TELEDYNE LECROY
Everywhereyoulook™

frontline® Automation Server Protocol

Programmers Guide

Revision Date: 1/17/2017

Contents

Chapter 1 Overview

1.1 Purpose

This document describes a protocol that provides remote control of Frontlline's Protocol Analysis System (Frontline software). Utilizing the protocol, a remote client can bypass the Microsoft Windows user interface and interact directly with the Frontline software..

1.2 Audience

This document is intended for technical people with knowledge of TCP socket communications who want to automate the process of testing Classic Bluetooth® devices, *Bluetooth* low energy devices, 802.11 and SD/SDIO devices using the Frontline software. The test process automation is achieved by writing a client application which talks over a TCP network socket connection with the Frontline Automation Server component.

1.3 Hardware and Software Requirements

- The Automation Server requires one or more of the following Frontline products.
  - Sodera
  - BPA 600
  - BPA low energy Analyzer
  - 802.11
  - SD/SDIO
- The Automation Server requires the use of the following software that is delivered with each Frontline hardware device.
  - Frontline's ComProbe Protocol Analysis System (Frontline software)
- TCP/IP network

1.4 Typical System Topology

The typical system topology includes:

- One server machine running one or more instances of the Frontline software application and one instance of the Automation Server, and
- One or more client machines running one or more client applications.

The client application is platform independent. The only requirement for the client application is that it is able to establish a TCP socket connection with the server machine in order to communicate using the valid Frontline Automation Server ASCII protocol described later in this document.

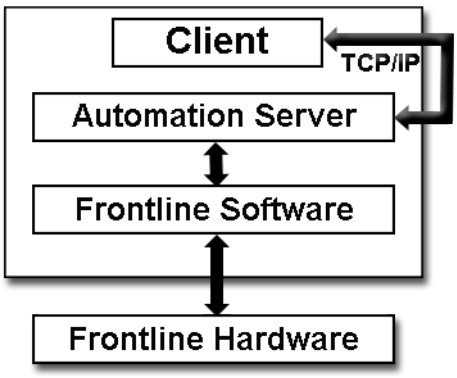

Figure 1.1 - Client/Server Hardware Configuration Using One Computer

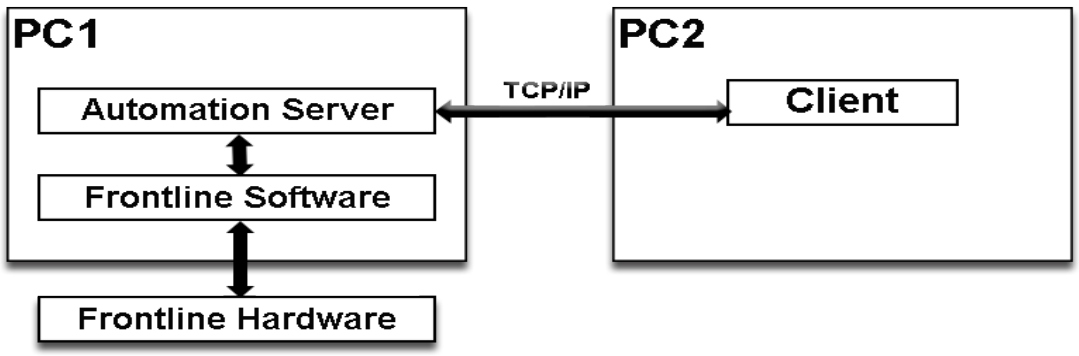

Figure 1.2 - Client/Server Hardware Configuration Using Two Computers

1.5 Client Implementations

Multiple clients can connect to the Automation Server. Once a connection is established with the server, the client communicates with the server using delimited ASCII strings. The delimiting token is the ';' character. The ASCII strings are formatted as follows:

<COMMAND>;<PARAM 1>;<PARAM 2>;<PARAM X>

An example of a complex command requiring a named, value pair parameter format is the HTML Export command.

HTML Export;Summary=0;Data Bytes=1;File=htmlexport1.htm

The <PARAM> is sometimes a named, value pair of the format:

<PARAM NAME>=<PARAM VALUE>

Some simpler commands only have one parameter that is either on or off. These commands do not require naming. An example of this is the SYNC STATUS command.

```
Sync Status;ON
```

An optional parameter exists that one can use in cases with multiple data sources, (e.g. Wi-Fi Coexistence), to identify the desired data source. This parameter applies to the commands Config Settings, Start Sniffing, and Stop Sniffing.

```
<COMMAND>;<DATASOURCE>;<PARAM 1>;<PARAM 2>;<PARAM 3>
Config Settings; Datasource=0;IOParameters;Channel=3
Config Settings; Datasource=1;IOParameters;EncryptionSelection=5
```

When the data source parameter is missing, the value is 0.

The valid commands accepted by the server are discussed in detail in Chapter 3 Commands on page 8.

1.6 Server Implementation

The Automation Server returns either a FAILED or SUCCEEDED notification response to the client for each command it processes. The response is in the form of:

```
<COMMAND>;<SUCCEEDED> | <FAILED>;<Timestamp>;<Reason>
```

Where the <Timestamp> token is of the form:

```
<Timestamp>=<Time & Date>
```

And the <Reason> token is of the form:

```
<Reason>=<Some string error message>
```

All notification responses are terminated with the characters '\r' and '\n'.

Chapter 2 Getting Started

In this chapter, we describe how to get started with running the Automation Server and a sample client script. You need to know the values of two parameters before starting the Automation Server. These include the IP Address and a valid TCP Port number on the PC running the Automation Server. On the Client side, you need to know the Personality Key to use while launching the program.

2.1 Personality Key

Frontline software runs based on a specific hardware personality. The following table is a list of the personality keys, and a description of required Frontline hardware and the data sourcing methods. The Personality Key is an important part of using the Automation Server to launch the Frontline application.

Table 2.1 - Personality Keys

Personality Key	Description
Sodera	This method requires one Frontline Sodera and may be used to capture Classic *Bluetooth* and *Bluetooth* low energy data.
Sodera_80211_COEX	This method requires one Frontline Sodera hardware and one Frontline 802.11 hardware, and may be used to capture Classic *Bluetooth*, *Bluetooth* low energy data and 802.11 data..
BPA600	This method requires one Frontline BPA 600 and is used to capture Classic *Bluetooth* and *Bluetooth* low energy data. Used for typical applications to capture Classic Bluetooth and Bluetooth low energy data. Modes include: **LE Only** - *Bluetooth* low energy only **Classic Only Single Connection** **Dual Mode** - Classic *Bluetooth* and *Bluetooth* low energy. **Classic Only Multiple Connections**

Table 2.1 - Personality Keys(continued)

Personality Key	Description
BPA600_Coex	This method requires one Frontline BPA 600 hardware and one Frontline 802.11 hardware. Captures Classic *Bluetooth* , low energy, and 802.11 data and displays in the Frame Display and Coexistence View.
FTSLE	This method requires one Frontline BPA low energy and is used to capture low energy data. Used for typical applications to capture *Bluetooth* low energy data.
80211	Requires one Frontline 802.11 hardware. Captures 802.11 data on the selected channel.
TwoWiFi	Requires two Frontline 802.11 hardware. Captures 802.11 data on the selected channel.
SDIO	Requires one Frontline SD hardware.

2.2 Launching ComProbe Automation Server

2.2.1 Modify IPAddr and Port

Before launching the Automation server, the IP address and Port number values must be modified in the configuration file. The configuration file is an XML file named “FTSAutoServer.exe.config" located in the “C:\Program Files (x86)\Frontline Test System II\Frontline ComProbe Protocol Analysis System *<your version>*\Executables\Core” directory.

Open this file in any text editor and look for two keys named “IPAddr” and “Port”. The default value for “IPAddr” is “0.0.0.0”. Edit this value to the IP address of the machine running the Automation Server. The default value of the Port number is set to 22901. This port number is not a commonly used TCP port numbers. It is unlikely that there is another application using it. You can leave the default number as it is. Save the changes to the file.

```
<add key="IPAddr" value="192.168.1.0"/>
<add key="Port" value="22901"/>
```

2.2.2 Launch Automation Server

The Automation Server requires security privileges to run. A user with the right account privileges must be logged in to the machine that will run the Automation Server. The Automation Server launches as a member of the startup group.

To launch the Automation Server, go to the desktop folder that was created by the software installer. Then go to the ‘Development tools’ folder and double-click the ‘Automation Server’ short-cut. The location of this short-cut is “C:\Users\Public\Desktop\Frontline (*version #*)\Development Tools”.

If the setup was successful, the “FTS Automation Server” window launches and the first message will say “Listening for TCP Client on Port 22901”. See Automation Server Successful Setup Response on the facing page.

