Logic Socket API Users Guide

The Saleae Logic software includes a basic TCP socket serer, which can be used to script basic software actions for automation.

This lets custom applications connect to the software's socket server, and then send basic text commands to control different parts of the software, such as device configuration, capture, and export of raw data or protocol data.

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

Please download the latest beta version of the software, as well as the provided Saleae Socket API Github project.

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 require arguments. Arguments must be separated with the comma character: ',' . **Note: All commands sent over the socket API must be NULL terminated.** Note – by default, socket libraries will not append a null character to the transmitted data, so one must be added manually, usually with the '\0' escape sequence.

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." Some commands will return information, followed by the ACK or NAK string.

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.

Saleae API Sample Project Layout

The Saleae Socket API visual studio solution contains two important projects. (Other projects may be present, but are not covered by this document)

SaleaeSocketApi is a class library that provides a basic C# API over the socket functions described in this document. It produces a DLL which can be used by other C# projects.

SaleaeSocketApiExample is a basic console application that uses the SaleaeSocketApi class library. It demonstrates a handful of basic socket commands.

Demo Mode

The SaleaeSocketApiExample program will run through a sequence of demonstration commands prompted by the enter key.

Using the C# app

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

An instance of the `SaleaeClient` 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.

```
SaleaeClient( 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:

```
Client = new SaleaeClient(host, port);
Client.FunctionCall();
```

Contents and Socket Command Reference

Software Capture

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. (Note: USB transfer chunks are about 33ms of data so the number of samples you actually get are in steps of 33ms)

Example:

set_num_samples, 1000000

C# Function: `void SetNumSamples(int num_samples)`

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

Socket Command: set_capture_seconds

This command changes the number of seconds to capture for.

Example:

set_capture_seconds, 0.8

C# Function: `void SetCaptureSeconds(double seconds)`

The function takes a double to set the desired number of seconds to sample for.

Set Collection 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: To get the available sample rates use get_all_sample_rates).

Syntax: set_sample_rate, $(digital_sample_rate), $(analog_sample_rate)

Example:

Digital Only Capture: set_sample_rate, 1000000, 0

Analog Only Capture: set_sample_rate, 0, 100000

Both Analog and Digital Capture: set_sample_rate, 2000000, 1000000

C# function: `void SetSampleRate(SampleRate sample_rate)`

The function takes a struct that holds both the digital sample rate and the analog sample rate.

Get Available Sample Rates

Socket Command: get_all_sample_rates

This command returns all the available sample rate combinations for the current performance level and channel combination.

Example: get_all_sample_rates

Response(${digital sample rate), ${analog sample rate}):

5000000, 1250000

10000000, 625000

C# function: `List<SampleRate> GetAvailableSampleRates()`

This function returns a list of all the sample rates available for the current performance option and channel combination.

```
struct SampleRate
{
    public int AnalogSampleRate;
    public int DigitalSampleRate;
}
```

Get Performance Option

Socket Command: get_performance

This command returns the currently selected performance options.

Example: get_performance

Reponse:

100

C# function: `PerformanceOption GetPerformanceOption()`

`This function returns the currently selected performance option in the form of an enum.`

