

# **Virtual Sniffing**

# **Live Import**

## **Reference & User Manual**



# Contents

---

<b>Chapter 1 Introduction to Live Import</b> .....	<b>1</b>
1.1 Virtual Sniffing Example .....	1
1.2 Adding Live Import to an Application .....	2
1.2.1 Load Live Import API .....	2
1.2.2 Initialize Live Import .....	2
1.2.3 Send Data to ComProbe Software .....	2
1.2.4 Close Live Import .....	3
1.2.5 Clean Up Live Import .....	3
1.2.6 Free LiveImportAPI.dll .....	3
1.3 Reporting Data Statistical Information .....	3
1.4 Live Import Samples .....	3
<b>Chapter 2 Installing Live Import Developer Kit</b> .....	<b>5</b>
<b>Chapter 3 Overview - Creating Live Import Data Source</b> .....	<b>7</b>
3.1 Modify Liveimport.ini .....	8
3.2 The Configuration String .....	9
3.2.1 General Rules .....	9
3.2.2 Required Configuration String Elements .....	9
3.2.2.1 Version and Section Header .....	9
3.2.2.2 Title Bar and Driver Info .....	10
3.2.2.3 Sides and Utilization .....	10
3.2.3 Optional Elements .....	10
3.2.3.1 Basic Unit of Data - Single Data Event .....	10
3.2.3.2 Data Related Flags and Errors .....	11
3.2.3.3 Non-Data Related Flags .....	11
3.2.3.4 Protocol Stack .....	12
3.2.3.5 Custom Statistics .....	12
3.2.3.6 Miscellaneous Driver Items .....	14
<b>Chapter 4 Function Quick Reference</b> .....	<b>17</b>
<b>Chapter 5 Technical Support</b> .....	<b>23</b>

## Figures



---

Figure 2.1 - Frontline ComProbe Development Tools Directory .....	5
Figure 2.2 - Select Location to Install Live Import Developers Kit .....	6
Figure 2.3 - Successful Installation Notice .....	6
Figure 3.1 - Liveimport Component Relationships to ComProbe Software .....	8

## Tables

---

Table 1.1 - Live Import Development Kit Data Source Samples .....	3
Table 3.1 - Table Numbers .....	12
Table 4.1 - API Handle Function .....	17
Table 4.2 - LoadFunction .....	17
Table 4.3 - Initialize Function .....	18
Table 4.4 - IsAppReady Function .....	18
Table 4.5 - SendFrame Function .....	18
Table 4.6 - SendFrameWithComment Function .....	19
Table 4.7 - SendFrameWithCommentFromDatasource Function .....	19
Table 4.8 - SendEvent Function .....	20
Table 4.9 - UpdateStat Function .....	21
Table 4.10 - Release Function .....	21
Table 4.11 - Cleanup Function .....	21
Table 4.12 - Unload Function .....	21





## Chapter 1 Introduction to Live Import

The Live Import capability of the Frontline ComProbe software provides an open interface that allows any application to feed data into the analyzer so that it can be analyzed, processed, and displayed in the software Frame Display and Event Display. When used in conjunction with DecoderScript, the protocol decoding environment, Live Import can be applied to just about any data communications analysis need.

Common data communications and protocol analyzers acquire their data from a specific piece of hardware. Live Import is hardware independent—if a Windows application program, or device driver, can be written to acquire data from a particular device, then that data can also be sent to ComProbe software via Live Import.

The key to Live Import is having access to the data stream. Many embedded systems have “logging ports” that will send raw data and/or other diagnostic information to a “logging device”. The logging device may be connected by serial-data lines, Ethernet, whatever. A Windows application program could be written to act as the receiver for this logging information; the application could then forward the data into ComProbe software via Live Import.

In some cases there is no need for “sniffing” hardware at all. In a data communications system, various pieces of software will be “carrying” messages to and from applications—independent of any actual hardware. Live Import allows such messaging to be “tapped” and captured at the software level, eliminating the need to tap and capture the data at the actual physical transmission medium.

When used in this way, Live Import is referred to as “Virtual Sniffing.” Virtual Sniffing can be added to just about any Windows application or device driver. Once added, the full power of the ComProbe protocol decoding environment is available to assist in diagnostics, troubleshooting, or data analysis.

