
if(version < 1.0)
{
	upToDate = false;
	textWindow("Older version");
}
else
	textWindow("Current version");
```

for statements

```
// For loops
for(n = 0; n < 2; n++)
	textWindow(n);

openMessageWindow();
for(n = 0; n < 10; n++)
{
	writeMessageLine(toStr(n), 0);
	sleep(1000);
	if(abortFlag)
	{
		closeMessageWindow();
		break;
	}
}
closeMessageWindow();
```

foreach statements

```
// Foreach
foreach(value, valArray[])
	textWindow(value);

foreach(key, value, valArray[])
	textWindow(key, value);

foreach(value in valArray[])
	textWindow(value);
```

while and do-while statements

```
// While loop
n = 2;
textWindow("Start while");
while (n > 0)
{
	n--;
	sleep(1000);
}
textWindow("Done");

// Do loop
n = 2;
textWindow("Start do");
do
{
	n--;
	sleep(1000);
}
while (n > 0)
textWindow("Done");
```

switch statement

```
// switch - must be numeric variable
n = 3;
switch(n)
{
	case 1:
		textWindow("Value is 1");
		break;
	case 2:
		textWindow("Value is 2");
		break;
	case 3:
		textWindow("Value is 3");
		break;
	default:
		textWindow("Value is not expected");
		break;
}
```

break

Terminates the enclosing while, for, or do-while execution

continue

Restarts the execution of the enclosing while, for, or do-while at the beginning of the block.

User Defined Functions

Function parameters can be a number or string, an array reference such as *myArray[],* or an expression that evaluates to a number or string. Parameters are passed by value. An array reference passes a pointer to the calling functions array. Any changes made affect the callers array variable. If the array does not exist in the callers scope, an array will be created.

```
// Function that sends commands to Live Logic tool
function LL(cmd)
{
	response = mbCommand("liveLogic", cmd);
	return response;
}

// Use the function
resp = LL("run");
if(strLen(resp) > 0)
	textWindow(resp);
else
	textWindow("Done");
```

```
// Copy data and time to their own arrays
function copyValues(v[], da[], ti[])
{
      da[i] = values[0];
      ti[i] = values[1];
}

// Break a sample into its data and time components
function getValues(s[], d[], t[])
{
      for(i=0; i<n; i++)
      {
            split(s[i], ";", values[]);
            copyValues(values[], d[], t[]);
      }
}

// Get data from LA and break it into sample array
samples = mbCommand("logicAnalyzer", "getWaveform 10");
n = split(samples, ",", sample[]);

// Declare global arrays data and time
//   Could leave these lines out and allow the folling
//   function call to implicitly create them
local data[];
local time[];

// Process samples into data and time arrays
getValues(sample[], data[], time[]);

// Display the data for the user
openMessageWindow();
for(i=0;i<n;i++)
{
      msg = toStr(data[i]) + " : " + toStr(time[i]);
      writeMessage(msg);
}

// Clean up and exit when abort button is pressed
while(!abortFlag)
      ;
closeMessageWindow();

exit();
```

File I/O

MBScript provides the ability to read and write simple text files one line at a time.

```
// Write a single line to a file
data=5;
handle = openWriteFile("C:/someTextFile.txt");
writeFileLine(handle, strFormat("Data = ", data));
closeWriteFile(handle);


// Read from a file
handle=openReadFile("C:/someTextFile.txt");
line=readFileLine(handle);
if(0 > strFind(0,"::EOF:", line))
{
      s = strFormat("Line from file is: {0}", line);
      textWindow(s);
}
closeReadFile(handle);

// Open a file dialog to get the path for a file
path = getFilePath();
path = getFilePath(directoryPath);
path = getFilePath(directoryPath, suffix);

// Open the selected file for read
handle = openReadFile(path);
```

Comm port I/O

MBScript provides the ability to read and write to a comm port.

```
// Open COM3, optional term char with default \n
openCommPort(38400, "COM3");
openCommPort(38400, "COM3", "\r");

// Send a string to COM3
commSend("test");
// Send a string to COM3, sendLine appends \n to string
commSendLine("test");
// Send a string to COM3 with optional 25ms delay between each char
commSendLine("cmpTime", 25);

// Wait up to 1 second for a response
if(commRespWait(1000))
      textWindow(commRespGet());

// Get the number of response strings that are waiting
n = commRespCount();

//Release the comm port
closeCommPort();
```

GUI Windows

MBScript provides a number of windows that can display user data and other useful information.

textWindow(*message [, message, ...]***)**

textWindowLoc(*message , x, y***)**

Displays a text window showing the *message* string to the user and an 'OK' button. Script execution stops until the 'OK' button is pressed. Using textWindow, one or more message strings can be included separated by commas. Using textWindowLoc a single message can be displayed at the specified x,y location on the screen.

```
// Display some text
textWindow(date,time);
textWindowLoc(date, 100, 150);
```

yesNoWindow(*message [, message, ...]***)**

Displays a text window showing the *message* string to the user and two buttons, 'Yes' and 'No'. If the 'Yes' button is pressed *true* is returned. If the 'No' button is pressed *false* is returned. Script execution is stopped until one of the buttons is pressed. One or more message strings can be included separated by commas. Each line will be presented in a sequence of lines in the dialog box.

```
// Get user input
if(yesNoWindow("Yes or no"))
      textWindow("Selected yes");
else
      textWindow("Selected no");
```

openMessageWindow()

Displays a multi-line message window which does not block execution of the script. An Abort button is bound to the abortFlag providing a method to terminate the script.

writeMessage(*msg***)**

Displays the *msg* string in the message window on sequential lines

writeMessageLine(*msg, line***)**

Displays the *msg* string in the message window on the specified *line*

closeMessageWindow()

Closes the message window

```
// Display some data
openMessageWindow();
n = 5;
writeMessageLine("Count down", 0);
while (n > 0)
{
	writeMessage(toStr(n));
	sleep(1000);
	n--;
	if(abortFlag)
	{
		closeMessageWindow();
		break;
	}
}
closeMessageWindow();
```

readInput(*[[label], default input]*)

Returns the user input string or an empty string if canceled. Note a string is always returned event if the value represents a number. Use the toNum() function to convert the response to a number.

```
// Get some data
userInput = readInput("Enter an amount", "3");
textWindow(userInput);
```

readSelection(*label, select list*)

Returns the selected string from the CSV list or an empty string if canceled

```
// Get one of the possible colors
userInput = readInput("Select color", "red,green,blue");
textWindow(userInput);
```

openPictureWindow(*[[label], default input]*)

Open a window that contains a picture

```
// Display a picture
openPictureWindow();
path = myDocuments + "\\Anewin\\FiveWire\\userData\\pic.png";
mbCommand("pictureWindow", "loadFileFullPath " + path);
```

closePictureWindow()

openVideoWindow()

Open a window that shows a video

```
// Display a video
openVideoWindow();
path = myDocuments + "\\Anewin\\FiveWire\\userData\\vid.wmv";
mbCommand("videoWindow", "loadFileFullPath " + path);
mbCommand("videoWindow", "play");
```

closeVideoWindow()

openWebWindow()

Open a window that displays a web page

```
// Display a web page
openWebWindow();

// Split the url to avoid // comment
url = "https:/" + "/www.google.com";
```

```
mbCommand("webWindow", "loadFileFullPath " + url);
```

closeWebWindow(*[[label], default input]***)**

Standard Functions

mbCommand(*"tool", "command"*)

Sends a command string to the specified tool and returns the commands response.

```
// Get some data
runState = mbCommand("liveLogic", "getRunState");
if(runState == "stopped")
	textWindow("Live Logic is stopped");
else
	textWindow("Live Logic is running");
```

isDefined(*var*)

Returns true if *var* has been assigned (created) otherwise false

```
// Verify that vars are defined before using
myVar = "A defined variable";
if(isDefined(test))
	textWindow(test);
else
	textWindow("test is not defined");
if(isDefined(myVar))
	textWindow(myVar);
```

isArrayDefined(*arrayName[], index*)

Returns true if *var[index]* has been assigned (created) otherwise false

```
// Verify that array vars are defined before using
one = "one";
two = "two";
myVar[one] = 1;
if(isArrayDefined(myVar[], one))
      textWindow(myVar[one]);
else
      textWindow("myVar[one] is not defined");
if(isArrayDefined(myVar[], two))
      textWindow(myVar[two]);
else
      textWindow("myVar[two] is not defined");
```

runScript(*path*)

Executes script at *path*

```
// hello.mbs script in local directory
textWindow("Hello");

// Script that calls hello.mbs script
runScript(scriptDirectory + "hello.mbs");
```

breakpoint [*{string | variable | array}, ...*]

Stops script execution and presents a dialog indicating the source line of the break point and following lines which contain the name and value of listed variables or the contents of the literal string.