```
enum PerformanceOption { Full = 100, Half = 50, Third = 33, Quarter = 25, Low = 20 };
```

Set Performance Option

Socket Command: set_performance

This command returns the currently selected performance options. Valid performance options are: 20, 25, 33, 50, and 100. Note: This call will change the sample rate currently selected.

Syntax: set_performance, ${value}

Example: set_performance, 50

C# function: `void SetPerformanceOption( PerformanceOption performance )`

`This function sets the currently selected performance option to the value of “performance”.`

Get Capture Pre-Trigger 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 Pre-Trigger Buffer Size

Socket Command: set_capture_pretrigger_buffer_size

This command sets the pretrigger buffer size of the capture. Note: Currently, the pretrigger buffer size has to be one of the following values: `1000000, 10000000, 100000000, or 1000000000.`

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. Note: Indices start at 1, not 0.

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.

Changing Active Channels

Socket Command: get_active_channels

This command will return a list of the active channels.

Example:

get_activ_channels

Response:

digital_channels, 0, 4, 5, 7, analog_channels, 0, 1, 2, 5, 8

C# Function: `void GetActiveChannels(List<int> digital_channels, List<int> analog_channels)`

This function populates the lists with the currently active channels. The integer in each list represents the channel number that is active.

Set Active Channels

Socket Command: set_active_channels

This command allows you to set the active channels. Note: This feature is only supported on Logic 16, Logic 8(2nd gen), Logic Pro 8, and Logic Pro 16

Example:

set_active_channels, digital_channels, 0, 4, 5, 7, analog_channels, 0, 1, 2, 5, 8

C# Function: `void SetActiveChannels(int[] digital_channels = null, int[] analog_channels = null)`

This functions takes an array of integers for both digital channels and analog channels. Each integer represents the channel number to set active. Both parameters support a null array. A null array represents disabling all the channels for that type.

Example:

```
SetActiveChannels(new int[] { 0, 1, 2, 3, 4 }, new int[] { 0, 1 });
```

Reset Active Channels

Socket Command: reset_active_channels

This command will set all channels active for the device.

Example:

reset_active_channels

C# Function: `void ResetActiveChannels()`

This function takes no parameters and returns none.

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()
```

Stop Capture

Socket Command: capture

This command stops the current capture. It will ACK if data is present after the capture is stopped or it will NAK if no data is present.

Example:

stop_capture

C# function: `void StopCapture()`

The function takes no parameters.

Note: The C# capture command blocks until the capture is complete.

Example:

```
StopCapture()
```

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.

The path passed in must be absolute and the destination directory must exist, or the software will NAK.

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 has been disabled temporarily.

Is Processing Complete

Socket Command: is_processing_complete

With the introduction of analog channels, processing data is no longer instant and may take some time after the capture. You cannot export or save data until processing is complete (The commands will NAK). This command returns a Boolean expressing whether or not the software is done processing data.

C# Function: `bool IsProcessingComplete()`

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)

(Note: Data processing much be complete before this command is ran or it may NAK. See “Is Processing Complete” to check if processing is done before exporting data)

The path passed in must be absolute and the destination directory must exist, or the software will NAK.

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.

The path passed in must be absolute and the destination directory must exist, or the software will NAK.

Example

load_from_file, C:\temp.logicdata

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

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

Example:

```
LoadFromFile("C:\temp_file")
```

Close All Tabs

Socket Command: close_all_tabs

This command closes all currently open tabs.(Note: This command does not delete the data in the capture tab)

Example

close_all_tabs

C# Function: `void CloseAllTabs()`

The function writes the close_all_tabs command to the socket API.

Example:

```
CloseAllTabs()
```

Analysis and Export

Export Data 2

Socket command: export_data2

It is highly recommended to examine the examples of this in the SaleaeSocketApiUnitTests project. The raw data export process can be extremely complex, due to the large number of parameters and possible export configurations.

The Saleae software supports four export file formats: CSV, Binary file, Matlab, and VCD (Value Change Dump)

Each export option has its own specific parameters.

The CSV and Matlab formats are the only formats capable of exporting a mix of analog and digital channels.

The Binary format supports exporting analog or digital, but not both at the same time.

The VCD format is only capable of exporting digital data.

The export formats that support analog and digital data have some parameters that only apply to one data type or the other. For instance, displaying analog values as ADC values (0-4095 for most devices) or as calibrated voltages only apply when exporting analog data.

To complicate things more, the CSV export mode has three sets of export options. Options when exporting only digital data, options when exporting only analog data, and then options when exporting both.

Finally, just to add icing to the cake, the types of channels that are active in the original capture, regardless of which ones you would like to export, also has an effect.