### 1.1 Virtual Sniffing Example

Microsoft Windows provides a number of different ways to tap into the flow of messages between applications, or between applications and external devices. For example, at the device driver level, “filter drivers” can be inserted between an application and the real driver for a certain device. The filter driver passes everything that it gets from an application to a device; and everything that it gets from a device to an application. While the driver has the data “in its hands”, it can use Live Import to pass the data into ComProbe software for analysis.

Virtual Sniffing is also applied to areas beyond the realm of data communications. Many communications protocols are nothing more than starting at the first byte of a data structure and transmitting each successive byte until the end of the structure. Because Live Import is not tied to a specific piece of hardware, an application program, such as a payroll system, could have a field-troubleshooting mode where it writes the data structures it

uses into a disk file. Virtual Sniffing could then be used to read the disk file and forward the data into ComProbe software, where the “protocol” decoding system could break down the fields of the data and display them for detailed analysis.

## 1.2 Adding Live Import to an Application

At it's simplest, Live Import can be added to an existing application by adding five function calls to the Live Import API and one function call to the Windows API.

- Live Import API
  - Load
  - Initialize
  - Send
  - Close
  - Clean
- Windows API
  - Free

### 1.2.1 Load Live Import API

Before the Live Import API functions can be called, the API dll and functions need to be loaded.

```
if(!LoadLiveImportAPIFunctions())  
    return (false);
```

Upon success, a global HMODULE object, g\_hLiveImportAPI, will be initialized.

### 1.2.2 Initialize Live Import

Before data can be sent to the ComProbe software, the client side of Live Import needs to be initialized

```
g_pInitializeLiveImport(  
    szMemoryName,  
    szConfiguration,  
    &boolSuccess)
```

### 1.2.3 Send Data to ComProbe Software

Live Import provides a number of function calls for sending information into ComProbe software. The most commonly used one is g\_pSendFrame(), which is used to send a self contained block of data such as a message in a communications protocol.

```
g_pSendFrame(  
    dataLength, // Original size of the frame  
    dataLength, // "Sliced" size of the frame  
    abytfFrameData,  
    flags,  
    side,  
    timestamp);
```



## 1.2.4 Close Live Import

When done, the client side of Live Import disconnects allowing shared resources to be freed and other cleanup to occur.

```
dwStatus = g_pReleaseLiveImport();
```

## 1.2.5 Clean Up Live Import

After the API is released, the function pointers need to be reset to NULL.

```
NullLiveImportFunctionPointers();
```

## 1.2.6 Free LiveImportAPI.dll

Prior to exit from the data source, the global handle to LiveImportAPI.dll should be freed.

```
FreeLibrary(g_hLiveImportAPI);
```

## 1.3 Reporting Data Statistical Information

Another commonly used Live Import API feature allows the application program to generate customized statistical information. The application defines the additional statistical tables that it wants to use, and then sends updated values to them as needed. The tables appear automatically on the ComProbe software Statistics View.

## 1.4 Live Import Samples

There are five Live Import Samples provided with the Live Import Development Kit. Your Data Source will likely be similar to one or a mixture of these samples. The easiest way to begin is to start with one of these samples and make modifications to it.

Table 1.1 - Live Import Development Kit Data Source Samples

Name	Language	Frame or Byte
GUI Sample	C++ with MFC	Frame with comments
GUI Sample Byte Oriented	C++ with MFC	Byte
Wrapper Sample	C with C++ wrapper	Frame
Straight C Sample	C	Frame
Straight C Sample Byte Oriented	C	Byte







## Chapter 2 Installing Live Import Developer Kit

The Live Import Developer Kit is delivered with your ComProbe software that you installed on your computer. Installation of the developer kit is an automatic process.

Navigate to C://Program Files (x86)/Frontline Test System II/Frontline ComProbe Protocol Analysis System <your version>/Development Tools/Live Import Developer Kit.exe.