```
// Breakpoint debugging
a = 1;
b = 2;
c[0] = "test";
breakpoint "Stop and inspect vars" a b c[0];
```

sleep(*milliseconds***)**

Delays script execution for specified time

```
// Count down seconds
n = 5;
textWindow("Start");
while(n > 0)
{
	n--;
	sleep(1000);
}
textWindow("Done");
```

Binary Math Operations

Numeric: +, -, *, /, %,^, <<, >> ==, !=, >, <, >=, <=, ||, &&, |, &

Note: Operations that necessarily operate on integers are limited to 52 bits.

String: +, ==, !=

```
// Do some math
//   Evaluation is from left to right
n = 5 + 3 - 2 / 2 * 3;
textWindow("The result is " + toStr(n));
textWindow("1 << 4 = " + toStr(1<<4));
textWindow("5 % 2 = " + toStr(5%2));
textWindow("5 > 4 = " + toStr(5>4));
textWindow("Incorrect! " + toStr(5>4 && 3>2));
textWindow("Correct " + toStr((5>4) && (3>2)));
textWindow("abc == abd = " + toStr("abc"=="abd"));
```

Unary Math Operations

Prefix: -, ++, --

Suffix: ++, --

```
// Do some math
n = 1;
textWindow("n++ " + toStr(n++));
textWindow("++n " + toStr(++n));
```

Equation

=, +=, -=, *=, /=, %=, |=, &=

```
// Do some math
n = 8;
n /= 4;
textWindow(n);
n = 1;
n |= 4;
textWindow(n);
```

Math Functions

sin(deg), cos(deg), tan(deg), rand(smallest, largest), log(), exp(), sqrt()

```
// Do some math
textWindow("sin(45) = " + toStr(sin(45)));
textWindow("rand(0,255) = " + toStr(rand(0,255)));
textWindow("sqrt(2) = " + toStr(sqrt(2)));
```

Libraries

Standard and user defined libraries may be included within the script using the addLibrary command. Libraries are MBScript files that are evaluated at the addLibrary line.

Libraries typically contain function definitions which are used in the main script. If executable code is included in the library it will be executed when the addLibrary line is evaluated.

A library stdLib.mbs in the "../Documents/Anewin/FiveWire/userScripts" directory contains useful helper functions for accessing instruments and their data.

```
// Example library in the local file stdLib.mbs
function LL(cmd)
{
	local resp = mbCommand("liveLogic", cmd);
	if((strLen(resp) > 0) && (strFind(0, "Error", resp) == 0))
	{
		if(yesNoWindow("Command error line " + toStr(callLine) + "
			- exit?", resp))
			exit;
	}
	return resp;
}

See standard library source for parameter details.
```

```
// MBScript file test.mbs that uses the local library
addLibrary userScriptsDirectory + "stdLib.mbs";
LL("run");
```

stdLib.mbs functions

```
SY(cmd); // Send cmd to the System tool
LL(cmd); // Send cmd to the liveLogic tool
LA(cmd); // Send cmd to the logicAnalyzer tool
LS(cmd); // Send cmd to the logiSource tool
WS(cmd); // Send cmd to the waveformSource tool
PR(cmd); // Send cmd to the protocol tool
```

llLib.mbs functions

```
LLgetDVM(channel); // Get the DVM value
LLisStopped(); // Returns true if acquisition has stopped
LLgetWaveform(channel, time[], data[]); // Get acquisition data
```

laLib.mbs functions

```
LAisStopped(); // Returns true if acquisition has stopped
LAgetWaveform(channel, time[], data[]); // Get acquisition data
```

wsLib.mbs functions

```
WSsetFast(flag); // Controls fast mode
WSsetPoint(index, value, duration, mode); // Set an analog point
WsconfigMode(controller, type, target, count); // Set jump control
WSupdate(duration); // Change duration of an analog point
```

lsLib.mbs functions

```
LSvector(sequence, value, duration); // Set a vector
LsvectorMask(sequence, value, mask, duration); // Set bits in vector
LSlongVector(sequence, value, duration); // Set extended vector
LsconfigMode(sequence, mode); // Set jump on sequence
LsconfigJump(controller, type, target); // Set jump type
LSconfigLoop(controller, target, count); // Set outer loop
LsconfigInnerLoop(controller, target, count); // Set inner loop
```

MBScript Decoders

MBScript can be used to write custom decoders. By placing an mbs file in the ../Documents/Anewin/FiveWire/decoders/ directory, the script name will be listed in the Live Logic and Logic Analyzer Add Decoder menu list. Scripts not in the decoder directory can also be used by selecting 'User Script' in the decoder selection menu.

A script that is added as a decoder will be executed at the end of each acquisition. Typically the script will read the acquisition data and analyze the data for the information to be displayed. Information can be placed on the acquisition display using the Marks API included with each tool. When a script decoder is added, it will only execute in single acquisition mode and will be ignored when continuous mode is selected.

MBScript includes optional special comment lines that are evaluated when the decoder is added to the tool. If these comments exist, a dialog will be presented to the user to configure the decoder. These comment lines define the configure dialog properties that get information from the user such as channel assignments. These comments must be in the following format:

```
//LL@=<Label1>;<var1>;<default>;<val1>;val2>;<valN>
//LA@=<Label1>;<var1>;<default>;<val1>;val2>;<valN>
```

1. **Prefix:** //LL@= or //LA@= selects which tool the values are applied to.
2. **Label:** A series of chars and spaces that describe the entry
3. **var:** The name of the variable that will be initialized with the selected value
4. **default:** The value that will be used if the user does not select a value
5. **Values:** one or more values that will be added to the selection list
6. **Additional selections:** Up to 4 selections can be included in the special comment. To add another value, add a comma after the current value and follow the same pattern starting with the item label.

When the script is run, a global variable set to the selected value for each varName will be defined and can be referenced in the script. One additional global variable 'tool' is defined which indicates which tool is associated with the decoder:

tool is set to either "liveLogic" or "logicAnalyzer"

Section 2 FiveWire System Commands

These commands provide control over functions that are system wide. To execute these commands from MBScript use **mbCommand("system", "*command*")** where *command* is one of the commands described below.

Window Commands

openWindow

closeWindow

positionWindow

minimizeWindow

restoreWindow

Setup and Execution Commands

runScript

openSystem

saveSystem

Management and Control Commands

clickSystemButton

instrumentConnected

getRevision

getSwRevisionValues

getSerialNumber

getCurrentPath

Complete System Command Reference

clickSystemButton

Simulates user pressing the system button resulting in a system event.

MBScript example:

```
mbCommand("system", "clickSystemButton");
```

closeWindow

parameter: {system|liveLogic|waveformSource|protocol|logicSource|logicAnalyzer|pictureWindow|webWindow|videoWindow|all}

Closes tool windows.

MBScript example:

```
mbCommand("system", "closeWindow pictureWindow");
```

getCurrentPath

returns: default directory

Gets the directory that will be used if no explicit directory is provided.

MBScript example:

```
dir =  mbCommand("system", "getCurrentPath");
```

getRevision

returns: csv revision string

Gets the current revision string: software, firmware, hardware.

MBScript example:

```
rev =  mbCommand("system", "getRevision");
split(rev, ",", revs[]);
swRev = revs[0];
fwRev = revs[1];
hwRev = revs[2];
```

getSerialNumber

returns: serial number string

Gets the product serial number.

MBScript example:

```
serialNum =  mbCommand("system", "getSerialNumber");
```

minimizeWindow

parameter: {system|liveLogic|waveformSource|protocol|logicSource|logicAnalyzer|pictureWindow|webWindow|videoWindow}

Minimizes a tool window.

MBScript example:

```
mbCommand("system", "minimizeWindow pictureWindow");
```

openSystem

parameter: directory

Opens a previsously saved system file set in directory.

MBScript example:

```
mbCommand("system", "openSystem c:\\systemSave\\system1");
```

openWindow

parameter: {system|liveLogic|waveformSource|protocol|logicSource|logicAnalyzer|pictureWindow|webWindow|videoWindow}

Opens a tool window.

MBScript example:

```
mbCommand("system", "openWindow liveLogic");
```

positionWindow

parameter: {system|liveLogic|waveformSource|protocol|logicSource|logicAnalyzer|pictureWindow|webWindow|videoWindow}
parameter: x
parameter: y

Sets the location of a tool window.

MBScript example:

```
mbCommand("system", "positionWindow pictureWindow 200 300");
```

restoreWindow

parameter: {system|liveLogic|waveformSource|protocol|logicSource|logicAnalyzer|pictureWindow|webWindow|videoWindow}

Restores a minimized a tool window.

MBScript example:

```
mbCommand("system", "restoreWindow pictureWindow");
```

runProgram

parameter: full execution path

Runs an executable PC program.

MBScript example:

```
mbCommand("system", "runProgram c:\\myProgram.exe");
```

runScript

parameter: filename
optional parameter: directory

Runs a script file.

MBScript example:

```
mbCommand("system", "runScript myScript c:\\scriptDir");
```

saveSystem

parameter: directory

Saves system file set to directory.

MBScript example:

```
mbCommand("system", "saveSystem c:\\systemSave\\system1");
```

Section 3 FiveWire Tool Commands

Tool Commands directly control or return values from the FiveWire tools. Each tool includes its own set of commands.

Live Logic commands

Setup Commands

enableDecoder

eventIn

eventOut

logicFamily

mode

timeout

setInputThreshold

Trigger Commands

followedBy

triggerPosition

triggerPosition

triggerType

trigMaxDuration

trigMinDuration

Display Commands

centerAndScale

clearDecode

deleteAllMarks

deleteMark

leftAndScale

loadWaveform

refreshDecode

removeMarks
setCh1Measurement
setCH2Measurement
setMark
showView
zoomAll
zoomMarks

Execution and Readback Commands

getCaptureTime
getCH1Waveform
getCH2Waveform
getDVM
getMarks
getNumberOfSamples
getRunState
getTimeOfTrigger
getTriggerIndex
getTriggerState
getWaveformData
getWaveformDurations
getWaveform
getWaveformTimeFiltered
getWaveformTime
run
saveWaveform
stopOnAddressNACK

Complete Live Logic Command Reference

centerAndScale

parameter: float time to center
parameter: float time span of display

Adjusts the acquisition display center and time span to the specified values.

MBScript example:

