Logic Automation Users Guide

Recently we added a TCP socket server to the software which will allow you control the software using simple text commands.

The socket connection is by default on port 10429, but can be modified in the software’s preferences dialog.

Please download the beta version of the software, as well as the .NET automation tester.

To enable the scripting interface in the software, open the preferences dialog from the options menu, select the developer tab, and then check the box for “Enable scripting socket server.” Save these changes.

The scripting interface is cross platform, and will work on Windows, Linux and Mac.

The scripting interface supports a few commands. Some commands support require arguments. Arguments must be separated with the comma character: ‘,’

Upon execution, each command will return back with either an “ACK” or a “NAK” string. If any error occurs, or if the command is unrecognized, the software will return “NAK.”

We have also provided an automation tester program written in C# that will only run on windows. The program includes functions which will access and pass commands to the software’s socket for you.

**Demo Mode**

The application is, by default set to demo\_mode. The software will run through a sequence of demonstration commands prompted by the enter key. The Demo() function within SocketAPI implements some of the C# convenience functions described under "Using the C# console APP."

You can turn on and off demo mode from a bool flag at the top of the code, under GlobalVariable:

public static bool demo\_mode = false;

**Using the Socket Commands from the console prompt**

While the application is running, Socket commands can also be typed directly into the command prompt. For convenience, we recommend using the C# Functions as described in "Using the C# console app," when creating automation scripts.

From the prompt, press h for a semi-complete list of available commands and examples.

**Using the C# app**

As a convenience, the SocketAPI class, in the automation program, directly implements C# functions for controlling the software.

An instance of the SocketAPI class must be constructed in order to establish the socket connection. The constructor takes both the IP address and port number of the requested socket. The default values are set for localhost and the default software socket port.

SocketAPI( String host\_str = "127.0.0.1", int port\_input = 10429 )

{

this.port = port\_input;

this.host = host\_str;

Socket = new TcpClient(host, port);

Stream = Socket.GetStream();

}

The functions may then be called from the created object:

SAPI = new SocketAPI(host, port);

SAPI.FunctionCall();

**List of Software Controls Commands**

**Software PreCapture 4**

Set Trigger 4

Set Number of Sample in Collection 5

Set Collection Sample Rate 5

Get Capture Pretrigger Buffer Size 6

Set Capture Pretrigger Buffer Size 6

**Device/Channel Selection 7**

Get Connected Devices 7

Select Active Device 8

Get Active Logic16 Channels 8

Set Active Logic16 Channels 9

Reset Active Logic16 Channels 9

**Capture Data 10**

Capture 10

Capture and Save to File (Under Save/Load) 10

Get Inputs 11

**Save/Load 12**

Save to File 12

Load from File 12

**Analysis and Export 13**

Export Data 13

Get Analyzers 16

Export Analyzers 17

**Software PreCapture**

**Set Trigger**

**Socket Command: set\_trigger**

This command lets you configure the trigger. The command must be sent with the same number of parameters as there are channels in the software. For use with Logic, 8 parameters must be present. Blank parameters are allowed.

Parameter value options:

* high
* low
* negedge
* posedge

Example:

set\_trigger, high, low, posedge,,,,,high

**C# Function:** void SetTrigger(Trigger[] triggers)

The function takes in an array of enum Trigger. The index of the array corresponds to the channel.

Example:

Trigger[] trigger = { Trigger.High, Trigger.Posedge, Trigger.None, Trigger.Low, Trigger.High, Trigger.High, Trigger.None, Trigger.None }

SetTrigger( trigger )

**Set Number of Samples in Collection**

**Socket Command: set\_num\_samples**

This command changes the number of samples to capture. You must specify a value that the software recognizes. (Note: the only allowed values are those in the drop-down menu in the software )

Example:

set\_num\_samples, 1000000

**C# Function:** void SetNumSamples(int num\_samples)

The function takes an integer to set the desired number of samples.