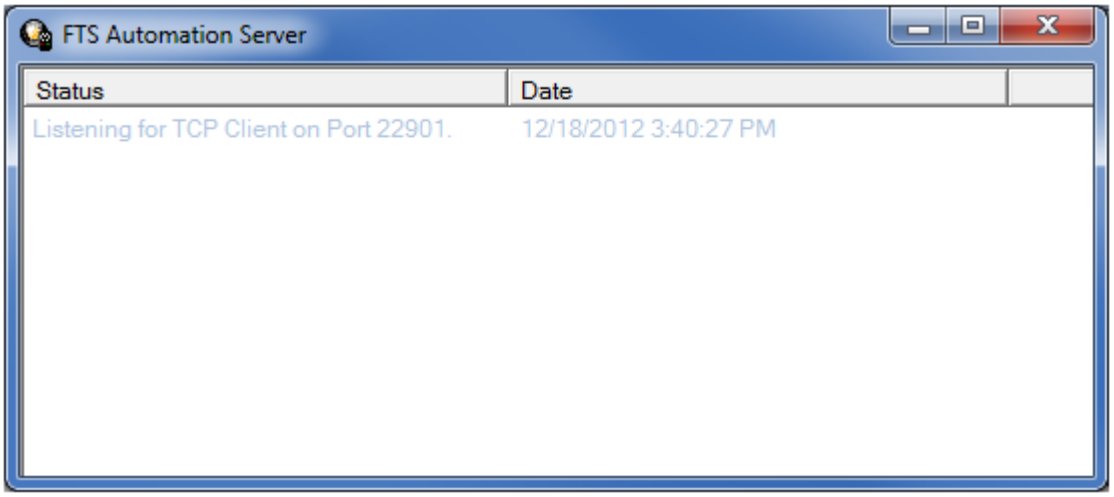

Figure 2.1 - Automation Server Successful Setup Response

2.2.3 Client Side

Appendix 1 of this document contains a sample script named “SampleClient.tcl”. You can modify that file for this portion of the guide. Open “SampleClient.tcl’ and modify the Connections(Host) and Connections(Port) to reflect the same changes made in the Automation configuration file, see Section [1]Modify IPAddr and Port on page 5. The Connections (Host) should be set to the IPAddr value and the Connections(Port) should be set to the same Port value.

The only other changes necessary to make are the Config Settings command parameters. These are specific to the devices being sniffed. Find the statement in the script that set the state: “Config Settings:IOParameters;BPA600”. The Slave and Master BD addresses should be modified to reflect those addresses of the devices under test. Along with that change the “EncryptionSelection” and possibly “PinCode” and “LinkKey” configuration pairs should be modified. All configuration pairs become key/value pairs followed by a “;”. For instance if the PinCode is 1234 then the parameter “PinCode=1234” would be added with a preceding “;”. Adding a link key parameter would look like this: “LinkKey=0x13456789abcdef1234567890abcdef12”.

Once these changes have been made to suit the needs of the devices being tested, then run the script by typing tclsh client.tcl at the prompt.

2.2.4 Running the Automation Script

Once the script is running, it will attempt to connect to the TCP socket that the automation server has setup. This is successful if you see the “Accepted TCP client on Port {?}” in the window of the automation server. Then the script will begin to execute the commands in the correct order. The first command must be "Start FTS". This command starts the instance of the Frontline software to run. The server will always launch the instance that is in the same directory as the Automation Server that is running or it can be configured to launch different instances. The last command to always run is "Stop FTS". The "Stop FTS" command always shuts down the instance of the Frontline software that is running. Some commands are specifically for a given certain type of hardware. For example, the Start/Stop Sniffing, Resync Status Now, and Sync Status are commands that are specific to *Bluetooth* sniffing hardware. If any of these commands were sent to a Frontline 802.11, the status return would be “Command not supported.”

[1]tcl files are based on Tool Command Language (TCL). TCL is an open-source, cross-platform programming language. More information is available at www.tcl.tk.

2.2.5 Log File

Error, warning, or information messages that occur as a result of starting and stopping the ComProbe software application - such as problems encountered when validating the software license - will be logged to the FTSAutoServer.log file normally located in the “C:\Users\Public\Documents\Frontline Test Equipment\My Log Files” directory. If an error is detected, the message will be logged and the ComProbe Automation Server will report a timeout error back to the client.

Chapter 3 Commands

The following are the current commands supported by the ComProbe Automation Server. Upon receipt of these commands, the ComProbe Automation Server packages these commands to send to the ComProbe Software application. Several important concepts to remember about these commands include:

- The commands are not case sensitive, unless specifically noted.
- The ComProbe Automation Server does not block waiting on a response to any command. The client is responsible for the block based on whether the response of a command is required before the next command can be sent.
- A failure notification is always sent if a command is not delivered to ComProbe software successfully.
- A success notification is always sent if a command is delivered to ComProbe software successfully. The word “delivered” is important. A success notification does not mean that the command was completed successfully, only that it was delivered successfully and that the action was started. The successful delivery of a command does not guarantee a successful execution of the command.
- Italics are used to indicate non-literals.
- A typical client could be implemented using the TCL (Tool Command Language).
    - The TCL interpreter would be required on the client machine.
    - The user would write a TCL script file and invoke it using the command

      tclsh clientscript.tcl

3.1 Config Settings

This command instructs the Frontline software application to initialize the IO settings for a *Bluetooth* Air device. Important ideas to remember about Config Settings include:

- The parameters will be parsed for correctness.
- If there is an invalid parameter name, then the parameter will be saved to the configuration INI file but will be ignored by the Frontline software application because it is unknown to the application.

- If the value is incorrect, then Frontline software will automatically default the parameter to the valid default value.
- The client can begin capturing or sniffing without setting the IO. Frontline software will use the previously configured values from the INI file or the system defaulted values.
- Valid IO settings are dependent upon the devices being tested.
- For any configuration that you send, you must have a data source key that matches the type of I/O settings in the application.
- Some devices do not support Extension channels or 5 GHz channels. The configuration will fail if the settings are not supported by the device.

Table 3.1 - Decoder Keys

Data Source Key	Data Source Type
Sodera	Classic or Low energy *Bluetooth*
Bluetooth	Classic *Bluetooth* and *Bluetooth* low energy datasource
Coexistence	Classic *Bluetooth* and 802.11 datasources
802.11	Wi-Fi datasources

3.1.1 Bluetooth

3.1.1.1 Classic *Bluetooth*

The following table lists the valid settings and corresponding values for Classic *Bluetooth* devices.

Table 3.2 - Classic *Bluetooth* Configuration Settings

Name	Values	Default	Description	ComProbe
ClearChannelMapOnResync	False=0, True=1	0	channel map gets cleared on resync	BPA 600
EncryptionSelection	NoEncryption=0, PinCodeASCII=1, PinCodeHex=2, LinkKey=3 SSP=4	0	Method of Encryption Used	BPA600
FilterOutNullsPolls	False=0, True=1	1	Filter out Nulls and Polls	BPA600
FilterOutSco	False-0, True=1	0	Filter out Sco frames	BPA600
LinkKey	32 digit hex number with 0x		Used only in LinkKeyMode	BPA600 Sodera
Master	12 digit hex number with 0x	“”	Master bdaddr	BPA600 Sodera

Table 3.2 - Classic Bluetooth Configuration Settings(continued)

Name	Values	Default	Description	ComProbe
PinCode	16 character ASCII string	“”	Used only in PinCodeAscii Mode For Sodera: The master and slave addresses and the PIN must be sent in the same config setting command. See the example in For Sodera: PinCode Settings on page 12.	BPA600 Sodera
PinCodeHEX	32 digit hex number with 0x	“”	Used only in PinCodeHex Mode	BPA600 Sodera
Slave	12 digit hex number with 0x	“”	Slave bdaddr	BPA600 Sodera
Slave2	12 digit hex number with 0x	“”	Slave bdaddr when SnifferUIMode=3 (Classic Only Multiple Connectioins). Slave address for the second device in multiple connection.	BPA 600
SnifferUIMode	Classic Only = 1 Ie Only = 2 Classic Only Multiple Connection = 3 Dual Mode = 0	1	Datasource modes	BPA 600

3.1.1.2 *Bluetooth* low energy

The following table lists the valid settings and corresponding values for *Bluetooth* low energy devices.

Table 3.3 - *Bluetooth* low energy Configuration Settings

Name	Values	Defaults	Description	ComProbe
leDevice	Address = 12 digit hex number with 0x Sync with first master = 0x1111000000000000	Empty string	low energy device address	BPA 600 BPAle Sodera
LongTermKey	32 digit hex number with 0x	Empty string	Long Term Key value for low energy encryption	BPA 600 BPAle Sodera

Table 3.3 - Bluetooth low energy Configuration Settings(continued)

Name	Values	Defaults	Description	ComProbe
PairingParameter	PIN = 6 digit decimal number OOB = 16 digit hex code	Empty String	PIN/OOB value for low energy encryption	BPA 600 BPAle Sodera
SnifferDiagnostics	False=0, True=1	0	Diagnostic information between the ComProbe device and PC capture.	BPA 600 BPAle

3.1.2 802.11 Wi-Fi

The following table identifies the configuration settings for Wi-Fi. The default settings for missing parameters are the existing configuration settings.

Table 3.4 - 802.11 Configuration Settings

Name	Value(s)	Description
Channel	1 - 165	Channel and Frequency both specifies the capture channel but in different ways.
Frequency	2412 - 5825	
ExtensionChannel	-1, (Below) 0, (None) +1 (Above)	This allows specifying a 40 MHz wide channel when capturing traffic in an 802.11n environment.
FcsFilter	0 (All) 1 (Valid) 2 (Invalid)	This allows specifying the if ComProbe hardware should capture frames with a bad checksum.
CaptureType	0 (Radio Tap Header) 1 (Per-Packet Information Header)	Determines the type of radio information to append to a frame.
EnableWEPDecryption	0 (False) 1 (True)	Specifies if hardware decryption should be enabled.