```
// Center display at 100us and show 100us of data
mbCommand("liveLogic", "centerAndScale 10e-6 100e-6");
```

clearDecode

Clear decode removes all marks from the current displayed acuqisition.

MBScript example:

```
mbCommand("liveLogic", "clearDecode");
```

deleteAllMarks

Removes all marks from the acquisition display.

MBScript example:

```
mbCommand("liveLogic", "deleteAllMarks");
```

deleteMark

parameter: mark ID

Removes the mark from the acquisition display with the mark ID.

MBScript example:

```
mbCommand("liveLogic", "deleteMark @VALUE-3210:4.4640s");
```

enableDecoder

parameter: {true|false}

Enables or disables the loaded decoder. If disabled, decode marks will not be placed on the waveform when a new acquisition is displayed.

MBScript example:

```
mbCommand("liveLogic", "enableDecoder false");
```

eventIn

parameter: {none|system|waveformSource|protocolA|protocolB|logicSource|logicAnalyzer}

Selects the tool which provides an external trigger signal.

MBScript example:

```
mbCommand("liveLogic", "eventIn system");
```

eventOut

parameter: {start|trigger}

Determines when Live Logic generates its trigger output.

MBScript example:

```
mbCommand("liveLogic", "eventOut start");
```

followedBy

parameter: {X|1|0}{X|1|0} X = don't care

Followed by selects the logical value that will complete the trigger after the initial trigger has been satisfied.

MBScript example:

```
mbCommand("liveLogic", "followedBy 01");
```

getCaptureTime

Returns value: float time

Returns to total duration time of the current acquisition.

MBScript example:

```
captureTime = mbCommand("liveLogic", "getCaptureTime");
```

getCH1Waveform

Returns CSV list: samples

Returns a comma separated list of samples. Each sample contains the logical value of channel 1 and the time of the sample.

Format: "data1;time1,data2;time2,…,dataN;timeN"

MBScript example:

```
ch1Wfm = mbCommand("liveLogic", "getCH1Waveform");
size = split(ch1Wfm, ",", ch1Samples[]);
foreach(sample in ch1Samples[])
{
   split(sample, ";", values[]);
   data = values[0];
   time = values[1];
}
```

getCH2Waveform

Returns CSV list: samples

Returns a comma separated list of samples. Each sample contains the logical value of channel 2 and the time of the sample.

Format: "data1;time1,data2;time2,…,dataN;timeN"

MBScript example:

```
ch2Wfm = mbCommand("liveLogic", "getCH2Waveform");
```

getDVM

Returns CSV value: channel 1 voltage, channel 2 voltage

Reads the average voltage of both input channels and returns a pair comma separated float values.

Format: "ch1Volts,ch2Volts"

MBScript example:

```
dvmValues = mbCommand("liveLogic", "getDVM");
size = split(dvmValues, ",", dvmArray[]);
vCh1 = dvmArray[0];
vCh2 = dvmArray[1];
```

getMarks

Returns sc list: mark

Returns a list of currently displayed marks. Each mark in the list is separated by a semicolon. The mark has the following comma separated elements:

@<mark type>:<time of mark start>,
<time of mark end in nanoseconds>,
<mark label>

MBScript example:

```
marks = mbCommand("liveLogic", "getMarks");
split(marks, ",", markList[]);
foreach(mark in markList[])
{
   textWindow(mark);
}
```

getNumberOfSamples

Returns value: int number of samples

Returns the number of samples contained in the current acquisition.

MBScript example:

```
numSamples = mbCommand("liveLogic", "getNumberOfSamples");
```

getRunState

Returns value: {stopped|running}

Returns running if currently acquiring a new acquisition otherwise returns stopped.

MBScript example:

```
runState = mbCommand("liveLogic", "getRunState");
while(runState == "running")
   runState = mbCommand("liveLogic", "getRunState");
```

getTimeOfTrigger

Returns value: float time

Returns the time of the sample that satisfied the trigger.

MBScript example:

```
trigTime = mbCommand("liveLogic", "getTimeOfTrigger");
```

getTriggerIndex

Returns value: int index

Returns the zero based index of the sample that satisfied the trigger.

MBScript example:

```
trigIndex = mbCommand("liveLogic", "getTriggerIndex");
```

getTriggerState

Returns value: {0|1|3}

Returns the trigger state of a stopped acquisition. Returns true if the trigger was satisfied otherwise false.

0: Not triggered
1: First trigger value found but not second value
3: Trigger complete

MBScript example:

```
if (!toNum(mbCommand("liveLogic", "getTriggerState")))
   sleep(100);
```

getWaveformData

parameter: {1|2|3} mask
Returns CSV list: int data value

Returns a comma separated list of data values AND mask.

MBScript example:

```
wfmData = mbCommand("liveLogic", "getWaveformData");
```

getWaveformDurations

Returns CSV list: float duration

Returns a comma separated list of time duration for each sample.

MBScript example:

```
wfmDur = mbCommand("liveLogic", "getWaveformDurations");
```

getWaveform

Optional parameter: {1|2}
Returns CSV list: samples

Returns comma separated list of sample values. Each sample contains <ch1 val>;<ch2 val>;<time>. If the optional string is provided then only the channel selected is included in the sample value.

MBScript example:

```
wfm = mbCommand("liveLogic", "getWaveform 1");
size = split(wfm, ",", samples[]);
foreach(sample in samples[])
{
   split(sample, ";", vals[]);
   level = vals[0];
   time = vals[1]
}
```

getWaveformTimeFiltered

parameter: {CH1|CH2}
parameter: float minimum duration

Returns a comma separated list of samples for the selected channel. Only samples where the channel changes state are included. This excludes samples where the channel did not change (the other channel changed in these cases).

The sample list is also subjected to a glitch filter. If the state duration is less than the specified minimum duration the sample is excluded.

MBScript example:

```
wfmTime = mbCommand("liveLogic",
   "getWaveformTimeFiltered CH1 50e-9");
size = split(wfmTime, ",", times[]);
```

getWaveformTime

Returns CSV list: float time

Returns a comma separated list of the time each sample was taken.

MBScript example:

```
wfmTimes = mbCommand("liveLogic", "getWaveformTime");
size = split(wfmTimes, ",", timeArray[]);
```

leftAndScale

parameter: float time
parameter: float time span of display

Adjusts the acquisition display left and time span to the specified values.

MBScript example:

```
mbCommand("liveLogic", "leftAndScale 0 20e-6");
```

loadWaveform

parameter: filename
parameter: path

Loads a previously saved waveform and displays it. The file extension ".csv" is appended to the filename.

MBScript example:

```
mbCommand("liveLogic",
  "loadWaveform LLwaveform " + myDocuments);
```

LogicFamily

parameter: {CMSO_5V|CMOS_3.3V|CMOS_3V|CMOS2.5V|CMOS1.8V|TTL|custom}
optional parameter: float threshold voltage Only used with custom

Sets the logic threshold voltage.

MBScript example:

```
mbCommand("liveLogic", "logicFamily CMOS_3.3V");
```

mode

parameter: {continuous|single}

Sets the acquisition mode.

- continuous: If the trigger is not satisfied within the timeout period, the acquisition will be stopped and displayed, and a new acquisition will be started. This provides a live waveform display.
- single: Acquisitions are started manually. An acquisition will continue until the trigger is satisfied or the user terminates the acquisition. The acquisition will then be displayed.

MBScript example:

```
mbCommand("liveLogic", "mode single");
```

refreshDecode

Refresh decode updates all decode marks

MBScript example:

```
mbCommand("liveLogic", "refreshDecode");
```

removeMarks

Removes all marks from the current acquisition display.

MBScript example:

```
mbCommand("liveLogic", "removeMarks");
```

run

Starts an acquisition.

MBScript example:

```
mbCommand("liveLogic", "run");
```

samples

parameter: {32|64|128|256|512|1024|2048|<int 8..2047>}

Selects the total number of samples to be captured.

MBScript example:

```
mbCommand("liveLogic", "samples 512");
```

saveWaveform

parameter: filename
optional parameter: directory path

Saves the current acquisition and setup to the specified file name. The file extension ".csv" is automatically appended to the name. If the optional directory path is provided, the path is prefixed to the file name. Otherwise the file path will be the execution directory of the MBScript.

MBScript example:

```
mbCommand("liveLogic", "saveWaveform testWfm");
```

setCh1Measurement

parameter: {frequency|period|pulseWidth|duty|none}

Selects a measurement associated with the channel 1 signal. The measurement is displayed when an acquisition stops.

MBScript example:

```
mbCommand("liveLogic", "setCh1Measurement frequency");
```

setCh2Measurement

parameter: {frequency|period|pulseWidth|duty|none}

Selects a measurement associated with the channel 2 signal. The measurement is displayed when an acquisition stops.

MBScript example:

```
mbCommand("liveLogic", "setCh2Measurement duty");
```

setInputThreshold

parameter: <float threshold>

Sets the input threshold voltage

MBScript example:

```
mbCommand("liveLogic", "setInputThreshold 1.50");
```

setMark

parameter: float mark start time
parameter: float mark end time
parameter: {text|line|box|fixed} type of mark
parameter: mark text
parameter: int x offset for text
parameter: int y offset for text
parameter: mark ID
parameter: text color transparancy int value
parameter: text color red int value
parameter: text color green int value
parameter: text color blue int value
parameter: {true|false} mark persistence
parameter: mark parameter

Sets or replaces a mark on the display with the specified characteristics.

showView

parameter: view name

Zooms the current acquisition display to show a named view which was defined earlier.

MBScript example:

```
mbCommand("liveLogic", "showView mySavedView");
```

stop

Terminates an acquistion

MBScript example:

```
mbCommand("liveLogic", "stop");
```

timeout

parameter: {0.1|1|2|5|10|none|<float>}

If the timeout is not set to none, the timeout will terminate an acquisition after the timeout has expired.

MBScript example:

```
mbCommand("liveLogic", "timeout 30");
```

trigger

parameter: <ch2><ch1> with values {X|1|0} X = don't care

Trigger selects the logical value that will satisfy the trigger. Once the trigger is satisified and sufficient samples are acquired to position the trigger, the acquisition is terminated and the acquisition displayed.

Note: The trigger may also depend on other trigger parameters: followedBy, trigMinDuration, trigMaxDuration, triggerPosition,triggerType, timeout

MBScript example:

```
mbCommand("liveLogic", "trigger 1X");
```

triggerPosition

parameter: {start|middle|end}

After the trigger condition is satisified additional samples will be acquired to set the position of the trigger. For example, if the trigger is to be placed in the middle of the acquisition then the number of samples that follow the trigger must be half as many as the total number of samples requested.

MBScript example:

```
mbCommand("liveLogic", "triggerPostion Start");
```

triggerType

parameter: {level|duration|pulse|external}

The trigger type selects the kind of trigger used.

- *level*: Looks for a sequence of logical values
- *duration*: Like level but sets a minimum time duration on the value before it is recognized
- *pulse*: Like Level but sets and minimum and a maximum duration on the value before it is recognized
- *external*: Accepts a trigger produced by one of the other tools

MBScript example:

```
mbCommand("liveLogic", "trigMinDuration 100e-9");
mbCommand("liveLogic", "triggerType Duration");
```

trigMaxDuration

parameter: <float time> [20e-9 to 20e-6]

When the trigger mode is set to Pulse, the acquired state must be less than the specified time to be recognized.

MBScript example:

```
mbCommand("liveLogic", "trigMaxDuration 1e-6");
```

trigMinDuration

parameter: <float time> [20e-9 to 10e-3]

When the trigger mode is set to Duration or Pulse, the acquired state must be equal to or greater than the specified time to be recognized.

MBScript example:

```
mbCommand("liveLogic", "trigMinDuration 100e-9");
```

zoomAll

Zooms the current acquisition display to show all samples.

MBScript example:

```
mbCommand("liveLogic", "zoomAll");
```

zoomMarks

Zooms the current aquisition display to show the range of the A and B time marks.

MBScript example:

```
mbCommand("liveLogic", "zoomMarks");
```

Waveform Source commands

Example:

```
mbCommand("waveformSource", "mode continuous");
```

Setup Commands

autoGenerate

eventIn

eventOut

fastMode

loadWaveform

mode

selectFastMode

threePoint

maintainOutputOnStop

Waveform Synthesizer Commands

batteryDischargeTime

batteryNumCells

batteryType

generateWaveform

pulseDuration

pulseInvert

pulse

pulsePeriod

rcInvert

rc

rcTimeConstant

selectWaveformType

sineAmplitude

sineFrequency

sineOffset

supplyVolts

triangleDutyCycle

triangle

trianglePeriod

Custom Waveform Commands

setJump

setMode

setValue

Execution and Display Commands

getRunState

run

stop

updateWaveformDisplay

Complete Waveform Source Command Reference

autoGenerate

parameter: {true|false}

If set to true, the waveform will be regenerated and loaded each time the pattern is run. This can be set to false to avoid start up delay once the pattern is loaded

MBScript example:

```
mbCommand("waveformSource", "autoGenerate false");
```

batteryDischargeTime

parameter: <time> float [100e-6 to 850]

Selects the battery discharge time to simulate.

MBScript example:

```
mbCommand("waveformSource", "batteryDischargeTime 600");
```

batteryNumCells

parameter: count [1 (LiO, LiPo), 1-3(Aliline, NiCd, NiMH) limited by 5V max output]

Selects the number of series battery cells to simulate.

MBScript example:

```
mbCommand("waveformSource", "batteryNumCells 2");
```

batteryType

parameter: {Alkiline|NiCd|LiO|NiMH|LiPo}

Selects the battery chemistry to be simulated.

MBScript example:

```
mbCommand("waveformSource", "batteryType NiMH");
```

eventIn

parameter: {none|logicSource|protocolA|protocolB|liveLogic|logicAnalyzer|system}

Selects the source of the input event.

MBScript example:

```
mbCommand("waveformSource", "eventIn logicSource");
```

eventOut

parameter: {start|end}

Determines if the event output is sent at the start or end of the waveform.

MBScript example:

```
mbCommand("waveformSource", "eventOut end");
```

fastMode

parameter: {true|false}

Fast mode speeds up the clock rate providing higher time resolution. Disabling fast mode provides longer pattern times.

MBScript example:

```
mbCommand("waveformSource", "fastMode true");
```

generateWaveform

Run the waveform synthesizer to create a new waveform.

MBScript example:

```
mbCommand("waveformSource", "generateWaveform");
```

getRunState

returns: {stopped|running}

Determine if the pattern is executing.

MBScript example:

```
runState = mbCommand("waveformSource", "getRunState");
```

loadWaveform

parameter: filename
parameter: directory

Loads a previously stored pattern file. A .csv suffix is automatically added to the filename.

MBScript example:

```
mbCommand("waveformSource",
  "loadWaveform test myDocuments");
```

maintainOutputOnStop

parameter: {true|false}

If set to true the output will remain at the output value of the last vector executed.

MBScript example:

```
mbCommand("waveformSource", "maintainOutputOnStop false");
```

mode

parameter: {continuous|onEvent|single}

Sets the execution mode for the waveform:

- continuous - pattern loops infinately
- onEvent - pattern executes each time an event is received
- single - pattern executes once

MBScript example:

```
mbCommand("waveformSource", "mode onEvent");
```

pulseDuration

parameter: <time> float [10us to 800 seconds]

Sets the pulse duration time.

MBScript example:

```
mbCommand("waveformSource", "pulseDuration 100e-6");
```

pulseInvert

parameter: {true|false}

Inverts the polarity of the pulse.

MBScript example:

```
mbCommand("waveformSource", "pulseInvert false");
```

pulse

parameter: {VH|VL}
parameter: <voltage> float [0 to 5]

Sets the highest or lowest value for the waveform

MBScript example:

```
mbCommand("waveformSource", "pulse VH 3.3");
mbCommand("waveformSource", "pulse VL 0.5");
```

pulsePeriod

parameter: <time> float [5us to 800 seconds and less than pulse duration]

Sets the pulse period.

MBScript example:

```
mbCommand("waveformSource", "pulsePeriod 1e-3");
```

rcInvert

parameter: {true|false}

Inverts the RC waveform.

MBScript example:

```
mbCommand("waveformSource", "rcInvert true");
```

rc

parameter: {VH|VL}
parameter: <voltage> float [0 to 5]

Sets the highest or lowest value for the waveform

MBScript example:

```
mbCommand("waveformSource", "rc VH 3.3");
mbCommand("waveformSource", "rc VL 0.5");
```

rcTimeConstant

parameter: <time> float [10us to 160 seconds]

Sets the time constant for the RC waveform. Five time constants are generated.

MBScript example:

```
mbCommand("waveformSource", "rcTimeConstant 0.1e-3");
```

run

Start pattern execution.

MBScript example:

```
mbCommand("waveformSource", "run");
```

selectFastMode

parameter: {true|false}

Select fast mode execution

MBScript example:

```
mbCommand("waveformSource", "selectFastMode true");
```

selectWaveformType

parameter: {triangle|square|rc|battery|dc|sine|custom}

Configures the waveform synthesizer to create the selected waveform.

MBScript example:

```
mbCommand("waveformSource", "selectWaveformType rc");
```

setJump

parameter: {jumpA|jumpB|jumpC}
parameter: {always|never|event|notEvent|wait|loop|loopInner}
parameter: jump target index [0 to 1019]
optional parameter: loop count [1 to 255] required for loop commands

Sets the function for the selected jump controller:

- always - always jump to the target index
- never - jump to the following index
- event - jump if an event has been received
- notEvent - jump if an event has not been received
- wait - stay on the current index until an event is received
- loop - jump until the loop counter is zero
- loopInner - jump until the loop counter is zero and reload count at termination

Configure the jump controllers

MBScript example:

```
mbCommand("waveformSource", "mode");
```

setMode

parameter: line index [0 to 1019]
parameter: {stop|next|jumpA|jumpB|jumpC|event}

Sets the mode for the selected vector:

- stop - stops pattern sequence execution
- next - jumps to the following vector when this vector duration is finished
- jumpA, jumpB, jumpC - the next executing vector is determined by the jump controller
- event - same as next except an event is generated during the vector

MBScript example:

```
mbCommand("waveformSource", "setMode 8 event");
```

setValue

parameter: line index [0 to 1019]
parameter: voltage [0 to 5]
parameter: float vector time duration [1us to 1.0 seconds]

Overwrites the pattern vector at index with new data and duration values.

MBScript example:

```
mbCommand("waveformSource", "setValue 2.0 1e-3");
```

sineAmplitude

parameter: volts float [0 to 5]

Sets the peak amplitude of the sine wave

MBScript example:

```
mbCommand("waveformSource", "sineAmplitude 1.0");
```

sineFrequency

parameter: frequency float [0.002 to 50e3]

Sets the frequency of the sine wave.

MBScript example:

```
mbCommand("waveformSource", "sineFrequency 10e3");
```

sineOffset

parameter: volts float [0 to 5]

Sets the offset of the sine wave

MBScript example:

```
mbCommand("waveformSource", "sineOffset 2.5");
```

stop

Stop pattern execution.

MBScript example:

```
mbCommand("waveformSource", "stop");
```

supplyVolts

parameter: volts float [0 to 5]

Selects the output voltage

MBScript example:

```
mbCommand("waveformSource", "supplyVolts 3.3");
```

threePoint
parameter: {true|false}

The threePoint parameter determines how the waveform display is rendered. It should be set true for discontinuous pulses and false for smooth waveforms.

MBScript example:

```
mbCommand("waveformSource", "threePoint false");
```

triangleDutyCycle
parameter: duty float [0 to 100]

Sets the duty cycle of the triangle waveform

MBScript example:

```
mbCommand("waveformSource", "triangleDutyCycle 50");
```

triangle
parameter: {VH|VL}
parameter: voltage float [0 to 5]

Sets the highest or lowest value for the waveform

MBScript example:

```
mbCommand("waveformSource", "triangle VH 3.3");
mbCommand("waveformSource", "triangle VL 0.5");
```

trianglePeriod
parameter: time float [20us to 850 seconds]

Sets the waveform time period

MBScript example:

```
mbCommand("waveformSource", "trianglePeriod 1e-3");
```

updateWaveformDisplay

parameter: time scale
parameter: three point

Refresh the waveform display

MBScript example:

```
mbCommand("waveformSource",
  "updateWaveformDisplay 1 true");
```

Logic Analyzer commands

Setup Commands

enableDecoder

eventIn

eventOut

loadWaveform

logicFamily

mode

samples

timeout

Trigger Commands

trigger

triggerPosition

Display Commands

centerAndScale

channelColor

channelLabel

deleteAllMarks

deleteMark

displayOrder

displayType

leftAndScale

showView

zoomAll

zoomMarks

Control and Readback Commands

getCaptureTime

getMarks
getNumberOfSamples
getRunState
getTimeOfTrigger
getTriggerIndex
getTriggerState
getWaveformData
getWaveformDurations
getWaveform
getWaveformTime
run
saveWaveform
stop

Complete Logic Analyzer Command Reference

centerAndScale

parameter: float time to center
parameter: float time span of display

Adjusts the acquisition display center and time spaln to the specified values.

MBScript example:

```
// Center display at 100us and show 100us of data
mbCommand("logicAnalyzer", "centerAndScale 10e-6 100e-6");
```

channelColor

parameter: {0|1|2|3|4|5|6|7|8} channel
parameter: transparency [0 to 255]
parameter: red [0 to 255]
parameter: green [0 to 255]
parameter: blue [0 to 255]

Sets the selected channels color

MBScript example:

```
mbCommand("logicAnalyzer", "channelColor 255 255 0 0");
```

channelLabel

parameter: {0|1|2|3|4|5|6|7|8} channel
parameter: label

Sets the selected channels label text

MBScript example:

```
mbCommand("logicAnalyzer", "channelLabel CLK");
```

deleteAllMarks

Removes all marks from the acquisition display.

MBScript example:

```
mbCommand("logicAnalyzer", "deleteAllMarks");
```

deleteMark

parameter: mark ID

Removes the mark from the acquisition display with the mark ID.

MBScript example:

```
mbCommand("logicAnalyzer",
  "deleteMark @VALUE-3210:4.4640s");
```

displayOrder

parameter: display order string

Sets the order which the channels are displayed. The order string may contain any sequence of numbers 0..9 and period characters (which inserts a blank line).

MBScript example:

```
mbCommand("logicAnalyzer", "displayOrder 312.0");
```

displayType

parameter: {timing|state}

Sets the type of display shown, timing diagram or state table.

MBScript example:

```
mbCommand("logicAnalyzer", "displayType state");
```

enableDecoder

parameter: {true|false}

Enables or disables the loaded decoder. If disabled, decode marks will not be placed on the waveform when a new acquisition is displayed.

MBScript example:

```
mbCommand("logicAnalyzer", "enableDecoder false");
```

eventIn

parameter: {none|system|liveLogic|waveformSource|protocolA|protocolB|logicSource}

Selects the tool which provides an external trigger signal.

MBScript example:

```
mbCommand("logicAnalyzer", "eventIn system");
```

eventOut

parameter: {start|trigger}

Determines when Live Logic generates its trigger output.

MBScript example:

```
mbCommand("logicAnalyzer", "eventOut start");
```

getCaptureTime

Returns value: float time

Returns to total duration time of the current acquisition.

MBScript example:

```
cTime = mbCommand("logicAnalyzer", "getCaptureTime");
```

getMarks

Returns list: mark

Returns a list of currently displayed marks. Each mark in the list is separated by a semicolon. The mark has the following comma separated elements:

@<mark type>:<time of mark start>,
<time of mark end>,
<mark label>

MBScript example:

```
marks = mbCommand("logicAnalyzer", "getMarks");
split(marks, ",", markList[]);
foreach(mark in markList[])
{
   m = mark;
}
```

getNumberOfSamples

Returns value: int number of samples

Returns the number of samples contained in the current acquisition.

MBScript example:

```
nSamp = mbCommand("logicAnalyzer", "getNumberOfSamples");
```

getRunState

Returns value: {stopped|running}

Returns running if currently aquiring a new acquisition otherwise returns stopped.

MBScript example:

```
runState = mbCommand("logicAnalyzer", "getRunState");
while(runState == "running")
   runState = mbCommand("logicAnalyzer", "getRunState");
```

getTimeOfTrigger

Returns value: float time

Returns the time of the sample that satisfied the trigger.

MBScript example:

```
trigTime = mbCommand("logicAnalyzer", "getTimeOfTrigger");
```

getTriggerIndex

Returns value: int index

Returns the zero based index of the sample that satisfied the trigger.

MBScript example:

```
trigIndex = mbCommand("logicAnalyzer", "getTriggerIndex");
```

getTriggerState

Returns value: {0|1|2|3}

Returns the trigger state:
0: Not triggered
1: Trigger A received, waiting for trigger B
2: Trigger A and B recevied, waiting for trigger C
3: Triggered

MBScript example:

```
if (3!= toNum(mbCommand("logicAnalyzer", "getTriggerState")))
   sleep(100);
```

getWaveformData

parameter: bit mask [1 to 511]
Returns CSV list: int data value

Returns a comma separated list of data values AND mask.

MBScript example:

```
wfmData = mbCommand("logicAnalyzer", "getWaveformData 2");
```

getWaveformDurations

Returns CSV list: float duration

Returns a comma separated list of time duration for each sample.

MBScript example:

```
dur = mbCommand("logicAnalyzer", "getWaveformDurations");
```

getWaveform

Optional parameter: {<channel list ie. 210>}
Returns CSV list: samples

Returns comma separated list of sample values. Each sample contains <data>;<time>. If the optional string is provided then only the channel selected is included in the sample value.

MBScript example:

```
wfm = mbCommand("liveLogic", "getWaveform 210");
size = split(wfm, ",", samples[]);
foreach(sample in samples[])
{
   split(sample, ";", vals[]);
   level = vals[0];
   time = vals[1]
}
```

getWaveformTime

Returns CSV list: float time

Returns a comma separated list of the time each sample was taken.

MBScript example:

```
wfmTimes = mbCommand("logicAnalyzer", "getWaveformTime");
size = split(wfmTimes, ",", timeArray[]);
```

leftAndScale

parameter: float time
parameter: float time span of display

Adjusts the acquisition display left and time span to the specified values.

MBScript example:

```
mbCommand("logicAnalyzer", "leftAndScale 0 20e-6");
```

loadWaveform

parameter: filename
parameter: path

Loads a previously saved waveform and displays it. The file extension ".csv" is appended to the filename.

MBScript example:

```
mbCommand("logicAnalyzer",
  "loadWaveform LAwaveform " + myDocuments + "test");
```

logicFamily

parameter: {CMSO_5V|CMOS_3.3V|CMOS_3V|CMOS2.5V|CMOS1.8V|TTL|custom}
optional parameter: float threshold voltage [0 to 5] Only used with custom

Sets the logic threshold voltage.

MBScript example:

```
mbCommand("logicAnalyzer", "logicFamily CMOS_3.3V");
```

mode

parameter: {continuous|single}

Sets the acquisition mode.

- Continuous: If the trigger is not satisfied within the timeout period, the acquisition will be stopped and displayed, and a new acquisition will be started. This provides a live waveform display.
- Single: Acquisitions are started manually. An acquisition will continue until the trigger is satisfied or the user terminates the acquisition. The acquisition will then be displayed.

MBScript example:

```
mbCommand("logicAnalyzer", "mode single");
```

run

Starts an acquisition.

MBScript example:

```
mbCommand("logicAnalyzer", "run");
```

samples

parameter: {32|64|128|256|512|1024|2048}

Selects the total number of samples to be captured.

MBScript example:

```
mbCommand("logicAnalyzer", "samples 512");
```

saveWaveform

parameter: filename
optional parameter: directory path

Saves the current acquisition and setup to the specified file name. The file extension ".csv" is autmatically appeded to the name. If the optional directory path is provided, the path is prefixed to the file name. Otherwise the file path will be the execution directory of the MBScript.

MBScript example:

```
mbCommand("logicAnalyzer", "saveWaveform testWfm");
```

setMark

parameter: float mark start time
parameter: float mark end time
parameter: {text|line|box|fixed} type of mark
parameter: mark text
parameter: int x offset for text
parameter: int y offset for text
parameter: mark ID
parameter: text color transparancy int value
parameter: text color red int value
parameter: text color green int value
parameter: text color blue int value
parameter: {true|false} mark persistence
parameter: mark parameter

Sets or replaces a mark on the display with the specified characteristics.

showView

parameter: view name

Zooms the current acquisition display to show a named view which was defined earlier.

MBScript example:

```
mbCommand("logicAnalyzer", "showView mySavedView");
```

stop

Terminates an acquistion

MBScript example:

```
mbCommand("logicAnalyzer", "stop");
```

timeout

parameter: {0.1|1|2|5|10|none|<float>}

If the timeout is not set to none, the timeout will terminate an acquisition after the timeout has expired.

MBScript example:

```
mbCommand("logicAnayzer", "timeout 30");
```

trigger

Parameter: {A|B|C}
parameter: {value|type|minimumTime|maximumTime} command type
parameter: based on command type
value: trigger value
type: {none|value|duration|range|window|external| immediate}
minimumTime: float time
maximumTime: float time

Set the A,B, or C trigger parameters

MBScript example:

```
mbCommand("logicAnayzer", "A value XXXXXXXX1");
mbCommand("logicAnayzer", "A type window");
mbCommand("logicAnayzer", "A minimumTime 400e-9");
mbCommand("logicAnayzer", "A maximumTime 600e-9");
```

triggerPosition

parameter: {Start|Middle|End}

After the trigger condition is satisified additional samples will be acquired to set the position of the trigger. For example, if the trigger is to be placed in the middle of the acquisition then the number of samples that follow the trigger must be half as many as the total number of samples requested.

MBScript example:

```
mbCommand("logicAnalyzer", "triggerPostion Start");
```

zoomAll

Zooms the current acquisition display to show all samples.

MBScript example:

```
mbCommand("logicAnalyzer", "zoomAll");
```

zoomMarks

Zooms the current aquisition display to show the range of the A and B time marks.

MBScript example:

```
mbCommand("logicAnayzer", "zoomMarks");
```

Logic Source commands

Setup Commands

eventIn

mode

Vhigh

Vlow

Data Pattern Commands

eventIn

loadPattern

manual

setJump

setMode

setValue

setVertorMasked

setVector

Execution and Readback Commands

getRunState

getVector

loadPatternMemory

run

stop

Complete Logic Source Command Reference

eventIn

parameter: {none|waveformSource|protocolA|protocolB|liveLogic|logicAnalyzer|system}

Selects the source for input events.

MBScript example:

```
mbCommand("logicSource", "eventIn system");
```

getRunState

returns: {stopped|running}

Determine if the pattern is executing.

MBScript example:

```
runState = mbCommand("logicSource", "runState");
```

getVector

parameter: line index
returns: decimal data value

Gets the data value at the selected line.

MBScript example:

```
vect = mbCommand("logicSource", "getVector 4");
```

loadPatternMemory

Update the hardware memory

MBScript example:

```
mbCommand("logicSource", "loadPatternMemory");
```

loadPattern

parameter: filename
parameter: directory path

Loads a previously stored pattern file. The extension .csv is automatically appended to the file name.

MBScript example:

```
mbCommand("logicSource",
  "loadPattern testPat " + myDocuments);
```

manual

parameter: {values|momentary|invert}
parameter: {<logic values>}

Sets the manual output levels. Logic values formats:

- false;true;true;true;true;false;false;false;false bool array
- 0xF0 Hexadecimal
- 250 Decimal

MBScript example:

```
mbCommand("logicSource", "manual values 0x1FF");
```

mode

parameter: {continuous|onEvent|single|custom|manual}

Set the type of pattern that will be used

MBScript example:

```
mbCommand("logicSource", "mode single");
```

run

Start pattern execution

MBScript example:

```
mbCommand("logicSource", "run");
```

setJump

parameter: {jumpA|jumpB|jumpC}
parameter: {always|never|event|notEvent|wait|loop|loopInner}
parameter: jump target index [0 to 1019]
optional parameter: loop count [1 to 255] required for loop commands

Sets the function for the selected jump controller:

- always - always jump to the target index
- never - jump to the following index
- event - jump if an event has been received
- notEvent - jump if an event has not been received
- wait - stay on the current index until an event is received
- loop - jump until the loop counter is zero
- loopInner - jump until the loop counter is zero and reload count at termination

Configure the jump controllers

MBScript example:

```
mbCommand("logicSource", "setJump jumpA always 0");
```

setMode

parameter: line index [0 to 1019]
parameter: {stop|next|jumpA|jumpB|jumpC|event}

Sets the mode for the selected vector:

- stop - stops pattern sequence execution
- next - jumps to the following vector when this vector duration is finished
- jumpA, jumpB, jumpC - the next executing vector is determined by the jump controller
- event - same as next except an event is generated during the vector

MBScript example:

```
mbCommand("logicSource", "setMode 8 event");
```

setValue

parameter: line index [0 to 1019]
parameter: data value [0 to 511]
parameter: float vector time duration [30e-9 to 40e-3]

Overwrites the pattern vector at index with new data and duration values.

MBScript example:

```
mbCommand("logicSource", "setValue 0 16 1e-6");
```

setVectorMasked

parameter: line index [0 to 1019]
parameter: decimal data value [0 to 511]
parameter: mask [0 to 511]
parameter: duration [30e-9 to 40e-3]

Updates the selected vector bits which are included in the mask leaving the remaining bits unchanged and updates the vector duration.

MBScript example:

```
mbCommand("logicSource", "setVectorMasked 0 255 8 1e-6");
```

setVector

parameter: line index [0 to 1019]
parameter: decimal data value [0 to 511]

Modifies the selected vector value but does not change its duration.

MBScript example:

```
mbCommand("logicSource", "setVector 0 16");
```

stop

Stop pattern execution

MBScript example:

```
mbCommand("logicSource", "stop");
```

updatePatternDisplay

Refresh the pattern display.

MBScript example:

```
mbCommand("logicSource", "updatePatternDisplay");
```

Vhigh

parameter: float high logic level [0 to 4.5]

Sets the high logic level

MBScript example:

```
mbCommand("logicSource", "Vhigh 3.3");
```

Vlow

parameter: float low logic level [0 to 4.5]

Sets the low logic voltage

MBScript example:

```
mbCommand("logicSource", "Vlow 0.0");
```

SPI Protocol commands

Setup Commands

bitOrder

clockInvert

clockRate

commandSet

dataSize

eventIn

inputThreshold

mode

selectInputEnable

tool

Vhigh

Vlow

Data Pattern Commands

getSdiData

setCommand

setComment

setSdoData

Trigger Commands

skipJumpOnTrigger

trig0

trig1

trig2

trig3

trigEnabled

Execution and Readback Commands

getRunState

loadFile

restart

run

stop

Complete SPI Protocol Command Reference

bitOrder

parameter: {msbFirst|lsbFirst}

Determines which is the first bit sent either the most significant or the least significant bit.

MBScript example:

```
mbCommand("protocol", "bitOrder msbFirst");
```

clockInvert

parameter: {true|false}

Inverts the clock output

MBScript example:

```
mbCommand("protocol", "clockInvert false");
```

clockRate

parameter: {10KHz|20KHz|50KHz||100KHz|200KHz|500KHz|1MHZ|2MHz|5MHz|10MHz|20MHz}

Sets the bit rate when a master.

MBScript example:

```
mbCommand("protocol", "clockRate 1MHz");
```

commandSet

parameter: {i2c|spi|trigger}

Can override the currently selected tool if no probe is installed.

MBScript example:

```
mbCommand("protocol", "commandSet spi");
```

dataSize

parameter: {8|16|24|32}

Sets the number of bits in each transaction.

MBScript example:

```
mbCommand("protocol", "dataSize 8");
```

eventIn

parameter: {none|logicSource|waveformSource|liveLogic|logicAnalyzer|system}

Selects the trigger input source

MBScript example:

```
mbCommand("protocol", "eventIn system");
```

getRunState

returns: {stopped|running}

Returns the execution state of the tool.

MBScript example:

```
state = mbCommand("protocol", "getRunState");
```

getSdiData

parameter: line number

Returns the SDI data at the line number or -1 if data at that line has not been captured.

MBScript example:

```
state = mbCommand("protocol", "getSdiData 1");
```

loadFile

parameter: filename
optional parameter: directory

Loads a previously stored pattern file.

MBScript example:

```
mbCommand("protocol", "loadFile test");
```

mode

parameter: {slave|onEvent|single|continuous}

Sets the mode of SPI execution

MBScript example:

```
mbCommand("protocol", "mode slave");
```

restart

Restarts tool execution

MBScript example:

```
mbCommand("protocol", "restart");
```

run

Starts tool execution

MBScript example:

```
mbCommand("protocol", "run");
```

selectInputEnable

parameter: {true, false}

Enables the select input line when in slave mode.

MBScript example:

```
mbCommand("protocol", "selectInputEnable true");
```

setCommand

parameter: line that command will be set on

parameter: {gotoZero, jumpToAddress, stop, generateEvent}

parameter *jumpToAddress only*: target line

Sets a command on the selected line number. Commands may only be applied to one line.

gotoZero: After this line, execution will proceed at line 0

jumpToAddress: After this line, execution will proceed at the target line. If triggering is enabled and has been satisfied, execution will fall through to the next line.

stop: Execution will stop on this line

generateEvent: The output event will be produced when this line is executed

MBScript example:

```
mbCommand("protocol", "setCommand 7 gotoZero");
```

setComment

parameter: line that comment will be set on

parameter: comment text

Sets a comment text on the selected line number. Note: surround the comment with single quotes if it contains a space character or comma.

MBScript example:

```
mbCommand("protocol", "setComment ‘start of test’");
```

setSdoData

parameter: line that data will be set on

parameter: SDO data

Sets the SDO data on the selected line number.

MBScript example:

```
mbCommand("protocol", "setSdoData 0 128");
```

skipJumpOnTrigger

parameter: {true, false}

Disables the jump function when a trigger has been received.

MBScript example:

```
mbCommand("protocol", "skipJumpOnTrigger true");
```

stop

Stops tool execution.

MBScript example:

```
mbCommand("protocol", "stop");
```

tool

returns: {i2c|spi|trigger}

Returns the currently selected protocol tool.

MBScript example:

```
tool = mbCommand("protocol", "tool");
```

trig0

parameter: {0|1|X} *1 to 8 digits*

Sets the binary value of trigger 0.

MBScript example:

```
mbCommand("protocol", "trig0 XXXX0001");
```

trig1 {0, 1, X} *1 to 8 digits*

parameter: {0|1|X} *1 to 8 digits*

Sets the binary value of trigger 1.

MBScript example:

```
mbCommand("protocol", "trig1 00000001");
```

trig2 {0, 1, X} *1 to 8 digits*

parameter: {0|1|X} *1 to 8 digits*

Sets the binary value of trigger 2.

MBScript example:

```
mbCommand("protocol", "trig2 XXXXXXXX");
```

trig3 {0, 1, X} *1 to 8 digits*

parameter: {0|1|X} *1 to 8 digits*

Sets the binary value of trigger 3.

MBScript example:

```
mbCommand("protocol", "trig3 XXXXXXXX");
```

trigEnabled

parameter: {true|false}

Enables the trigger function

MBScript example:

```
mbCommand("protocol", "trigEnabled true");
```

Vhigh

parameter: voltage float [0 to 4.5]

Sets the output logic high voltage.

MBScript example:

```
mbCommand("protocol", "Vhigh 3.3");
```

Vlow

parameter: voltage float [0 to 4.5]

Sets the output logic low voltage.

MBScript example:

```
mbCommand("protocol", "Vlow 0");
```

Trigger I/O Protocol tool commands

Trigger Output Setup Commands

out1Source

out2Source

out3Source

out1Invert

out2Invert

out3Invert

Vhigh

Vlow

Trigger Input Setup Commands

protASource

protBSource

inputThreshold

Complete Trigger Protocol Command Reference

inputThreshold

parameter: threshold float [0 to 5]

Sets the input logic threshold.

MBScript example:

```
mbCommand("protocol", "inputThreshold 1.67");
```

out1Invert

parameter: {true|false}

Invert the polarity of output 1

MBScript example:

```
mbCommand("protocol", "out1Invert true");
```

out1Source

parameter: {none|logicSource|waveformSource|liveLogic|logicAnalyzer|system}

Selects the protocol 1 output source

MBScript example:

```
mbCommand("protocol", "out1Source system");
```

out2Invert

parameter: {true|false}

Invert the polarity of output 2

MBScript example:

```
mbCommand("protocol", "out2Invert true");
```

out2Source

parameter: {none|logicSource|waveformSource|liveLogic|logicAnalyzer|system}

Selects the protocol 2 output source

MBScript example:

```
mbCommand("protocol", "out2Source system");
```

out3Invert

parameter: {true|false}

Invert the polarity of output 3

MBScript example:

```
mbCommand("protocol", "out3Invert true");
```

out3Source

parameter: {none|logicSource|waveformSource|liveLogic|logicAnalyzer|system}

Selects the protocol 3 output source

MBScript example:

```
mbCommand("protocol", "out3Source system");
```

protASource

parameter: {none|trig1|trig2|trig3}

Selects the protocol A trigger input source

MBScript example:

```
mbCommand("protocol", "protASource trig1");
```

protBSource

parameter: {trig1|trig2|trig3|1or2or3|1and2and3|1or2andNot3|1and2andNot3}

Selects the protocol B trigger input source

MBScript example:

```
mbCommand("protocol", "protBSource 1or2or3");
```

Vhigh

parameter: voltage float [0 to 4.5]

Sets the output logic high voltage.

MBScript example:

```
mbCommand("protocol", "Vhigh 3.3");
```

Vlow

parameter: voltage float [0 to 4.5]

Sets the output logic low voltage.

MBScript example:

```
mbCommand("protocol", "Vlow 0");
```

I2C Protocol commands

Setup Commands

eventIn

inputThreshold

masterClock

mode

slaveAddress

slaveRunMode

stopOnAddressNACK

stopOnTimeout

Display Commands

boxLine

showStartStop

showRestart

inputThreshold

Trace Commands

traceTriggerType

traceTriggerAddress

traceTriggerDataOffset

traceTriggerData

traceTriggerStopOnNACK

Execute and Readback Commands

getReadData

setWriteData

loadFile

run

stop

getRunState

Complete Trigger Protocol Command Reference

boxLine

parameter: line integer

Selects the line that will be boxed

MBScript example:

```
mbCommand("protocol", "boxLine 3");
```

eventIn

parameter: {none|logicSource|waveformSource|liveLogic|logicAnalyzer|system}

Selects the trigger input source

MBScript example:

```
mbCommand("protocol", "eventIn system");
```

getReadData

parameter: line integer

Returns: CSV list of values

Gets the captured read data in master mode. The line must be set to a read line before this command is executed.

MBScript example:

```
data = mbCommand("protocol", "getReadData 3");
```

getRunState

returns: {stopped|running}

Returns the execution state of the tool.

MBScript example:

```
state = mbCommand("protocol", "getRunState");
```

inputThreshold

parameter: threshold float [0 to 5]

Sets the input logic threshold.

MBScript example:

```
mbCommand("protocol", "inputThreshold 1.67");
```

loadFile

parameter: filename

optional parameter: directory

Loads a previously stored pattern file.

MBScript example:

```
mbCommand("protocol", "loadFile test");
```

masterClock

parameter: {20KHz|50KHz|100KHz|400KHz}

Selects the master clock rate

MBScript example:

```
mbCommand("protocol", "masterClock 100KHz");
```

mode

parameter: {master|slave|trace}

Sets the mode of I2C execution

MBScript example:

```
mbCommand("protocol", "mode master");
```

run

Starts tool execution

MBScript example:

```
mbCommand("protocol", "run");
```

setWriteData

parameter: line [0 to 63]

parameter: data 0 decimal

optional parameter: data 1 decimal

optional parameter: data 2 decimal

optional parameter: data 3 decimal

optional parameter: data 4 decimal

optional parameter: data 5 decimal

optional parameter: data 6 decimal

Set the data that will be written in write transactions in master mode. The line must be defined as a write before calling this command.

MBScript example:

```
mbCommand("protocol", "setWriteData 10 5 0 0 0 0 0 0");
```

showRestart
parameter: {true|false}

Display restart transactions

MBScript example:

```
mbCommand("protocol", "showRestart true");
```

showStartStop
parameter: {true|false}

Display start and stop transactions

MBScript example:

```
mbCommand("protocol", "showStartStop true");
```

slaveAddress
parameter: <X|hex digit><X|hex digit>

Selects the address that the slave will respond to

MBScript example:

```
mbCommand("protocol", "slaveAddress 5F");
```

slaveReadData

parameter: data 0 decimal

parameter: data 1 decimal

parameter: data 2 decimal

parameter: data 3 decimal

parameter: data 4 decimal

parameter: data 5 decimal

parameter: data 6 decimal

parameter: data 7 decimal

Set the read data

MBScript example:

```
mbCommand("protocol", "slaveReadData 10 5 0 0 0 0 0 0");
```

slaveRunMode

parameter: {single|stopOnEvent}

Selects the slave run mode

MBScript example:

```
mbCommand("protocol", "slaveRunMode single");
```

stopOnAddressNACK

parameter: {true|false}

Stop if an address access returns NACK

MBScript example:

```
mbCommand("protocol", "stopOnAddressNACK true");
```

stopOnTimeout
parameter: {true|false}

Stop if an address access time exceeds the timeout value

MBScript example:

```
mbCommand("protocol", "stopOnTimeout true");
```

stop

Stops tool execution.

MBScript example:

```
mbCommand("protocol", "stop");
```

traceTriggerAddress
parameter: <X|hex digit><X|hex digit>

Set the address of the trace trigger

MBScript example:

```
mbCommand("protocol", "traceTriggerAddress 4C");
```

traceTriggerDataOffset
parameter: offset [0 to 7]

Select which data byte will be used in trace trigger

MBScript example:

```
mbCommand("protocol", "traceTriggerDataOffset 0");
```

traceTriggerData
parameter: value 8 bit binary value 1|0|X

Set the data value used in the trace trigger

MBScript example:

```
mbCommand("protocol", "traceTriggerData XXX01XX");
```

traceTriggerStopOnNACK
parameter: {true|false}

Set trace trigger when NACK is received

MBScript example:

```
mbCommand("protocol", "traceTriggerStopOnNACK true");
```

traceTriggerType
parameter: {none|read|write|event}

Display restart transactions

MBScript example:

```
mbCommand("protocol", "showRestart true");
```

Picture Window commands

The pictureWindow acccepts image formats

pictureWindow Commands

loadFileFullPath *path*

loadFile *fileName path*

loadSequence *label image1 image2 timeDelay directoryPath*

size *x y*

title *label*

startTimer *time*

timeoutImage *replaceImage dirPath*

Video Window commands

The videoWindow acccepts video formats

videoWindow Commands

loadFileFullPath *path*

loadFile *fileName path*

title *label*

play

stop

pause

setPosition *time*

getPosition

isStopped

isPaused

isPlaying

Web Window commands

webWindow Commands

loadFileFullPath *url* //note: work around below to avoid //

loadFile *fileName path*

size *x y*

title *label*

Section 4 Complete Examples

The following example scripts are complete programs that may be copied to a text file and executed. They are designed to illustrate the use of the MBScript scripting language in practical terms.

Display Live Logic DVM Values

This script opens the Live Logic tool, reads the current input voltages, and displays their values. Before exiting a dialog is presented that pauses execution so that the values can be viewed.

```
// Open the Live Logic tool and stop acquisition
mbCommand("system", "openWindow liveLogic");
mbCommand("liveLogic", "stop");

// Open the message window
openMessageWindow();

do
{
	// Read the DVM values
	result = mbCommand("liveLogic", "getDVM");
	sleep(500);

	//  Break the csv response into a list of string
	split(result, "," , cmp[]);


	// Display the measured values
	writeMessageLine("The CH1 voltage is: " + toStr(cmp[0]), 0);
	writeMessageLine("The CH2 voltage is: " + toStr(cmp[1]), 1);
}
while(!abortFlag);

closeMessageWindow();
```

Test Live Logic DVM values against limits

This script shows how to capture a Live Logic DVM reading and determine if it is within defined limits. The Waveform source is used to provide an input voltage for this test. To reproduce this test connect the Live Logic CH1 probe to the Waveform source output.

```
// Set up the Waveform Source to produce 3.0V DC
mbCommand("system", "openWindow waveformSource");
mbCommand("waveformSource", "selectWaveformType supply");
mbCommand("waveformSource", "supplyVolts 3.0");
mbCommand("waveformSource", "run");

// Open the Live Logic tool and stop acquisition
mbCommand("system", "openWindow liveLogic");
mbCommand("system", "positionWindow liveLogic 550 60");
mbCommand("liveLogic", "stop");

// Open the message window
openMessageWindow();
writeMessageLine("Connect CH1 probe to Waveform Source", 0);
writeMessageLine("Test Limits: 2.8V < V < 3.2V");

do
{
	// Read the DVM values
	result = mbCommand("liveLogic", "getDVM");
	sleep(500);

	//  Break the csv response into a list of string
	split(result, "," , components[]);

	writeMessageLine("The CH1 voltage is: " + components[1], 2);

	ch1Voltage = components[1];
	if( (ch1Voltage > 3.2) || (ch1Voltage < 2.8) )
		writeMessageLine("Out of range", 3);
	else
		writeMessageLine("In range", 3);
}
while(!abortFlag);

closeMessageWindow();
```

Use time functions to periodically test and log results

This example uses the Live Logic DVM to take a measurement at 10 second intervals for 3 minutes. Results are logged to a local file named testResults.txt. The Waveform Source is used to provide a ramp voltage to make the readings more interesting. To execute this example connect the Live Logic CH1 probe to the Waveform Source.

```
// Open, configure, and start Waveform Source
mbCommand("system", "openWindow waveformSource");
mbCommand("waveformSource", "stop");
mbCommand("waveformSource", "selectWaveformType triangle");
mbCommand("waveformSource", "trianglePeriod 200");
mbCommand("waveformSource", "triangleDutyCycle 100");
mbCommand("waveformSource", "generateWaveform");
mbCommand("waveformSource", "run");
mbCommand("system", "positionWindow waveformSource 600 400");

// Open the Live Logic tool and stop acquisition
mbCommand("system", "openWindow liveLogic");
mbCommand("liveLogic", "stop");

// Open the message window
openMessageWindow();

fileName = "C:/Users/yourUser/Documents/testResults.txt";
file = openWriteFile(fileName);

// Add a header to the file
writeFileLine(file,"Periodic log of CH1 DVM reading");
writeFileLine(file,"Reading frequency is 10 seconds");
writeFileLine(file,"Readings are taken for 3 minutes");

writeMessageLine("Logging voltage values", 0);

// Values to track the time
startTime = osTick; // now
endTime = startTime + (3*60*1000); // 3 minutes from now
count = 1;
```

```
do
{
	// Read the DVM values
	result = mbCommand("liveLogic", "getDVM");

	//  Break the csv response into a list of string
	split(result, "," , components[]);


	//  Update the messageWindow
	writeMessageLine(count, 1);
	count++;

	str = strFormat("date : ", components[1]);
	writeMessageLine(str, 2);

	// Add the value to the file
	writeFileLine(file, str);

	// Wait 10 seconds and watch for user "abort"
	local timer = osTick + 10000;
	repeat
	{
	}
	until( (osTick >= timer) || abortFlag);
}
while(!abortFlag && (osTick < endTime));

closeWriteFile();

// Wait for the user
textWindow("Press OK to exit");

closeMessageWindow();
```

Calculate the frequency of a captured waveform

In this example the frequency of a previously captured Live Logic waveform is evaluated to determine its frequency. It requires that more than 100 complete cycles of the wave were captured and assumes that the waveform is periodic. To execute this script connect Live Logic channel 1 to Logic source channel 0 and then run the script.

```
// Some useful functions
function system(cmd)
{
	return mbCommand("system", cmd);
}

function logicSource(cmd)
{
	return mbCommand("logicSource", cmd);
}

function liveLogic(cmd)
{
	return mbCommand("liveLogic", cmd);
}

// Open, configure, and start Waveform Source
system("openWindow logicSource");

logicSource("stop");
logicSource("setValue 0 0 1e-6");
logicSource("setMode 0 next");
logicSource("setValue 1 1 1e-6");
logicSource("setMode 1 jumpA");
logicSource("setJump jumpA always");
logicSource("updateWaveform");
logicSource("run");

system("positionWindow logicSource 600 400");

// Open the Live Logic tool and stop acquisition
system("openWindow liveLogic");

liveLogic("stop");
liveLogic("mode single");
liveLogic("samples 256");
```

```
// Open the message window
openMessageWindow();

writeMessageLine("Example Measure Frequency", 0);

// Get an acquisition with timeout if no trigger
tries = 0;
liveLogic("run");
do
{
      sleep(250);
      runState = liveLogic("getRunState");
      writeMessageLine("Live Logic is "+runState, 1);
}
while( (runState != "stopped") && (tries < 10) );

// Check that there was a trigger
if (tries > 10)
{
      textWindow("Acuqisition failed");
      liveLogic("stop");
      exit;
}

// Get the time between edges csv
edgeTimes = liveLogic("getWaveformTime");

//  Break the csv response into a list of string
n = split(edgeTimes, "," , edgeTimeList[]);

if (n < 220)
{
      textWindow("Instufficient number of edges");
      exit;
end

// Average over 100 cycles (200 edges)
edgeTimeFirst = edgeTimeList[5];
edgeTimeLast = edgeTimeList[205];

// Calculate the frequency from 100 periods and display result
frequency = (1/(edgeTimeLast - edgeTimeFirst))*100;
writeMessageLine(strFormat("Frequency is {0:0.00}",frequency), 2);

logicSource("stop");
liveLogic("stop");

// Wait for the user
```

```
textWindow("Press OK to exit");

closeMessageWindow();
```