**Set Sample Rate**

**Socket Command: set\_sample\_rate**This command changes the sample rate in the software. You must specify a sample rate which is listed in the software. There is currently no helper function to get a list of sample rates. (Note: the only allowed values are those in the drop-down menu in the software )

Example:

set\_sample\_rate, 1000000

**C# function:** void SetSampleRate(int sample\_rate)

The function takes an integer to set the desired sample rate.

**Get Capture PreTrigger Buffer Size**

**Socket Command: get\_capture\_pretrigger\_buffer\_size**

This command gets the pretrigger buffer size of the capture.

Example:

get\_capture\_pretrigger\_buffer\_size

**C# Function:** int GetCapturePretriggerBufferSize()

The function returns an integer with the current pretrigger buffer size.

**Set Capture PreTrigger Buffer Size**

**Socket Command: set\_capture\_pretrigger\_buffer\_size**

This command sets the pretrigger buffer size of the capture

Example:

set\_capture\_pretrigger\_buffer\_size, 1000000

**C# Function:** void SetCapturePretriggerBufferSize(int buffer\_size)

The function takes an integer with the desired buffer size

**Device/Channel Selection**

**Get Connected Devices**

**Socket Command: get\_connected\_devices**

This command will return a list of the devices currently connected to the computer. The connected device will have the return parameter ACTIVE at the end of the line.

Example:

get\_connected\_devices

Response

1, Demo Logic, LOGIC\_DEVICE, 0x19b2

2, My Logic 16, LOGIC16\_DEVICE, 0x2b13, ACTIVE

ACK

**C# Function:** ConnectedDevices[] GetConnectedDevices()

The function returns an array of ConnectedDevices structs. The structs contains the type of device, the name, the device id, the index of the device and whether or not the device is currently active.

struct ConnectedDevices

{

String type;

String name;

int device\_id;

int index;

bool is\_active;

}

**Select Active Device**

**Socket Command: select\_active\_device**

This command will select the device set for active capture. It takes one additional parameter: the index of the desired device as returned by the get\_connected\_devices function.

Example:

select\_active\_device, 2

**C# Function:** void SelectActiveDevice(int device\_number)

The function takes in the index of the desired device as returned by the GetConnectedDevices() function.

**Get Active Logic16 Channels**

**Socket Command: get\_active\_logic16\_channels**

This command will return a list of the active Logic16 channels. (Note: function for use with Logic16 only)

Example:

get\_active\_logic16\_channels

Response:

2, 11, 14, 16

**C# Function:** int[] GetActiveLogic16Channels()

The function returns an array of integers containing the active Logic16 channels.

**Set Active Logic16 Channels**

**Socket Command: set\_active\_logic16\_channels**

This command allows you to set the active logic 16 channels. (Note: function for use with Logic16 only)

Example:

set\_active\_logic16\_channels, 0, 4, 11, 15

**C# Function:** void SetActiveLogic16Channels(int[] channels)

The function takes in an array of integers of the desired channel numbers.

Example:

SetActiveLogic16Channels({ 2, 5, 7, 12 })

**Reset Active Logic16 Channels**

**Socket Command: reset\_active\_logic16\_channels**

This command will reset the active logic16 channels, activating all non-active logic16 channels. (Note: function for use with Logic16 only)

Example:

reset\_active\_logic16\_channels

**C# Function:** void ResetActiveLogic16Channels()

The function does not require any additional parameters.

**Capture Data**

**Capture**

**Socket Command: capture**

This command starts a capture. It will return NAK if an error occurs.

Example:

capture

**C# function:** void Capture()

The function takes no parameters.

Example:

Capture()

**Capture to File**

**Socket Command: capture\_to\_file**

This command starts a capture, and then auto-saves the results to the specified file. If an error occurs, the command will return NAK.

Note: You must have permissions to the destination path, or the save operation will fail. By default, applications won’t be able to save to the root of the C drive on windows. To do this, the Logic software must be launched with administrator privileges.