3.1.3 Config Settings Example

The following is an example of the command string for the Config Settings command. IOParameters, which is the 1st parameter, is the name of the configuration type.

```
CONFIG SETTINGS;IOParameters;Bluetooth;FilterOutSco=0;ClearChannelMapOnResync=1
```

Any configuration setting without a parameter name and value pair will be set to the default value from the table above. If a failure occurs processing the Config Settings command for the device, then a failure notification response is returned. A failure could result because of an error occurring within Frontline software as it attempts to process the command. The form for the failure notification response is the following.

```
CONFIG SETTINGS;FAILED;Timestamp=14:15:00,10/05/2006;Reason=Some Reason
```

A successful attempt to configure the IO for the devices results in a successful notification response. The form for the successful notification response is the following.

```
CONFIG SETTINGS;SUCCEEDED;Timestsamp=11:35:00
```

For Sodera: PinCode Settings

When sending a PIN code to Sodera, the master address, slave address and PIN must be sent in the same config settings command:

```
$FTE_CMD="Config
    Settings;IOParameters;Sodera;Master=".$Address1.";Slave=".$Address2.";PinCode=".$PIN;
```

3.2 Live Capture Commands

3.2.1 Start FTS

This command tells the Automation Server to launch the Frontline software application. A subsequent application launches on the machine, if the application is already running on the machine but was not launched by the Automation Server. A SUCCEEDED response is returned from the Automation Server if the application has been successfully launched previously or launches successfully with the current command. The Automation Server can send back the following responses to this command: FAILED, SUCCEEDED.

Caution: It is very important to note here that if you do not have the correct personality defined in the Start FTS command, then it will default to the personality associated with the client session, as defined in the configuration file. For example, Client 1 will default to Personality (which for *Bluetooth* installations is Interlaced Page Scan) if you do not change the configuration file to the correct personality. See Section 3.1 Config Settings on page 8 for more information on configurations settings.

3.2.2 Stop FTS

This command tells the ComProbe Automation Server to shut down the ComProbe software (if one is running) that was previously launched by the ComProbe Automation Server. Only an instance of the ComProbe software launched by the ComProbe Automation Server can be shut down using this command. If no ComProbe software was successfully launched, then a failure notification response will be sent from the server. If the attempt to shut down the ComProbe software times out, then a failure notification response is sent from the server.

The following is an example of the command string for this command.

```
Stop FTS
```

Failure to shut down the ComProbe software results in a failure notification response. The form for the failure notification response is the following.

```
Stop FTS;FAILED;Timestamp=11:35:00;10/05/2006;Reason=Some Reason
```

A successful attempt to shut down the ComProbe software will result in a successful notification response from the server. The form for the successful notification response is the following.

```
Stop FTS;SUCCEEDED;Timestamp=11:35:00
```

3.2.3 Go Live

The Go Live command tells ComProbe software to go live.

If an error occurs – which could be due to already being in live mode, a failure notification response is sent from the server. The form for the failure notification is the following.

Go live;FAILED;Timestamp=10/26/2009 5:11:45 PM;Reason=Already in live mode

A successful attempt going live results in a successful notification response. The form for the successful notification response is the following

Go live;succeeded;Timestamp=10/26/2009 5:09:42 PM

3.2.4 Exit Live Mode

The Exit Live Mode command tells ComProbe software to exit live mode.

If an error occurs – which could be due to already not being in live mode, a failure notification response is sent from the server. The form for the failure notification is the following.

Exit live mode;FAILED;Timestamp=10/26/2009 5:16:47 PM;Reason=No active capture file

A successful attempt going live results in a successful notification response. The form for the successful notification response is the following

Exit live mode;SUCCEEDED;Timestamp=10/26/2009 5:14:00 PM

3.2.5 Clear

The Clear command clears the capture buffer. The following is an example of the command string for this command.

Clear

If an error occurs - which could be due to (1) a time out waiting for the Clear command to process, (2) the ComProbe software not starting, or (3) the Stop Capture command not previously processed - then a failure notification response is sent from the server. The form for the failure notification response is the following.

Clear;FAILED;Timestamp=15:38:00,10/05/2006;Reason=Invalid command

A successful attempt to clear will result in a successful notification response. The form for the successful notification response is the following.

Clear;SUCCEEDED;Timestsamp=15:39:00

3.2.6 Start Capture

The Start Capture command starts the capture of data between devices being tested. The Start Capture command should only be sent after a successful FTS Start command has been processed. For Bluetooth devices only, the Start Capture command does not start the sniffing processing, which is the case for all other devices supported by ComProbe software. The following is an example of the command string for this command.

Start Capture

When you start ComProbe software, the client side is responsible for keeping track of the connections, not the server. For example, Client A is first connected to the ComProbe application and Client B is connected to a second instance of ComProbe application. Then both are disconnected. If Client B is reconnected first, it will be connected to the first instance of ComProbe application, not the second as it was originally.

The client is also responsible for the Sniff, Capture, and Sync On states.

If an error occurs - which could be due to (1) a time out waiting for the Start Capture command to process or (2) the ComProbe software has not been started, then a failure notification response is sent from the server. The form for the failure notification response is the following.

Start Capture;FAILED;Timestamp=14:25:00,10/05/2006;Reason=FTS not started

A successful attempt to start capturing will result in a successful notification response. The form for the successful notification response is the following.

Start Capture;SUCCEEDED;Timestsamp=14:26:00,10/05/2006

3.2.7 Stop Capture

The Stop Capture command stops the capturing of data between devices. The Stop Capture command should only be sent if successful Start FTS and Start Capture commands have previously been processed. Similar to the Start Capture command, the Stop Capture command does not stop the sniffing of Classic Bluetooth devices and is not required for stopping the sniffing of Classic Bluetooth devices. The following is an example of the command string for this command.

Stop Capture

If an error occurs - which could be due to (1) a time out waiting for the Stop Capture command to process, (2) the CPAS application has not been started, or (3) the Start Capture command has not previously been processed - then a failure notification response is sent from the server. The form for the failure notification response is the following.

Stop Capture;FAILED;Timestamp=15:38:00,10/05/2006;Reason=FTS not in capture mode

A successful attempt to stop capturing will result in a successful notification response. The form for the successful notification response is the following.

Stop Capture;SUCCEEDED;Timestsamp=15:39:00

3.2.8 Save Capture

The Save Capture command saves the captured data after a capture was started, then stopped with the Stop Capture command described below. This command has an optional parameter for setting the file name the captured data will be saved to. If no parameter is sent, the default capture file name provided in the application configuration file described above will be used.

Save Capture;c:\program files\common files\fte\my capture files\mycap.cfa

There are some conditions in which this command will fail. If an error occurs - which could be due to no capture data being present when the command is sent - then a failure notification response is sent from the server. The form for the failure notification response is the following.

Save Capture;FAILED;Timestamp=14:25:00,10/05/2006;Reason=Cannot save to disk, actively capturing or no capture data to save.

A successful attempt to save captured data will result in a successful notification response. The form for the successful notification response is the following.

Save Capture;SUCCEEDED;Timestamp=14:26:00,10/05/2006

3.3 Capture File Analysis/Reporting

3.3.1 Open Capture File

The Open Capture File command tells ComProbe software to open the specified capture file. The Open Capture File command has one parameter indicating the path and name of the file to open. The following is an example of the command string for this command.

Open Capture File;C:\MyFiles\newfile.cfa

If an error occurs – which could be due to the file not existing, a failure notification response is sent from the server. The form for the failure notification response is the following.

Open Capture File;FAILED;Timestamp=10/26/2009 4:13:54 PM;Reason=File (C:\temp\abc.cfa) does not exist

A successful attempt opening the capture file results in a successful notification response. The form for the successful notification response is the following.

Open capture file;SUCCEEDED;Timestamp=10/26/2009 4:27:38 PM

By default, the successful notification response is sent immediately after the capture file is opened successfully. However the frame compiler may run for several minutes after a large capture file is opened. If an HTML Export command or Export command is sent before the frame compiler completes, the exported file will not contain all the frames.

If it is important to wait for the frame compiler to complete before proceeding the optional Notify= parameter specification can be used to delay sending the success notification. The format for using the Notify= parameter is as follows:

Notify=< 0 | 1>

Where:

0 : Indicates the default behavior of sending the success notification immediately after the capture file is successfully opened.

1 : Indicates to send the success notification after the frame compiler has finished compiling all the frames.

Following is an example of the command format to wait until the frame compiler has finished would:

Open Capture File;C:\users\public\documents\Frontline Test Equipment\My Capture Files\my cap.cfa;Notify=1

If an invalid Notify= value is specified the following failure notification response is sent:

OPEN CAPTURE FILE;FAILED;Timestamp=8/23/2012 4:03:34 PM;Reason=Reason=Invalid Notify option

3.3.2 Close Capture File

The Close Capture File command closes the capture file.

Close Capture File

If an error occurs – which could be due to not having an open file, a failure notification response is sent from the server. The form for the failure notification response is the following.

Close capture file; FAILED;timestamp=10/26/2009 4:35:17 PM;Reason=No active capture file

A successful attempt opening the capture file results in a successful notification response. The form for the successful notification response is the following

Close capture file;SUCCEEDED;Timestamp=10/26/2009 5:05:09 PM

3.3.3 Add Bookmark

The Add Bookmark command allows you to add a bookmark to the last frame currently in the capture buffer at the time the command is processed OR optionally to a specified frame number.

A string representing the bookmark id must be specified using the String= parameter specification.