Name	Date modified	Type	Size
ApplicationNoteProgrammaticallyUpdat...	5/30/2013 3:09 PM	Adobe Acrobat D...	109 KB
ComProbe Automation Server Protocol.p...	5/14/2014 4:35 PM	Adobe Acrobat D...	446 KB
CPAS Decoders.exe	7/19/2014 4:45 AM	Application	3,243 KB
decoder_source_code_license.pdf	4/13/2012 3:39 PM	Adobe Acrobat D...	23 KB
DecoderScript Manual.pdf	7/14/2014 10:23 AM	Adobe Acrobat D...	2,048 KB
DecoderScript QSG.pdf	1/2/2014 2:07 PM	Adobe Acrobat D...	274 KB
decoderscript.uew	11/5/2012 5:07 PM	UEW File	6 KB
decoderscriptmethods.uew	11/5/2012 5:07 PM	UEW File	2 KB
DecoderScriptWizard.exe	7/19/2014 4:45 AM	Application	59 KB
FrameDecoder Add-On.exe	7/19/2014 4:45 AM	Application	1,422 KB
Live Import Developer Kit.exe	7/19/2014 4:45 AM	Application	2,849 KB
SampleClient.tcl	3/11/2013 4:48 PM	TCL File	20 KB
Test_1.01_44.1kHz_16Bit.wav	7/14/2014 12:53 PM	Wave Sound	517 KB
Test_1.01_44.1kHz_16Bit_2Ch.wav	7/14/2014 12:53 PM	Wave Sound	1,034 KB
Test_1.01_64kHz_16Bit.wav	7/14/2014 12:53 PM	Wave Sound	751 KB

Figure 2.1 - Frontline ComProbe Development Tools Directory

Double click on the Live Import Developer Kit.exe file and the installation will begin. If you have multiple installations of Frontline ComProbe Analysis System the following window will appear.

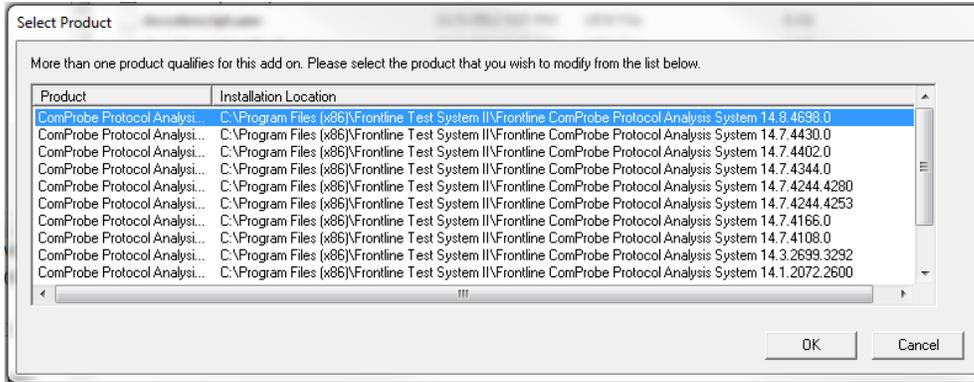


Figure 2.2 - Select Location to Install Live Import Developers Kit

Upon successful installation a notice will appear. Select **No** and the window will close. Installation is complete.

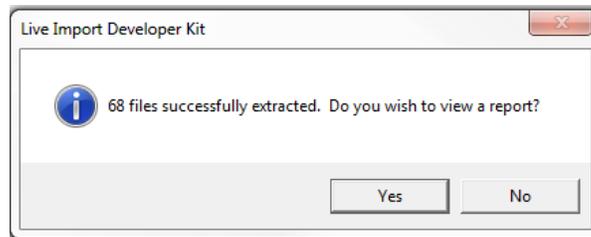


Figure 2.3 - Successful Installation Notice





## Chapter 3 Overview - Creating Live Import Data Source

As the writer of the Data Source you will have to perform the following steps. Additional details follow. The term "Data Source" refers to the software that you as the user provide to move data from your application or hardware into ComProbe software.

1. Modify Liveimport.ini Connection String and Configuration String. Liveimport.ini is located in the Live Import Developer Kit directory.
2. Modify your Data Source so that it knows where to find the Liveimport.ini so that it can read it. ComProbe software has to read this file also, but it knows where it is located. It is best not to move it. If for some reason Liveimport.ini has to reside in a special locations call Frontline Tech Support.
3. The Data Source creates a shared-memory connection to the ComProbe software by using LiveImportAPI.dll.
4. The Data Source reads Liveimport.ini and sends the Connection and Configuration data during initialization.
5. After initialization the Data Source checks connection and sends data either as frame or a byte at a time.
6. When the Data Source shuts down it releases the connection.