Example:

capture\_to\_file, c:\temp.logicdata

**C# function:** void CaptureToFile(String file)

The function takes a string with the file name to save to.

Example:

CaptureToFile("C:/temp\_file")

**Get Inputs**

**Socket Command: get\_inputs**

This command is a convenience function, and it reads the current value of each of the inputs from the connected device. (Note: Function for use with Logic device only)

Example:

get\_inputs

Response:  
01101101

ACK

**C# Function:** int[] GetInputs()

The function returns an array of integers with a boolean value representing each input. The index of the array corresponds to the channel.

**Save/Load**

**Save to File**

**Socket Command: save\_to\_file**

This command saves the results of the current tab to a specified file. (Write permission required, see capture to file)

Example

save\_to\_file, C:\temp.logicdata

**C# Function:** void SaveToFile(String file)

The function takes a string with the file name to save to.

Example:

SaveToFile("C:/temp\_file")

**Load From File**

**Socket Command: load\_from\_file**

This command loads the results of previous capture from a specific file.

Example

load\_from\_file, C:\temp.logicdata

**C# Function:** void LoadFromFile(String file)

The function takes a string with the file name to load.

Example:

LoadFromFile("C:\temp\_file")

**Analysis and Export**

**Export Data**

**Socket Command: export\_data**

This function exports the data from the current capture to a file. There are several options which are needed to specify how the data will be exported:

(Note: the options are order-specific. The software will send a NAK if the options are out of order.)

File Name: Specify the file to export to

Ex: export\_data, C:\temp\_file

Channels: Specify the channels to export

* all\_channels - export all active capture channels
* specific\_channels - export only the specified channels.
* (Ex: ..., specific channels, 2, 4, ... )

Time: Specify the range of time to export

* all\_time: export the entire time range of the capture
* time\_span: export time range between two values
* (Ex: ..., time\_span, 0, 0.5, ... )
* timing\_markers: export the time range between the two markers (note: timing markers MUST be active to use this option)

You will need to specify the data format that the exported data will be represented in. The required options vary depending on the format:

CSV: include command csv

* headers/no\_headers: include column headers
* tab/comma: tab or comma delimiter
* sample\_number/time\_stamp: display current time of sample number for samples
* combined/separate: output a number, or a column for every bit
* row\_per\_change/row\_per\_sample: output a row for every transition, or a row for every sample
* **If Combined**(above option) Dec/Hex/Bin/Ascii: display in desired format

Ex: ..., csv , headers, tab, sample\_number, combined, row\_per\_change, hex

Binary: include command bin

* each\_sample/on\_change: output data for each sample or output data on transition
* 8/16/32/64: number of bytes in output sample

EX: ..., binary, each\_sample, 32

VCD: include command vcd

No additional parameters required

Examples:

export\_data, C:\temp\_file, all\_channels, timing\_markers, csv, headers, comma, time\_stamp, separate, row\_per\_change

export\_data, C:\temp\_file, specific\_channels, 0, 1, 2, 3, time\_span, 0.2, 0.4, vcd

**C# Function:** void ExportData(ExportDataStruct export\_data\_struct)

The function takes a ExportDataStruct struct as its argument. The struct holds all the information that is needed to export the data. For each option, there is an enumeration to select which option you would like to choose.

public struct ExportDataStruct

{

//File Name

String FileName;

//Channels

bool ExportAllChannels;

int[] ChannelsToExport;

//Time Range

//{ RangeAll, RangeMarkers, RangeTimes }

DataExportSampleRangeType SamplesRangeType;

double StartingTime;

double EndingTime;

//Export Type

//{ ExportBinary, ExportCsv, ExportVcd }

DataExportType DataExportType;

//CSV

//{ CsvIncludesHeaders, CsvNoHeaders }

CsvHeadersType CsvIncludeHeaders;

//{ CsvComma, CsvTab }

CsvDelimiterType CsvDelimiterType;