Add Bookmark;String=Bookmark#1

A bookmark must not already exist for the frame or a failure notification response is sent:

Add Bookmark;FAILED;Timestamp=1/30/2013 10:18:01 AM;Reason=Frame already has bookmark

An optional frame number may be specified to set a bookmark on a specific frame.

```
Add Bookmark;String=Bookmark#1;Frame=1024
```

The frame number must be valid or a failure notification response is sent:

```
Add Bookmark;FAILED;Timestamp=1/30/2013 10:15:17 AM;Reason=Frame does not exist
```

When a bookmark is successfully added, the response notification will contain the frame number that the bookmark was added to. The following example shows that a bookmark was added to frame #3066.

```
Add Bookmark;SUCCEEDED;Frame=3066;Timestamp=1/30/2013 10:17:58 AM
```

3.3.4 Modify Bookmark

The Modify Bookmark command allows you to modify or delete an existing bookmark.

If no parameters are specified, the Modify Bookmark command will delete an existing bookmark on the last frame.

```
Modify Bookmark
```

An optional string representing the bookmark id can be specified using the String= parameter specification to change the name of an existing bookmark.

```
Modify Bookmark;String=NewBookmark#1
```

An optional frame number may be specified to modify a bookmark on a specific frame.

```
Modify Bookmark;String=NewBookmark#1;Frame=1024
```

The frame number must be valid or a failure notification response is sent:

```
Modify Bookmark;FAILED;Timestamp=1/30/2013 10:46:11 AM;Reason=Frame does not exist
```

When a bookmark is successfully modified, the response notification will contain the frame number that the bookmark was added to. The following example shows that a bookmark was modified for frame #3066.

```
Modify Bookmark;SUCCEEDED;Frame=3066;Timestamp=1/30/2013 10:44:05 AM
```

3.3.5 Export HTML

The HTML Export command tells the ComProbe software to invoke the Frame Display. It then automatically selects the HTML Export menu option. Using the Export HTML command parameters, the user can choose whether to include the summary pane information, data bytes, and frame range. Also, the user only has to specify a name for the saved HTML file. The saved html file is always written to the default directory which is: \\documents and settings\all users\documents\frontline test equipment\my log files.

If you enter a file name with an invalid character “\/:*?<>”, then the system automatically generates a file name using:

```
$(FrameDisplayPrint-<process id>-<counter>).htm
```

The following is an example of the command string for this command. If no frame range is specified, all frames will be exported.

```
HTML Export;Summary=<1 | 0>;Data Bytes=<1 | 0>;Decode=<1|0>;Frame Range
    Upper=30000;Frame Range Lower=1;File=htmlexport1.htm
```

There are several important ideas to remember about HTML Export.

- Be aware that this can cause the ComProbe software to spend considerable time generating the html file.
- The Frame Display window remains open once the export is complete.

- The frame range is a positive number followed by a hyphen, followed by another positive number greater or equal to the first positive number.
- If no html file is specified, a default one will be generated and saved in the log file directory created during installation of the ComProbe software.
- If an html file is specified, then this file is created and saved in the log file directory created during installation of the ComProbe software.
- If no summary is specified, summary information will not be included by default.
- If no data bytes are specified, data bytes will be included by default. Decode will always be included if Data Bytes are included.
- All layers for each frame will be exported. The user cannot select the layers.
- If the capture file is empty, no html is generated and an error is reported.
- If the range selected is outside the current range of processed frames, no html is generated and the last process frame is included in the reason.
- If all frames need to be exported, then select some unknown large number for the upper range.

Table 3.5 - Export HTML Command Parameter Default Values

Parameter	Default Values
Summary	0
Data Bytes	1
Decode	Data Bytes Value

An error can occur if there are no frames to export. If a failure occurs, a failure notification response is sent. The form for the failure notification response is the following.

```
HTML Export;FAILED;Timestamp=10:43:00,10/05/2006;Reason=Some Reason
```

A successful attempt to export returns a successful notification response. The form for the successful notification response is the following.

```
HTML Export;SUCCEEDED;Timestamp=5/18/2007 10:44:00 AM
```

3.3.6 Export

The Export command tells the CPAS application to invoke the Frame Display. It then automatically selects the Export menu option.

A file name must be specified using the File= parameter specification. If just a file name without a path is specified, the saved file will be written to the default directory which is: \\documents and settings\all users\documents\frontline test equipment\my log files.

```
Export;File=export1.csv
```

The file extension must be .csv or a failure notification response is sent:

```
Export;FAILED;Timestamp=8/23/2012 2:29:38 PM;Reason=Invalid file extension : export1.txt
```

An optional file path may be specified.

```
Export;File= C:\Users\JSmith\Documents\export1.csv
```

The path must be valid or a failure notification response is sent:

```
Export;FAILED;Timestamp=8/23/2012 2:40:37 PM;Reason=Failed to create file ( may be Read-only
    ): c:\users\jsmith\documents\export1.csv
```

By default the Export command will wait until all frames have been compiled before beginning to export the data. However in the scenario where the Export command is sent while frames are still being captured this behavior can be modified using the Mode= parameter specification. The format for using the Mode= parameter is as follows:

Mode=< 0,1>

Where

0 : indicates the default behavior which is to wait for the frame compiler to complete.

1 : indicates to export the frames that are currently in the Frame Display (i.e. do not wait for the frame compiler to complete).

Example of using the Mode parameter:

```
Export;File= C:\Users\JSmith\Documents\export1.csv;Mode=0
```

The Mode parameter must be valid or a failure notification response is sent:

Command:

```
Export;File= C:\Users\JSmith\Documents\export1.csv;Mode=2
```

Response:

```
Export;FAILED;Timestamp=8/23/2012 2:40:37 PM;Reason=Invalid Mode parameter: Mode=2
```

> **Note:** Note that if the default behavior of waiting for the frame compiler to complete is desired, then it is not necessary to specify the “Mode=0” parameter. However it is important to understand that if the Mode parameter is not specified and the sniffer is still running and capturing new frames, then the Export command may wait indefinitely.

The tab currently selected on the Frame Display is what gets exported. A specified tab can be selected using the optional Tab= parameter specification. The format for using the Tab= parameter is as follows:

Tab= < Technology >:< Layer Name >

Where < Technology > is one of the following:

Table 3.6 - Technology Field Values

Value	Technolgy
"General”	// Ethernet
"Classic"	// Classic Baseband
"LE"	// LE BaseBand
"80211"	// 802.11 Radio
"USB"	// USB
"SD"	// SDIO/SPI

Where < Layer Name > is one of the following:

> **Note:** Note: in general the Layer Name matches what you see on the Tab in the Frame Display.

Table 3.7 - Layer Names Values

802.2	802.11 AMP	802.11 AMP
802.11 MAC	802.11 Radio	802.15.4 MAC
802.1D	802.1X	A2DP
AB CSP	AB DF1 Full Duplex/Radio Modem (BCC)	AB DF1 Full Duplex/Radio Modem (CRC)
AB DF1 Half Duplex (BCC)	AB DF1 Half Duplex (CRC)	AB PCCC
AMP Manager"SDP	ARP	ATT
AVCTP	AVDTP	AVDTP Media
AVDTP Recover	AVDTP Report	AVDTP Signaling
AVRCP	BACnet APDU	BACnet MSTP Frame
BACnet Network Layer	BACnet NPDU	BACnet PTP Frame
BACnet/IP	Baseband	BCCMD
BlueCore Serial Protocol	Bluetooth FHS	Bluetooth PRP
Bluetooth USB	Bluetooth Virtual Transport	BNEP
BSAP Link Layer	BSAP Network Control	BSAP Transport
BT-HID	CAN 2.0A	CAN 2.0A/2.0B
CAPI	Chipcard Bulk Messages	CIP
CL-DceRpc	CMTP	CO-DceRpc
Context Missing	ControlNet MAC	DeviceNet
DeviceNet Explicit	DH+ (Async)	DH+ (U2DHP)
DH-485	DLR	DNP3 Application
DNP3 Transport	DNP3-Eth Link Layer	DNP3-Eth Raw Link Layer
DNP3-Serial Link Layer	Encapsulated AsyncPPP	Ethernet
Extended Inquiry Response	FAX	Frame Info
FTP	H4DS	Hands-Free
HCI	HCI UART	HCRP Contro
HCRP Data	Headset	HID Keyboard
HID Mouse	ICMP	IEC-60870-101 Application
IEC-60870-101 Link	IEC-60870-102 Application	IEC-60870-102 Link
IEC-60870-103 Application	IEC-60870-103 Link	IEC-60870-104 Application
IEEE P11073 20601	IGMP	IPv4
IPv6	ISO-DEP	L2CAP
LE ADV	LE BB	LE DATA

Table 3.7 - Layer Names Values(continued)