**Note:** Redistribution of an application that uses Live Import to transmit data to ComProbe Software requires LiveImportAPI.dll must be part of the distribution package. LiveImportAPI.dll must be in the same directory as the executable file

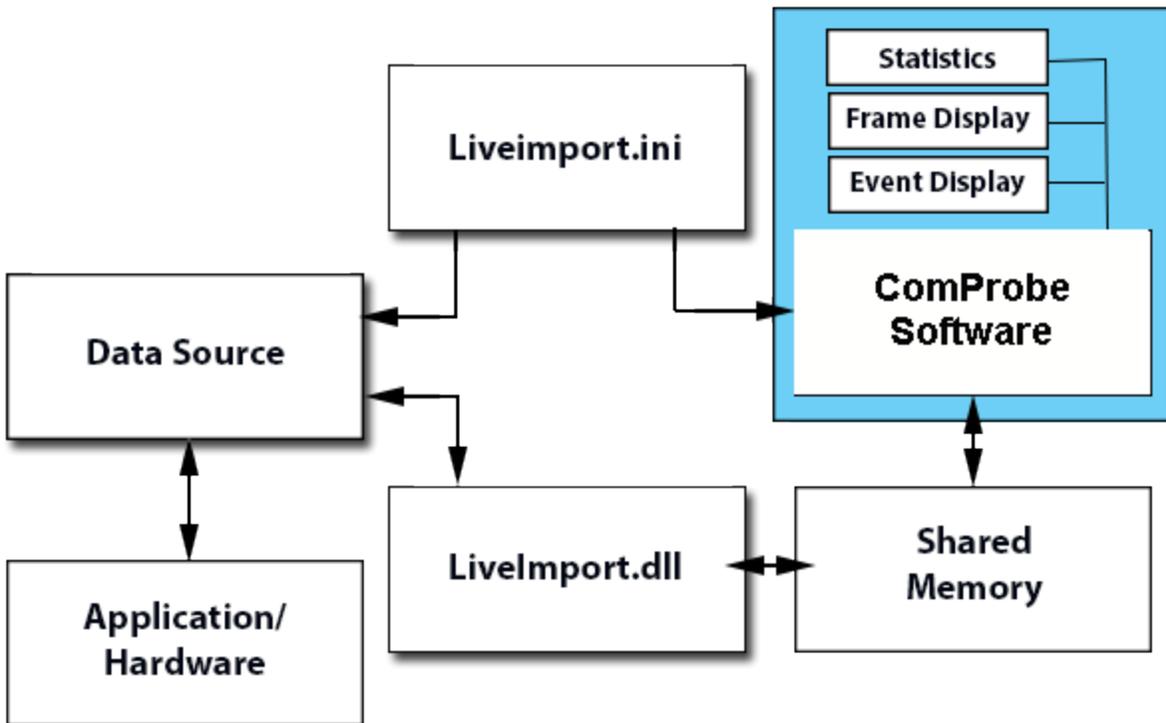


Figure 3.1 - Liveimport Component Relationships to ComProbe Software

### 3.1 Modify Liveimport.ini

Liveimport.ini is used to provide common connection and configuration information when using the Live Import capability. By default, ComProbe software will look for this file in the product installation directory C://Program Files (x86)\Frontline Test System II\Frontline ComProbe Protocol Analysis System <version no.>/.

There are two sections of information. The first section, [General], provides the connection string used by ComProbe software and a Live Import Data Source to find each other. This string is of your choosing but note that the connection string must end in a period ('.').

```
[General]
ConnectionString=FTS4BT Live Import.
```

The second section, [Configuration], provides the configuration to be shared between ComProbe software and a Live Import Data Source. Details of this string appear in [The Configuration String on page 9](#).

```
[Configuration]
Version=6
WindowTitle=FTS4BT Virtual Sniffer
DriverInfo=Bluetooth Virtual Sniffer
Sides=Host,1000000;Controller,1000000
SdeName=Octets
StackAuto=true
Stack=0x7f008034
Drf=Command;ACL;SCO;Event
```



The Data Source should read the data from this file and convert the configuration string into the form you see in [Example on page 9](#). All of the samples in the Live Import Developers Kit do this and you may simply copy and apply the code.

## 3.2 The Configuration String

The configuration string is a long string of text containing lines of descriptive information that specifies the communications environment for ComProbe software. The format is basically the Windows .INI file format.