For example, if you record a few digital channels, and then export them as CSV, you will need to use a slightly different command than if you recorded a few digital channels and a few analog channels, but still only want to export the digital channels to CSV.

All of this combined makes for a particularly complex command structure.

The export options are much easier to understand when using the GUI built into the software. The socket command merely mimics this dialog. The export options that are possible through the API are exactly the same as the export options on the software, and they are mapped one to one. In fact, the parameters for the export data command are actually the direct members of the backing store for the software's GUI.

There is good news. If any parameter is included out of order, or missing, or an extra parameter is added, the software's console output will tell you exactly which parameter was not understood, and it will tell you exactly which options were acceptable at that point.

For details on when each parameter is required, see comments in SocketApiTypes.cs for the structure `ExportDataStruct`.

For examples of each use case, see the examples in ExportUnitTests.cs in the class `ExportUnitTests`.

Note: The save path passed in must be absolute and the destination directory must exist, or the software will NAK.

Export Data [deprecated]

Socket Command: export_data

This function is still present in the Saleae software, but has never properly supported Mixed mode exports (when analog and digital channels are present in the export) and might not support digital-only or analog-only exports properly when the original capture included digital and analog channels.

If you are already using this function successfully, it should be safe to keep using. For new applications, please use Export Data 2. Export Data 2 is nearly identical for standard exports (digital-only or analog-only) but explicitly supports all possible configurations of export in mixed captures.

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.)

(Note: Data processing much be complete before this command is ran or it may NAK. See "Is Processing Complete" to check if processing is done before exporting data)

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
- digital_channels/analog_channels - export only the specified channels.
  - (Ex: …, digital_channels, 0, 2, 3, analog_channels, 1, 2, …)
  - Note: If you are only exporting digital channels you may leave off "analog_channels" and vice versa. However, if both are included, digital must be first.

Analog Format: Specify how to output analog values. This setting has to be included if you are exporting analog data. If you are exporting only digital data, do not include this setting.

- Voltage – Outputs the voltage value.
- ADC – Outputs the ADC value.
  - (Ex: …, voltage, …)

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

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
    - **If Separate:** Add row_per_change/row_per_sample afterwards: output a row for every transition, or a row for every sample
- Dec/Hex/Bin/Ascii: display in desired format

    Ex: ..., csv , headers, tab, sample_number, separate, 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 bits in output sample

    EX: ..., binary, each_sample, 32

VCD: include command vcd

No additional parameters required

Matlab: include command matlab

No additional parameters required

Examples:

export_data, C:\temp_file, digital_channels, 0, 1, analog_channels, 1, voltage, all_time, adc, csv, headers, comma, time_stamp, separate, row_per_change, Dec

export_data, C:\temp_file, all_channels, time_span, 0.2, 0.4, vcd

export_data, C:\temp_file, analog_channels, 0, 1, 2, adc, all_time, matlab

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
   public bool ExportAllChannels;
   public int[] DigitalChannelsToExport;
   public int[] AnalogChannelsToExport;
   //Time Range

   //{ RangeAll, RangeMarkers, RangeTimes }
   DataExportSampleRangeType SamplesRangeType;
   double StartingTime;
   double EndingTime;

   //Export Type
   //{ ExportBinary, ExportCsv, ExportVcd, ExportMatlab }
   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;  .
public AnalogOutputFormat AnalogFormat; //This feature needs v1.1.32+
    }
```

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.DigitalChannelsToExport = 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.

The path passed in must be absolute and the destination directory must exist, or the software will NAK.

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.

Socket Command: is_analyzer_complete

This command is used to get whether or not a specific analyzer has finished processing the data and generating results. This command should be used before trying to export an analyzer to verify it is done processing.

Syntax: is_analyzer_complete, $(analyzer_index)

Note: To get analyzer index, use get_analyzers.

Example: is_analyzer_complete, 1

C# Function: `bool IsAnalyzerProcessingComplete(int` analyzer_index)

This function takes the analyzer index and returns true if it is done processing data and safe to export.