LE LL	LE PKT	LLC 802.2
LLDP	LMP	LPacket
Mass Storage Bulk Only Transport	MCAP Control	Modbus
Modbus ASCII 2 Wire	Modbus ASCII Master	Modbus ASCII Slave
Modbus RTU 2 Wire	Modbus RTU Half Duplex	Modbus RTU Master
Modbus RTU Slave	Modbus/TCP	NFC
NFC-A	NFC-B	NFC-DEP
NFC-F	OBEX	Obsolete S-Bus Presentation
OPP	PN-CBA	PN-IO-ContextMgr
PN-IO-DCP	PN-IO-MRRT	PN-IO-PTCP
PN-IO-RTAcyclic	PNIO-RT-CBA	PN-IO-RTCyclic
PN-MRP	PPP	PreConnection
PROFINET	PTP/MTP	PTS
RA USBCIP	RFCOMM	ROC Master
ROC Plus Master	ROC Plus over Ethernet	ROC Plus Slave
ROC Slave	RTCP	RTP
S-Bus Data Mode	S-Bus Data Mode (Secure Mode base)	S-Bus Presentation
S-Bus RS-232 Parity Mode	S-Bus RS-485 Parity Mode	S-Bus Secure Data Mode
SCO/eSCO	SCSI Primary Commands	SDIO/SPI
SDIO/SPI/HCI	SIM ACCESS	SIM Application
SMP	SNAP	Tag Type 1
Tag Type 2	Tag Type 3	Tag Type 4A
Tag Type 4B	TCP	TCS
Three-Wire UART	UDP	USB
USB Data and Handshake Messages	USB Pre Messages	USB Setup
USB Split Message	USB Token Message	UWB Radio
VCP	VDP	Virtual Sniffer
VJC	VJU	WiMedia
Wireless USB	WUSB	ZigBee APS
ZigBee Cluster Library	ZigBee Device Profile	ZigBee Network

As an example, if you wanted to export the “802.11 MAC” layer from your capture file generated by the ComProbe 802.11 sniffer you would format your Export command as follows:

Export;File=export1.csv;Tab=80211:802.11 MAC

OR as another example, export the SCO/eSCO layer captured by the BPA500

Export;File=export1.csv;Tab=Classic:SCO/eSCO

> **Note:** If your capture file does not contain the specified layer name OR you make an error in entering the layer name then the export command will export the currently selected layer. In other words there will not be an error notification for this case.

There are several important ideas to remember about Export.

- Be aware that this can cause the ComProbe software to spend considerable time generating the file.
- The Frame Display window will close once the export is complete.

3.4 *Bluetooth* Commands

3.4.1 Start Sniffing

The Start Sniffing command starts the sniffing of data between Classic *Bluetooth* and *Bluetooth* low energy devices. This command has no effect on other Frontline device types. This command should only be sent if a successful Start FTS command has previously been processed. The following is an example of the command string for this command.

Start Sniffing

If an error occurs - which could be due to (1) the devices being tested are not *Bluetooth* devices, (2) the devices cannot be synchronized or (3) a timeout occurs trying to synchronize the device, etc. - then a failure notification response is sent. The form for the failure notification response is the following.

Start Sniffing;FAILED;Timestamp=15:54:00,10/05/2006;Reason=Devices cannot be synchronized

A successful attempt to start sniffing results in a successful notification response. The form for the successful notification response is the following.

Start Sniffing;SUCCEEDED;Timestamp=15:56:00

> **Note:** For Bluetooth devices only, the Start Capture command does not start the sniffing processing, which is the case for all other devices supported by ComProbe software.

3.4.2 Stop Sniffing - Classic *Bluetooth* Only

The Stop Sniffing command stops the sniffing of data between Classic*Bluetooth* devices only. This command has no effect on other ComProbe device. This command should only be sent if successful Start FTS and Start Sniffing commands have previously been processed. The following is an example of the command string for this command.

Stop Sniffing

If an error occurs - which could be due to (1) a time out waiting for the Stop Sniffing command to process, (2) the devices being test are not Bluetooth devices, or (3) the Start Sniffing command was not previously sent or not successful., then a failure notification response is sent. The form for the failure notification response is the following.

Stop Sniffing;FAILED;Timestamp=10:41:00,10/06/2006;Reason=Not in sniffing mode

A successful attempt to stop sniffing results in a successful notification response. The form for the successful notification response is the following.

Stop Sniffing;SUCCEEDED;Timestamp=10:42:00

3.4.3 Start Record - Sodera only

The Start Record command starts the capture of data between Classic *Bluetooth* and *Bluetooth* low energy devices. This command has no effect on other Frontline device types. This command should only be sent if a successful Start FTS command has previously been processed. The following is an example of the command string for this command.

```
Start Record
```

If an error occurs - which could be due to (1) the devices being tested are not Bluetooth devices, (2) the devices cannot be synchronized or (3) a timeout occurs trying to synchronize the device, etc. - then a failure notification response is sent. The form for the failure notification response is the following.

```
Start Record;FAILED;Timestamp=15:54:00,10/05/2006;Reason=Devices cannot be synchronized
```

A successful attempt to start Record results in a successful notification response. The form for the successful notification response is the following.

```
Start Record;SUCCEEDED;Timestamp=15:56:00
```

3.4.4 Start Analyze - Sodera Only

The Start Analyze sends data packets going to or from the device addresses configured up to the ComProbe software for analysys.

```
Start Analyze
```

If an error occurs:

```
Start Analyze;FAILED;Timestamp=15:54:00,10/05/2006;Reason=Devices cannot be synchronized
```

A successful attempt to start analyzing results in a successful notification response. The form for the successful notification response is the following.

```
Start Analyze;SUCCEEDED;Timestamp=15:56:00
```

> **Note:** It is advisable to add a short (5 second) pause between the Start Record and Start Analyze commands.

3.4.5 Stop Record - Sodera Only

The Stop Record command stops the recording of data between Classic Bluetooth devices and low energy Bluetooth devices. This command has no effect on other Frontline devices. This command should only be sent if successful Start FTS and Start Record commands have previously been processed. The following is an example of the command string for this command.

```
Stop Record
```

A successful attempt to stop Record results in a successful notification response. The form for the successful notification response is the following.

```
Stop Record;SUCCEEDED;Timestamp=15:56:00
```

3.4.6 Stop Analyze - Sodera Only

The Stop Analyze command stops the sending of data Classic Bluetooth and low energy devices to the ComProbe software. This command has no effect on other Frontline device. This command should only be sent if successful Start FTS and Start Analyze commands have previously been processed. The following is an example of the command string for this command.

```
Stop Analyze
```

If an error occurs:

Stop Analyze;FAILED;Timestamp=15:54:00,10/05/2006;Reason=Not in sniffing mode.

A successful attempt to stop sniffing results in a successful notification response. The form for the successful notification response is the following.

Stop Analyze;SUCCEEDED;Timestamp=15:56:00

3.4.7 Sync Status

The Sync Status command directs Frontline software to send the status of the *Bluetooth* devices being tested every time it changes. This command can also tell Frontline software to stop sending the status of the *Bluetooth* devices being tested. This command can only be used when testing Classic *Bluetooth*.

> **Note:** Once you turn on Sync Status, it sets a flag in Frontline software to export the state. If the state of any device changes, you are notified.

The following table identifies what each status color means.

Table 3.8 - Sync Status Color Codes

Color Code	Sync Status Description
Red	Unknown, Pending, Halted (State=0-2)
Green	Waiting for master to connect to the slave (State=4)
Blue	Synchronized with the master clock – link active (State=5)
Gray	Synchronized with the master clock – link inactive (State=6)
Yellow	Waiting for the master to resume transmission (State=7)

The following is an example of the command string for this command. The ON parameter for the command tells Frontline software to send the status; the OFF parameter tells Frontline software to stop sending the status. This command is not supported by the ComProbe low energy product.

Sync Status;On	(returns and turns on the subscription for all links)
Sync Status;On;1	(returns and turns on the subscription for link 1)
Sync Status;On;1,2	(returns and turns on the subscription for links 1 & 2)
Sync Status;Off	(turns off the subscription for all)

You can only subscribe once at a time. You must unsubscribe before subscribing again.

For Frontline BPA 500 only, the status is never returned for the gray state (6) which is synchronized with the master clock - link inactive. State 5 which is blue is only returned when the link initially goes in sync.

For Frontline FTS4BT all the states are returned.

If you have 2 links up and you subscribe for the first one, you will only get status for that link.

An error can occur if the devices under test are (1) not *Bluetooth* devices, (2) an invalid command is sent, or (3) an invalid synchronization state is passed back from Frontline software, etc. If a failure occurs, a failure notification response is sent. The form for the failure notification response is the following.

Sync Status;FAILED;Timestamp=10:43:00,10/05/2006;Reason=Invalid synchronization state

A successful attempt to sync the status results in a successful notification response. The form for the successful notification response is the following.

Sync Status;SUCCEEDED;Timestamp=5/18/2007 10:44:00 AM;State=1

3.5 Data Extraction Plug-In Commands

This section describes automation commands and statuses for the Data Extraction Plug-in. The plug-in pulls data from various decoded Bluetooth protocols and profiles. Once you have extracted the data, you can save them into different file types, such as text files, graphic files, email files, .mp3 files, and more.

Automation commands are sent by the client to the plug-in.

Automation statuses are sent by the plug-in to the client.

The Open Capture File on page 14, command is needed in order to use the plug-in.

A Data Extraction Plug-in command has the following format (multiple parameters are comma-separated):

```
Plugin Command;Plugin Name=Data and Audio
    Extraction;Command=command;Parameters=parameter(s)
```

A status message has one of the following formats:

```
Plugin Command;Plugin Name=Data and Audio
    Extraction;SUCCEEDED;timestamp;State=state;Description=description[;label=value]...
Plugin Command;Plugin Name=Data and Audio
    Extraction;FAILED;timestamp;Reason=reason;Description=description [;label=value]...
```

A status message is sent immediately after the command is received. If the command will take time to execute, one or more additional status messages are sent later.