### 3.2.1 General Rules

- The string is in single-byte ASCII format.
- Each item in the string must be terminated with a newline character (0x0A) ("\n" in C)
- The format of the string is similar to a section in an INI file. That is, the first line may be a section header starting and ending with square brackets (a section header is optional), and the rest of the lines have the format "item=value"
- All numbers can be expressed either in decimal or in hex by using the prefix "0x"
- All strings that contain a comma or a semicolon must be enclosed in double quotes. Double quote characters are not legal values for strings.
- Semi-colons separate all of the items in a list, and if there is more than one field in an item, the fields are separated by commas.
- Unrecognizable items will be assumed to be future expansions, so will be ignored. Be sure that the items you use are spelled correctly. Upper/lower case is ignored to the left of the equal sign.

#### Example

```
"Version=6\n"  
"WindowTitle=FTS4BT Virtual Sniffer\n"  
"DriverInfo=Bluetooth Virtual Sniffer\n"  
"Sides=Host,1000000;Controller,1000000\n"  
"SdeName=Octets\n"  
"StackAuto=true\n"  
"Stack=0x7f008034\n"  
"Drf=Command;ACL;SCO;Event\n"
```

#### List of Elements

- If a list of items is required, they must be separated by a semicolon.
- If there is a compound item requested, the item must be separated by a comma.
- If a string contains a semicolon or a comma, it must be enclosed in quotes.

### 3.2.2 Required Configuration String Elements

The configuration string elements in this section are required or highly recommended.

#### 3.2.2.1 Version and Section Header

For backwards compatibility, a version number is part of the configuration string. The version allows ComProbe software to use older versions of the information, defaulting new information that is not present. If the current version is "6", and the header will look like,



Version=6

If the configuration string does not begin with the version, it will be rejected. This is the only reason a configuration string is rejected.

If the device will emulate a ComProbe or Async product, set the section header [General] to [ComProbe] or [Async] respectively. Otherwise a section header is not needed.

### 3.2.2.2 Title Bar and Driver Info

You can control two ComProbe software **Control** window elements. WindowTitle string is displayed at the top of the **Control** window title bar, and DriverInfo string is displayed in the **Control** window **Configuration** field.

```
WindowTitle=Sample Window Title  
Driverinfo=Sample Configuration String
```

If you skip these items, the **Control** window fields will be blank.

### 3.2.2.3 Sides and Utilization

Data may be received from more than one device at a time. This occurs, for example, in normal two-way serial communication where you have a DTE device (computer, terminal, etc.) and a DCE device (modem, Palm Pilot, digital camera, etc.). Each direction of communication is a "side" ComProbe software supports either one or two sides. These sides must be given names that the ComProbe software can display.

In addition, if you want ComProbe software to calculate and display utilization information, you need to tell it what the maximum possible data rate is, per side. There are two forms of the side statement:

```
Sides=First side name,115200;Second side name,115200  
Sides=First side name;Second side name
```

The first form sets each side to a maximum of 115200 bits per second. The second form indicates that ComProbe should not calculate and display utilization information.

If there is only one side, a logical name is not necessary. The statement must still be present, though, with an empty string for the name, if ComProbe software is to calculate throughput. For instance, a 10-megabit Ethernet connection may be expressed like this:

```
Sides="",10000000
```

If this statement is omitted, then ComProbe software will use one side, unnamed, with no throughput.

If more than two sides are declared, only the first two will be used.

## 3.2.3 Optional Elements

The Configuration string optional elements provide the ability to set data units, flags, protocol stacks, and to create statistics. These elements are not required and if omitted will be set to a default value.

### 3.2.3.1 Basic Unit of Data - Single Data Event

The basic unit of data -- bytes, octets, etc -- is referred to in ComProbe software as a Single Data Event (SDE). This can be given any name for display purposes:

```
SdeName=Octets
```

If this statement is omitted, ComProbe software will default to "Chars"



### 3.2.3.2 Data Related Flags and Errors

Each data event can have up to 32 flags associated with it. These are called Data Related Flags (DRFs). The flags can be used for just about anything and indicate additional information about the event. In serial communications, the flags are used to indicate the error status reported by the serial communications controller: Parity, Overrun, Framing, etc.