//{ CsvSingleNumber, CsvOneColumnPerBit }

CsvOutputMode CsvOutputMode

//{ CsvTime, CsvSample }

CsvTimestampType CsvTimestampType;

//{ CsvBinary, CsvDecimal, CsvHexadecimal, CsvAscii }

CsvBase CsvDisplayBase;

//{ CsvTransition, CsvComplete }

CsvDensity CsvDensity;

//BINARY

//{ BinaryEverySample, BinaryEveryChange }

BinaryOutputMode BinaryOutputMode;

//{ BinaryOriginalBitPositions, BinaryShiftRight }

BinaryBitShifting BinaryBitShifting;

//{ Binary8Bit, Binary16Bit, Binary32Bit, Binary64Bit }

BinaryOutputWordSize BinaryOutputWordSize;

}

Example 1:

ExportDataStruct ex\_data\_struct = new ExportDataStruct();

ex\_data\_struct.FileName = @"C:\Users\Name\Desktop\test1";

ex\_data\_struct.SamplesRangeType = DataExportSampleRangeType.RangeAll;

ex\_data\_struct.ExportAllChannels = true;

ex\_data\_struct.DataExportType = DataExportType.ExportVcd;

ExportData(ex\_data\_struct);

Example 2:

ExportDataStruct ex\_data\_struct = new ExportDataStruct();

ex\_data\_struct.FileName = @"C:\User\Name\Desktop\test2";

ex\_data\_struct.SamplesRangeType = DataExportSampleRangeType.RangeTimes;

ex\_data\_struct.StartingTime = 0;

ex\_data\_struct.EndingTime = 0.000145;

ex\_data\_struct.ExportAllChannels = false;

ex\_data\_struct.ChannelsToExport = new int[] { 1, 4, 7 };

ex\_data\_struct.DataExportType = DataExportType.ExportCsv;

ex\_data\_struct.CsvDelimiterType = CsvDelimiterType.CsvTab;

ex\_data\_struct.CsvDensity = CsvDensity.CsvComplete;

ex\_data\_struct.CsvDisplayBase = CsvBase.CsvDecimal;

ex\_data\_struct.CsvIncludeHeaders = CsvHeadersType.CsvNoHeaders;

ex\_data\_struct.CsvOutputMode = CsvOutputMode.CsvOneColumnPerBit;

ex\_data\_struct.CsvTimestampType = CsvTimestampType.CsvSample;

ExportData(ex\_data\_struct);

**Get Analyzers**

**Socket Command: get\_analyzers**

This function will return a list of analyzers currently attached to the capture, along with indexes so you can access them later.

Example:

get\_analyzers

Return Value:

SPI, 0

I2C, 1

SPI, 2

Please note that each line is separated by the ‘\n’ character.

**C# Function:** Analyzer[] GetAnalyzers()

The function returns an array of Strings, each containing the name and index of the analyzer.

struct Analyzer

{

String type;

int index;

}

Example:

Analyzer[] Analyzers = GetAnalyzers()

**Export Analyzers**

**Socket Command: export\_analyzers**

This command is used to export the analyzer results to a specified file. Pass in the index from the get\_analyzers function, along with the path to save to.

Add a third, optional parameter to have the results piped back through the TCP socket to you.

Example:

export\_analyzer, 0, c:\spi\_results.csv

Or:

export\_analyzer, 0, c:\spi\_temp.csv, extra\_parameter

Results:

Time [s],Packet ID,MOSI,MISO

5.20833333333333e-006,0,'0','1'

9.375e-006,0,'1','2'

1.35416666666667e-005,0,'2','3'

2.6875e-005,1,'4','5'

3.10416666666667e-005,1,'5','6'

…

ACK

Please note that streaming the results back may add a little delay. This will be fixed in future versions.

**C# Function:** void ExportAnalyzers(int selected, String filename, bool mXmitFile)

The function takes 3 parameters. The first is the index of the analyzer returned from the GetAnalyzers() function. The second is the filename to save to, and the third determines whether or not to pipe the information back through the TCP socket to you.