The State and Reason fields indicate the meaning of each SUCCEEDED and FAILED message respectively. Where possible, corresponding State and Reason values are the same except that the Reason value has 9000 added to it. The Description field contains a description of the State or Reason value. Additional fields are copied from the original command for reference.

Table 3.9 - State and Description for SUCCEEDED/FAILED Message

Status	State	Description
SUCCEEDED	10	Command completed
SUCCEEDED	11	Command accepted
SUCCEEDED	30	Extraction completed
SUCCEEDED	40	File opened
SUCCEEDED	41	File closed
SUCCEEDED	60	In live mode
SUCCEEDED	63	In standby mode
FAILED	9010	Invalid syntax (explanation)
FAILED	9011	Unknown command
FAILED	9012	Missing parameter (explanation)
FAILED	9013	Invalid parameter (explanation)
FAILED	9014	Too many parameters

Table 3.9 - State and Description for SUCCEEDED/FAILED Message(continued)

Status	State	Description
FAILED	9020	Path does not exist
FAILED	9021	Invalid basename
FAILED	9030	No data found to extract
FAILED	9040	File open error
FAILED	9041	File close error
FAILED	9042	File write error
FAILED	9043	File does not exist
FAILED	9044	File not open
FAILED	9060	Already in live mode
FAILED	9061	Already not in live mode
FAILED	9062	Unable to enter live mode
FAILED	9063	Unable to exit live mode
FAILED	9998	Still executing previous user command
FAILED	9999	Still executing previous automation command

3.5.1 Open Files After Extraction

The Open Files After Extraction command tells the plug-in whether to open files after extraction.

The parameter is Yes or No. If this command is not sent, the plugin defaults to No.

```
Plugin Command;Plugin Name=Data and Audio Extraction;Command=Open Files After
    Extraction;Parameters=Yes
```

The plug-in immediately responds with the following:

```
Plugin Command;SUCCEEDED;timestamp;State=10;Description=Command
    completed;Plugin=Data Extraction;Command=Open Files After Extraction;Parameters=Yes
```

3.5.2 Two Mono Files

The Two Mono Files command tells the plugin whether to produce two mono files or one stereo file when extracting SCO. The parameter is Yes or No. If this command is not sent, the plugin defaults to Yes.

```
Plugin Command;Plugin Name=Data and Audio Extraction;Command=Two Mono
    Files;Parameters=Yes
```

The plugin immediately responds with the following:

```
Plugin Command;SUCCEEDED;timestamp;State=10;Description=Command
    completed;Plugin=Data Extraction;Command=Two Mono Files;Parameters=Yes
```

3.5.3 Convert to Linear PCM

The Convert To Linear PCM command tells the plug-in whether to convert to linear PCM. The parameter is Yes or No. If this command is not sent, the plug-in defaults to No.

```
Plugin Command;Plugin Name=Data and Audio Extraction;Command=Convert To Linear
    PCM;Parameters=Yes
```

The plug-in immediately responds with the following:

```
Plugin Command;SUCCEEDED;timestamp;State=10;Description=Command
    completed;Plugin=Data Extraction;Command=Convert To Linear PCM;Parameters=Yes
```

3.5.4 Output Path

The Output Path command tells the plug-in the path for output files. The parameter is the path. If this command is not sent, the plug-in defaults to the path of the capture file.

```
Plugin Command;Plugin Name=Data and Audio Extraction;Command=Output
    Path;Parameters=C:\MyFiles\Data Extraction
```

The plug-in immediately responds with the following, if the path exists:

```
Plugin Command;SUCCEEDED;timestamp;State=10;Description=Command
    completed;Plugin=Data Extraction;Command=Output Path;Parameters=C:\MyFiles\Data
    Extraction
```

The plug-in immediately responds with the following if the path does not exist:

```
Plugin Command;FAILED;timestamp;Reason=9020;Description=Path does not exist;Plugin=Data
    Extraction;Command=Output Path;Parameters=C:\MyFiles\Data Extraction
```

See **Output Basename**.

3.5.5 Output Basename

The Output Basename command tells the plug-in the basename (i.e. file name with no extension) for output files. The parameter is the basename. If this command is not sent, the plug-in defaults to the basename of the capture file.

> **Note:** The extracted data may specify a basename, in which case this command is ignored.

```
Plugin Command;Plugin Name=Data and Audio Extraction;Command=Output
    Basename;Parameters=alpha 002
```

The plug-in immediately responds with the following if the basename is valid:

```
Plugin Command;SUCCEEDED;timestamp;State=10;Description=Command
    completed;Plugin=Data Extraction;Command=Output Basename;Parameters=alpha 002
```

The plug-in immediately responds with the following if the basename is invalid:

```
Plugin Command;FAILED;timestamp;Reason=9021;Description=Invalid basename;Plugin=Data
    Extraction;Command=Output Basename;Parameters=alpha*002
```

See **Output Path**.

3.5.6 Extract

The Extract command tells the plug-in to extract the specified profiles / protocol (SCO is a protocol, everything else is a profile). There is no default.

```
Plugin Command;Plugin Name=Data and Audio
    Extraction;Command=Extract;Parameters=profiles/protocol
```

Profiles/protocol consists of a comma-separated list of one or more of the following:

- A2DP - Advanced Audio Distribution Profile
- BIP - Basic Imaging Profile
- BPP - Basic Printing Profile
- FTP - File Transfer Profile
- HCRP - Hard-Copy Cable Replacement Profile
- OPP - Object Push Profile
- PBAP - Phone Book Access Profile
- SCO - Synchronous Connection-Oriented link protocol
- SPP - Serial Port Profile
- SYNCH - Synchronization Profile

Example command:

```
Plugin Command;Plugin Name=Data and Audio
    Extraction;Command=Extract;Parameters=SCO,BIP
```

In this example, the plug-in immediately responds with the following:

```
Plugin Command;SUCCEEDED;timestamp;State=11;Description=Command
    accepted;Plugin=Data Extraction;Command=Extract;Parameters=SCO,BIP
```

In this example, the plug-in subsequently sends a separate status for each profile / protocol as extraction completes.

```
Plugin Command;FAILED;timestamp;Reason= 9030;Description=No data found to
    extract;Profile=BIP;Plugin=Data Extraction;Command=Extract;Parameters=SCO,BIP
Plugin Command;SUCCEEDED;timestamp;State=30;Description=Extraction
    completed;Protocol=SCO;Plugin=Data Extraction;Command=Extract;Parameters=SCO,BIP
```

Contacting Technical Support

Technical support is available in several ways. The online help system provides answers to many user related questions. Frontline's website has documentation on common problems, as well as software upgrades and utilities to use with our products.

On the Web: http://fte.com/support/supportrequest.aspx

Email: tech_support@fte.com

If you need to talk to a technical support representative about your ComProbe product, support is available between 9 am and 5 pm, U.S. Eastern Time zone, and between 9 am and 5 pm, Pacific Time zone, on Monday through Friday. Technical support is not available on U.S. national holidays.

Phone: +1 (434) 984-4500

Fax: +1 (434) 984-4505

Instructional Videos

Teledyne LeCroy provides a series of videos to assist the user and may answer your questions. These videos can be accessed at fte.com/support/videos.aspx. On this web page use the **Video Filters** sidebar to select instructional videos for your product.

Appendix 1: Sample Automation Script

The following script is provided as a typical automation application. The script is for demonstration and education purposes only. This sample may be copied and edited using common text editors, however Frontline is not responsible for the adverse results.

The sample script is written in TCL[1]. The script can be translated to any general purpose programming language such as C++.

Read the comments in the script for guidance in developing a script for your applications. You can always contact Frontline technical support for assistance with your ComProbe Automation Server applications.

```
#
# The following procedures are intended to provide the basics for writing TCL
# scripts that interface to the FTSAutoServer application.
#
# At the bottom of this file is a sample mainloop that utilizes some of these
# procedures.
#
# When you use these procedures in a script, the output has some formatted notations
# that attempt to help visualize what is going on.
#
# Those notations are as follows:
#
# >>> denotes a cmd being sent to the FTSAutoServer application
# <<< denotes a cmd response being received from the FTSAutoServer application
# ..... denotes a Sync Status response being received from the FTSAutoServer
# ### denotes a debug comment
# !!! denotes a Warning or Error condition
#
# There is a set of Command Wrappers for all the automation commands that are
# supported by FTSAutoServer application. Each wrapper has a default timeout
# value in msecs. A different timeout value can be passed to the Command Wrapper
# that will override the default value.
```