A handy use for DRFs is to communicate status or type information to ComProbe software without having to embed indicator bytes into the data stream. The sample applications for example send the type of the message to the software via DRFs. The DRFs are available to the protocol decoders; so, a decoder can be written to learn about the type of a message without your needing to insert a type indicator into the "wire data".

The names you give the DRFs are used in the **Event Display**. They will appear in the Errors section of the status lines at the bottom of the window for each byte that the flag is attached to. In order to see the flags, however, you must not only define them, but specify that they be tagged as Errors as well.

The DRFs are stored with each event as a low order justified bit map. That is, the first one defined is stored as 0x00000001, the second is stored as 0x00000002, the third, 0x00000004, etc.

For a framed data event, where an entire frame of "bytes" is delivered as a single unit, the entire frame gets one set of DRFs. In the example below, Overrun is stored as 0x00000001, Framing is stored as 0x00000002, and Parity as 0x00000004.

DRF=Overrun;Framing;Parity

The default is to have no DRFs.

In addition, ComProbe software needs to know whether the flags you have defined are considered errors. Errors are displayed in red in the Event Display and Frame Display and can be searched for. This item is a bit mask with a bit = "1" setting the DRF as an error-bit. In the example below, all three of the DRFs—Overrun, Framing, and Parity—we defined above are errors.

DrfErrorMask=0x07

The default mask is 0x0.

Event Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
286369	03	0b	e4	cd	00	00	9a	5f	00	00	82	07	10	R	0b	
286385	03	25	fc	cd	00	00	9a	3c	02	00	82	23	db	ab	c8	4d
286401	29	aa	ce	00	00	9a	5e	00	00	b3	07	11	R	0b	03	
286417	3d	ce	d0	00	00	d2	78	02	00	c3	0f	00	49	00	1b	ff

### 3.2.3.3 Non-Data Related Flags

In addition to the Data Related Flags, ComProbe software supports up to 32 Non Data Related Flags (NDRFs). These flags are configured similar to the DRFs, but they are not associated with a data event.

These flags appear in the status lines of the Event Display. There is a maximum size specified in pixels that each box can be. You will need to experiment with NDRF names to ensure that the entire name is displayed on the window. Also, if you specify too many NDRFs, they will go off the edge of the window, and as there is no scroll bar, you will never be able to see them. If you have a lot of NDRFs, try to give them names as short as possible. These names also appear in the **Breakout Box** and **Find Signals** windows in serial products.

In serial type communication circuits, these are used to store the states of the control signals. NDRFs are declared in the same way as DRFs. They are stored as a low order justified bit map. That is, the first one defined is stored as 0x00000001, the second is stored as 0x00000002, the third, 0x00000004, etc.

The first form sets each side to a maximum of 115200 bits per second. The second form indicates that ComProbe should not calculate and display utilization information.



If there is only one side, a logical name is not necessary. The statement must still be present, though, with an empty string for the name, if ComProbe software is to calculate throughput. For instance, a 10-megabit Ethernet connection may be expressed like this:

```
NDRF=RTS;CTS;DSR;CD;DTR;RI
```

The default is no NDRFs.

### 3.2.3.4 Protocol Stack

The Data Source can suggest a protocol stack to be used. Doing so makes life easier on the end user since they will not have to select a stack before starting data capture.

There are two components to setting the protocol stack: The predefined part of the stack, and whether ComProbe software should try to discover other layers on the fly (autotraversal).

If ComProbe software should try to autotraverse, then include the following:

```
StackAuto=true
```

If you are setting the entire stack in advance for each frame that is received, use this:

```
StackAuto=false
```

The default is "false".

The decoders are indicated by their unique Decoder ID numbers. These numbers are found at the top of each DecoderScript source file.

```
Stack=0x7f000408;0x7f000409
```

The above defines a stack of two decoders, where the first decoder listed is the lower layer on the stack. The default Stack setting is an empty string.

### 3.2.3.5 Custom Statistics

The Statistics window displays statistics grouped by general type. The groupings are called "tables" You can add items to existing Statistics tables, or create new tables.

Each table is assigned a number. The tables can be accessed with these values:

Table 3.1 - Table Numbers

Number	Table
0	Utilization
1	FramesPerSecond
2	ChactersPerSecond
3	Data
4	Errors
5	FrameSizes
6	BufferInfo
7	FirstExtraTable



The table numbers are used in determining the row order when updating the statistics you've created. See the section [Updating the Driver Statistics on page 14](#) for information.