[1]tcl files are based on Tool Command Language (TCL). TCL is an open-source, cross-platform programming language. More information is available at www.tcl.tk.

```
#

#################################################################
# Procedures
#################################################################
proc FTEBaseInit { ipAddr } {

    global Connections
    global Buffer
    global Cid

    set Connections(Host) $ipAddr
    puts "IP Address: $ipAddr"
    puts "Buffer size: [string length $Buffer]\n";
    # Setup up the TCP/IP connection
    set Cid [Connect $Connections(Host) $Connections(Port)];
    # Setup various fileevent handler(s)
    fileevent $Cid readable "ReadInput $Cid"

}

    proc FTEBaseCleanup {} {


        global Cid

        # Close the TCP/IP connection with the CPAS automation server
        Disconnect $Cid
        puts "FtsAutoServer Disconnected"

    }

# Connect to the automation server TCP/IP socket.
proc Connect { host port } {

    if { [catch { socket $host $port } server] } {

        puts stderr "Failed to connect to $host on $port"
        return ""

    }

    return $server

}
# Disconnect from the automation server TCP/IP socket.
proc Disconnect { cid } {


    if { ![catch { fileevent $cid readable {} }] } {

        close $cid

    }

}
# Read a status from the automation server TCP/IP socket.
proc ReadInput { cid } {


    global SyncStatusCmd
    global CmdStatusArray
    global CmdArrayHead
```

```
        set tempStatus [gets $cid]
        # If it is a Cmd Status
        if { [string first $SyncStatusCmd $tempStatus] == -1 } {

            # Add to Cmd Status Array
            array set CmdStatusArray [list $CmdArrayHead $tempStatus]
            incr CmdArrayHead

        # Else it must be a Sync Status
        } else {

            # Process Sync Status immediately
            SyncStatusCmdHandler $tempStatus


        }

    }

    # Write command to automation server TCP/IP socket.
    proc WriteOutput { cid cmd } {

        puts $cid $cmd

        flush $cid
        fileevent $cid writable {}

    }


    proc SyncStatusCmdHandler { syncStatus } {


        global SyncStatusArray

        ### Debug
        ###puts "\n### SyncStatusCmdHandler( $syncStatus )\n"

        # Convert Status to lower case
        set status [string tolower $syncStatus]

        set conn ""
        set state ""

        # Look for string "state="
        set index1 [string first "state=" $status]
        # Look for ","
        set index2 [string first "," $status $index1]
        set length [string length $status]

        if { $index1 == -1 || $index2 == -1 } {


            # Could not find a valid sync status format
            return 0

        }

        set index1 [expr $index1 + 6]
```

```
set index2 [expr $index2 - 1]

# Parse out the connection
set conn [string range $status $index1 $index2]

set index1 [expr $index2 + 2]
set index2 [expr $length - 1]

# Parse out the state
set state [string range $status $index1 $index2]

# Get previous sync state for this connection
set prevState [ array get SyncStatusArray $conn ]

# Update the Sync Status array with the new state for this connection
array set SyncStatusArray [list $conn $state]

# Parse out the state
set length [string length $prevState]
if { $length > 0 } {

    set index1 [expr [string first " " $prevState] + 1]
    set index2 [expr $length - 1]
    set prevState [string range $prevState $index1 $index2]

} else {

    set prevState 0

}

### Debug
###puts "### status : $status"
###puts "### conn : $conn"
###puts "### state : $state"
###puts "### prevState : $prevState"

#
# At this point we have the current and previous state for this connection
#

puts "\n....."
puts "..... $syncStatus"
puts [format "..... Sync Status\[%d\] changed : %s => %s" $conn [SyncStateColorLookup
      $prevState] [SyncStateColorLookup $state] ]

# Display various user tips on certain state changes
if 1 {

    if { $state == 4 && $prevState < 4 } {

        puts "..... Now connect your DUTs."

    }

    if { $state == 4 && $prevState == 5 } {

        puts "..... First send StopSniffing command."
        puts "..... Then send SyncStatus OFF command."
```

```
            }
        }

        puts "......\n"

        # Add any additional handler code here
    }


    proc GetSyncStateValue { conn } {

        global SyncStatusArray

        # Get the sync state for this connection
        set state [ array get SyncStatusArray $conn ]

        # Parse out just the state
        set length [string length $state]
        if { $length > 0 } {

            set index1 [expr [string first " " $state] + 1]
            set index2 [expr $length - 1]
            set state [string range $state $index1 $index2]

        } else {

            set state 0

        }

        return $state

    }


    namespace export GetSyncStateColor
    proc GetSyncStateColor { conn } {


        set state [GetSyncStateValue $conn]
        return [SyncStateColorLookup $state]

    }

    namespace export SyncStateColorLookup
    proc SyncStateColorLookup { state } {


        #
        # Note - the state input parameter can be a color that will be converted
        # to a valure OR it can be a value that will be converted to a color.
        #

        if { ![string is integer $state] } {

            set state [string tolower $state]

        }
        switch $state {
```

```
0 {
    set retVal "Red"
}
1 {
    set retVal "Red"
}
2 {
    set retVal "Red"
}
"red" {
    set retVal 2
}
4 {
    set retVal "Green"
}
"green" {
    set retVal 4
}
5 {
    set retVal "Blue"
}
"blue" {
    set retVal 5
}
6 {
    set retVal "Grey"
}
"grey" {
    set retVal 6
}
7 {
    set retVal "Yellow"
}
```

```
            "yellow" {
                set retVal 7
            }
            default {
                if { [string is integer $state] } {
                    set retVal "Uninitialized"
                } else {
                    set retVal 0
                }
            }
        }
        return $retVal
    }


    namespace export UnexpectedStatusHandler
    proc UnexpectedStatusHandler { unexpectedStatus } {

        ### Debug
        puts "\n### UnexpectedStatusHandler( $unexpectedStatus )\n"

        # Add any additional handler code here

    }

    proc SendCmd { cmd params } {

        global Cid

        # Combine the Cmd and Parameters to form a single cmdLine
        set cmdLine [format "%s;%s" $cmd $params]

        puts [format ">>> %s\n" $cmdLine]

        WriteOutput $Cid $cmdLine

    }
    namespace export GetCmdStatus
    proc GetCmdStatus { cmd {timeout 0} } {

        global ConfigSettingsCmd
        global StartSniffingCmd
        global StopSniffingCmd
        global SyncStatusCmd
        global NumberDatasources
        global Status
```

```
    #
    # Note : cmds that are handled by the Datasource will have a status for each
    # datasource in the setup.
    # Cmds that are handled by the Core will only have one status.
    #

    # Check for cmds that are handled by the Datasource
    if { $cmd == $ConfigSettingsCmd ||

        $cmd == $StartSniffingCmd ||
        $cmd == $StopSniffingCmd ||
        $cmd == $SyncStatusCmd } {
        set statusCnt $NumberDatasources

    } else {

        set statusCnt 1

    }

    puts [format "<<< Waiting for status ( timeout = %d msecs )..." $timeout ]
    set retVal 0
    for { set i 0 } { $i < $statusCnt } { incr i } {


        set retVal [WaitForCmdStatus $cmd "" $timeout]
        if { $retVal == 1 } {

            puts " $Status"

        } else {

            puts [format "\n!!! WARNING : Timed out waiting for status from : %s\n" $cmd]

        }

    }
    puts "\n"

}

proc WaitForCmdStatus { cmd {status ""} {timeout 0} } {


    global Status

    ###puts "... Waiting for Cmd=$cmd Status=$status"

    set retVal 0
    set done 0
    while { !$done } {


        # Wait for a Status reply
        set retVal [WaitForStatus $timeout]

        # Check for timeout
        if { $retVal != 1 } {
        set done 1
        }

        # Convert strings to lower case for comparing below
```

```
        set cmd [string tolower $cmd]
        set status [string tolower $status]
        set nextStatus [string tolower $Status]

        if { [string first $cmd $nextStatus] != -1 } {

            if { [string length $status] > 0} {

                if { [string first $status $nextStatus] != -1 } {

                    set done 1

                }

            } else {

                set done 1

            }

        }


        # If we got a status but it wasn't what we expected then pass it to the Unexpected Status
            Handler
        if { !$done } {

            UnexpectedStatusHandler $Status

        }

    }
    return $retVal

}


proc WaitForStatus { {timeout 0} } {


    global Status
    global CmdStatusArray
    global CmdArrayTail

    # if no timeout specified use a default of 1 hour
    if { $timeout == 0 } {

        set timeout 3600000

    }
    set retVal 0
    set Status ""
    set currTime [clock clicks -milliseconds]
    set expTime $currTime
    incr expTime $timeout

    ###puts [ format "### enter time : %d" $currTime ]

    set done 0
    while { !$done } {
```

```
        set nextStatus [ array get CmdStatusArray $CmdArrayTail ]

        if { [string length $nextStatus] != 0 } {

            array unset CmdStatusArray $CmdArrayTail
            incr CmdArrayTail

            # example of what nextStatus might look like
            # 1 {Start FTS;SUCCEEDED;Timestamp=8/14/2012 2:15:05 PM}
            #
            # We need to trim the extra stuff from the entry
            #
            set index [string first " \{" $nextStatus]
            if { $index != -1 } {

                incr index 2
                set length [ expr [string length $nextStatus] - 2 ]
                set Status [string range $nextStatus $index $length]

            }

        }

        # Check to see if we even got a Status
        if { [string length $Status] > 0 } {

            set retVal 1
            set done 1

        }

        if { $timeout != 0 } {

            if { [clock clicks -milliseconds] >= $expTime } {

                set retVal 0
                set done 1
                ###puts "\n### WARNING - Timed out waiting for status ...\n"

            }

        }

        # Process any pending events and idle callbacks
        update

    }

    ###puts [ format "### exit time : %d" [clock clicks -milliseconds] ]

    return $retVal

}

namespace export WaitForSyncState
proc WaitForSyncState { conn state {timeout 120000} } {


    set retVal 0

    if { ![string is integer $state] } {
```

```
        set state [SyncStateColorLookup $state]

        if { $state == 0 } {

            puts "!!! ERROR : Invalid Sync State Color"
            return retVal

        }

    }
    set currTime [clock clicks -milliseconds]
    set expTime $currTime
    incr expTime $timeout

    ###puts [ format "### enter time : %d" $currTime ]

    set color [SyncStateColorLookup $state]
    puts [format "..... Waiting for connection %d sync state to go %s(%d) ( timeout = %d msecs )..."
          $conn $color $state $timeout ]

    set done 0
    while { !$done } {


        if { $state == [GetSyncStateValue $conn] } {

            set done 1
            set retVal 1

        }

        if { $timeout != 0 } {

            if { [clock clicks -milliseconds] >= $expTime } {

                set done 1
                set retVal 0
                puts "\n!!!!!! WARNING : Timed out waiting for sync state ...\n"

            }

        }

        # Process any pending events and idle callbacks
        update

    }
    return retVal

}

##################################################################
# Command Wrappers
##################################################################
proc StartFTS { params {timeout 60000} } {


    global Cmd
    global StartFTSCmd
    global Params
    global Status
    global NumberDatasources
```

```
    set Cmd $StartFTSCmd
    set Params $params
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

    # Extract datasource count from Cmd Status

    set Status [string tolower $Status]
    set index1 [string first "count=" $Status]
    set index2 [string first ";" $Status $index1]
    if { $index1 != -1 && $index2 != -1 } {


        set index1 [expr $index1 + 6]
        set index2 [expr $index2 - 1]

        set NumberDatasources [string range $Status $index1 $index2]

    }

    puts [format " Number of Datasources: %d\n" $NumberDatasources]

}
proc StopFTS { {timeout 60000} } {

    global Cmd
    global StopFTSCmd
    global Params

    set Cmd $StopFTSCmd
    set Params ""
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc ConfigSettings { params {timeout 30000} } {

    global Cmd
    global ConfigSettingsCmd
    global Params

    set Cmd $ConfigSettingsCmd
    set Params $params
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc StartCapture { {timeout 30000} } {

    global Cmd
    global StartCaptureCmd
    global Params

    set Cmd $StartCaptureCmd
    set Params ""
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}
```

```
proc SaveCapture { params {timeout 120000} } {

    global Cmd
    global SaveCaptureCmd
    global Params

    set Cmd $SaveCaptureCmd
    set Params $params
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc StopCapture { {timeout 120000} } {

    global Cmd
    global StopCaptureCmd
    global Params

    set Cmd $StopCaptureCmd
    set Params ""
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc Clear { {timeout 60000} } {

    global Cmd
    global ClearCmd
    global Params

    set Cmd $ClearCmd
    set Params ""
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc StartSniffing { {timeout 30000} } {

    global Cmd
    global StartSniffingCmd
    global Params

    set Cmd $StartSniffingCmd
    set Params ""
    SendCmd $Cmd $Params
    return [GetCmdStatus $Cmd $timeout]

}

proc StopSniffing { {timeout 30000} } {

    global Cmd
    global StopSniffingCmd
    global Params

    set Cmd $StopSniffingCmd
    set Params ""
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout
```

```
}

proc OpenCaptureFile { params {timeout 120000} } {

    global Cmd
    global OpenCaptureFileCmd
    global Params

    set Cmd $OpenCaptureFileCmd
    set Params $params
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}


    proc CloseCaptureFile { {timeout 60000} } {
    global Cmd
    global CloseCaptureFileCmd
    global Params

    set Cmd $CloseCaptureFileCmd
    set Params ""
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc GoLive { {timeout 120000} } {

    global Cmd
    global GoLiveCmd
    global Params

    set Cmd $GoLiveCmd
    set Params ""
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc ExitLiveMode { {timeout 120000} } {

    global Cmd
    global ExitLiveModeCmd
    global Params

    set Cmd $ExitLiveModeCmd
    set Params ""
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc SyncStatus { params {timeout 30000} } {

    global Cmd
    global SyncStatusCmd
    global Params

    set Cmd $SyncStatusCmd
    set Params $params
```

```
    SendCmd $Cmd $Params
    # Note - do not wait for a status

}

proc ReSyncStatusCmdNow { {timeout 30000} } {

    global Cmd
    global ReSyncStatusCmdNowCmd
    global Params

    set Cmd $ReSyncStatusCmdNowCmd
    set Params ""
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc HTMLExport { params {timeout 1800000} } {

    global Cmd
    global HTMLExportCmd
    global Params

    set Cmd $HTMLExportCmd
    set Params $params
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}

proc Export { params {timeout 1800000} } {

    global Cmd
    global ExportCmd
    global Params

    set Cmd $ExportCmd
    set Params $params
    SendCmd $Cmd $Params
    GetCmdStatus $Cmd $timeout

}


################################################################
# FTE_BASE namespace vars
################################################################

# Some default options
set Connections(Host) 0.0.0.0;
set Connections(Port) 22901;
set Connections(Buffering) full;
set Connections(BufSize) 80000;
set Connections(DataLength) 1000;
set Buffer [string repeat "0" 1000];
set Cid 0
set NumberDatasources 1

set Cmd "";
set Params "";
```

```
set Status "";

set CmdArrayHead 0
set CmdArrayTail 0
array set CmdStatusArray [list 0 "initialize"]
array unset CmdStatusArray 0
array set SyncStatusArray [list 1 0]

# The various cmd strings
set StartFTSCmd "Start FTS";
set StopFTSCmd "Stop FTS";
set ConfigSettingsCmd "Config Settings";
set StartCaptureCmd "Start Capture";
set SaveCaptureCmd "Save Capture";
set StopCaptureCmd "Stop Capture";
set ClearCmd "Clear";
set StartSniffingCmd "Start Sniffing";
set StopSniffingCmd "Stop Sniffing";
set OpenCaptureFileCmd "Open Capture File";
set CloseCaptureFileCmd "Close Capture File";
set GoLiveCmd "Go Live";
set ExitLiveModeCmd "Exit Live Mode";
set SyncStatusCmd "Sync Status";
set ReSyncStatusCmdNowCmd "ReSync Status Now";
set HTMLExportCmd "HTML Export";
set ExportCmd "Export";

#################################################################
# Start of sample script
#################################################################

#
# FTEBaseInit is always the first thing that is called passing the appropriate IP Address
#
FTEBaseInit 192.168.0.90


#
# Typically the next thing to do is to start CPAS ( note FTS is a legacy reference )
#

# We must specify the version of Frontline software to be started. You would replace "10" with
# your appropriate version of Frontline sofware. Additionally if you installed Frontline software in a
      different location
# you would have to modify the path accordingly.
#
set CPASVersion "C:\\Program Files\\Frontline Test System II\\Frontline 10 \\Executables\\Core"

# Note: following are what the Profile setting would be for various technology platforms
# and combinations of technology platforms.
#
# - For BPA low energy:
# set Profile "FTSLE"
#
# - For 802.11 only:
# set Profile "80211"
#
# - For BPA600 only:
set Profile "BPA600"
#
```

```
# - For BPA600 Coexisttence
# set Profile "BPA600_Coex"
#
# -For two 802.11 Coexistence
#set Profile "TwoWiFi"
#
#-For SD/SDIO
#set Profile "SDIO"
#
#-For Sodera and 802.11 Coexistence
#set Profile "Sodera_80211_COEX"
#
#-For Sodera
#set Profile "Sodera"
StartFTS [format "%s;%s" $CPASVersion $Profile]

puts "Waiting 60 seconds for datasource initializations..."
after 60000
#
# Set Config settings for BPA 600
#
ConfigSettings [format "IOParameters;BPA600;Master=0x00025b01cb8b;Slave=0x00025b01cbe1"]

#
# Select first device in list for 802.11 datasource ( i.e. ahid=0 )
#
ConfigSettings [format "HWParameters;80211;ahid=0;DevIndex=0"]

#
# Set I/O parameters for 802.11 datasource
#
ConfigSettings [format "IOParameters;80211;ahid=0;Channel=2"]

#
# If you are using a BPA sniffer, you want to turn on Sync Status. Once Sync Status is turned on
# anytime the status of a connection changes, an unsolicited Sync Status Reply is sent back
# through the FTSAutoServer application.
#
SyncStatus "ON"

#
# If you are using BPA sniffer, you would want to Start Sniffing now.
#
StartSniffing

#
# If you are using BPA sniffer, you would probably want to wait until the Sync Status for
# the connection you are interested in goes "Green". This is an example of waiting for an
# unsolicited Snyc Status update. You can send an optional 3rd parameter which is a timeout
# in msecs. Note that the default timeout is 2 minutes.
#
# Also note if you prefer to use the Sync Status values instead of the colors you can pass
# the value as a parameter in place of the color ( e.g. Green = 4 )
#
# Once the Sync Status goes "Green", you would want to have code in your script that causes
# your DUTs to do pairing.
#
WaitForSyncState 1 "Green"

#
```

```
# Note: This is here just to allow us to capture for a short period of time.
# There is really no other significance to it being here.
#
puts "Capture for 10 secs ..."
after 10000

#
# This will stop the BPA sniffer, and Sync Status will subsequently go "Red"
#
StopSniffing

#
# Wait for Sync Status to go "Red" before proceding in script.
#
WaitForSyncState 1 "Red"

#
# After Sync State has gone red, stop capture
#
StopCapture

#
# Just before exiting your script, turn Off Sync Status if you turned it On above.
#
SyncStatus OFF

#
# One of the last commands you will probably be doing is to Stop CPAS ( again FTS is a
# legacy reference )
#
StopFTS

#
# FTEBaseCleanup is always the last thing you need to do.
#
FTEBaseCleanup

exit
```