You can set stylistic elements for each table you create and for the rows in the table. The styles are enumerated in a bit mask as defined in [Table Styles on page 13](#).

### 3.2.3.5.1 Table Styles

HAS\_SIDES (0x0001)

This table should have a column for each side. This has no effect if there is only one side.

HAS\_TOTAL (0x0002)

Only valid if HAS\_SIDES is also set and the number of sides is greater than one. This adds a total column to the left of the side columns.

ROWS\_HAVE\_PERCENT (0x0004)

If HAS\_TOTAL is also set, then the percents for each side appear next to the side's total.

### 3.2.3.5.2 Row Styles

SUPPRESS\_TOTALS (0x0002)

On tables that have totals, this row does not.

MAKE\_NON\_ZERO\_STANDOUT (0x0020)

This row will have non-zero items stand out. "Stand out" usually means appear in red.

ADD\_PERCENT (0x0040)

On tables without percents, this row has a percent.

### 3.2.3.5.3 Adding Items to a Standard Statistics Table

Extra rows can be appended to the bottom of a statistic table with the following line:

```
ExtraStatRows2=First New Row Title,32;Second Row Title
```

The "2" at the end of ExtraStatRows2 represents the table that we should add to, in this case the table "Characters Per Second", according to the enumeration of tables given above.

The "32" after the "First New Row Title" is a row style bit flag given in decimal. All applicable style bits are OR-ed together. The values for the style bit flags for rows are given in [Row Styles on page 13](#). In this case, the "32" refers to the MAKE\_NON\_ZERO\_STANDOUT item. The style bit flags can be given in either decimal or hex. If using hex, the above line would read:

```
ExtraStatRows2=First New Row Title,0x0020;Second Row Title
```



**Note:** In the example above that there is no style flag for the second row. This will assume a style flag of zero.

If you have no statistics to add to a table, do not include this line. There is a practical limit to how many rows may be added before the display becomes unmanageable.



### 3.2.3.5.4 Creating New Statistics Tables

The driver can also create brand new statistics tables similarly to the way it creates new rows. The first part of the definition is the title for the whole table, and the rest has the same format as the extra row definition.

```
ExtraStatTable0=Table Title,3;First Row,2;Second Row,2
```

The "0" represents the new table number. The extra tables must begin at 0 and be sequential or they will be ignored. Tables can be defined in any order, as they will be sorted by ComProbe software on loading, but the resulting list must be sequential.

The "3" after "Table Title" indicates the table style, and is composed of the table-style enumeration values OR-ed together. In this case, the table specified the HAS\_SIDES and HAS\_TOTAL styles. If no style is needed, the style bits can be omitted. As with row styles, the style bits can be represented in either decimal or hex.

The "2"s following the "First Row" and "Second Row" labels refer to the style for the row.

### 3.2.3.5.5 Updating the Driver Statistics

Once you have specified the new statistics to display, you need a way to update them. The extra statistics are given an enumeration beginning at zero. Remember that each table in the table list is given a number (see [Creating New Statistics Tables on page 14](#)) The first extra row of the lowest numbered standard table is first, followed by each of the other extra rows on that table, then all the extra rows on other standard tables, then all the rows on the driver created tables.

The values in the row are changed with

```
UpdateStat(row_number, value)
```

where row\_number is the enumeration described above.

For example, if you had added an extra row to the Data table (Table #3), two extra rows to the Errors table (Table #4), and defined a new table with 2 rows in it, the rows would be enumerated as follows:

0 = Data table extra row

1 = Errors table first extra row

2 = Errors table second extra row

3 = New Table first row

4 = New Table second row



**Note:** While the tables are numbered such that the first extra table is number 7, when you define a new table, you define it as table 0. This ensures that if we add additional standard tables, your configuration will not need to change.

### 3.2.3.6 Miscellaneous Driver Items

The following items describe various behaviors of the driver:

```
DeviceHasPowerIndicator=true
```

This indicates whether the driver will send power messages to ComProbe software. It causes the software to put the power indicator on the **Control** window. The default is false.

```
SyncHuntIsMeaningful=true
```



This command causes the Sync Hunt button to appear, and to cause an “Enter Sync Hunt” message to be sent to the driver when it is pushed. The default is false.

`DriverSupportsBpfFiltering=true`

This tells ComProbe software that the driver can accept Filter messages, and that it will filter out frames before they are received by the ComProbe software. The default is false.

`RateStringLabel=Baud`

This command sets the name of the rate string in the statistics display. The default is “Speed”.

`SmallestPossibleFrame=64`

This is the smallest frame that the driver will ever send ComProbe software in any circumstance. This affects the ranges on the frame sizes in the statistics display. The default is 1.

`LargestPossibleFrame=1518`

This is the largest frame that the driver will ever send ComProbe software in any circumstance. This affects the ranges on the frame sizes in the statistics display. The default is -1, which means “no limit”.







## Chapter 4 Function Quick Reference

There are a many functions defined in the Live Import interface. They are all defined in C://Program Files (x84) /Frontline Test System II/Frontline ComProbe Protocol Analysis System <version>/Live Import Developers Kit/h/LiveImportAPI.h. The following list describes the functions that are likely to be most useful. If you have questions on any of the functions that are not described, or if it is unclear whether or not Live Import can provide the functionality you need, please contact [Technical Support](#).

Table 4.1 - API Handle Function

API Handle		
HMODULE g_hLiveImportAPI		
<b>Return Value</b>	None	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	None	
<b>Remarks</b>	This handle is defined in LiveImportAPI.h, and is a global object accessible in any module that includes that file. It is the handle to the API.	

Table 4.2 - LoadFunction

Load		
bool LoadLiveImportAPIFunctions(void)		
<b>Return Value</b>	none.	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	none	

Table 4.2 - LoadFunction(continued)

<b>Remarks</b>	This function must be called before any Live Import API functions are called. It loads LiveImportAPI.dll and the API functions. This function initializes g_hLiveImportAPI, which is the handle to LiveImportAPI.dll. An alternate method of accessing the functions of the API would be to extract the LoadLibrary call in LoadLiveImportAPIFunctions that explicitly loads the dll, and only call GetProcAddress for the desired functions.
----------------	---

Table 4.3 - Initialize Function

Initialize		
HRESULT g_plnitializeLiveImport(const TCHAR* szMemoryName, const TCHAR* szConfiguration, bool* pboolSuccess);		
<b>Return Value</b>	A Standard HRESULT value.	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	szMemoryName	The connection string.
	szConfiguration	The configuration string. Please see <a href="#">The Configuration String on page 9</a> .
	pboolSuccess	Returns whether the initialization was a success
<b>Remarks</b>	This function must be called before any data is sent to ComProbe software. The data that is passed to ComProbe software through this function should have been read from Liveimport.ini.	

Table 4.4 - IsAppReady Function

IsAppReady		
HRESULT g_plsAppReady (bool* pboolIsAppReady);		
<b>Return Value</b>	A Standard HRESULT value.	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	pblsAppReady	Returns whether FTS is ready to accept data.
<b>Remarks</b>	This function should be called to see if ComProbe software is ready to accept data. If ComProbe software is not ready and data is sent, the data will be not be received.	

Table 4.5 - SendFrame Function

SendFrame	
HRESULT g_pSendFrame(int iOriginalLength, int iIncludedLength, const BYTE* pbytFrame, int iDrf, int iStream, __int64 i64Timestamp);	
<b>Return Value</b>	A Standard HRESULT value.



Table 4.5 - SendFrame Function(continued)

<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	iOriginalLength	The "real" length of a frame. Some frames may be truncated, so this may not be the same as included length.
	iIncludedLength	The size of the data passed in this call.
	pbytFrame	The actual bytes of the frame.
	iDrf	Any errors or other data related flag.
	iStream	Which side these data come from.
	i64TimeStamp	The timestamp that should be placed on this frame.
<b>Remarks</b>	Call this function to send a frame to ComProbe software.	

Table 4.6 - SendFrameWithComment Function

<b>SendFrameWithComment</b>		
HRESULT g_pSendFrameWithComment(int iOriginalLength, int iIncludedLength, const BYTE* pbytFrame, int iDrf, int iStream, __int64 i64Timestamp, const TCHAR* ptcComment, unsigned int uiCommentLength);		
<b>Return Value</b>	A Standard HRESULT value.	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	iOriginalLength	The "real" length of a frame. Some frames may be truncated, so this may not be the same as included length.
	iIncludedLength	The size of the data passed in this call.
	pbytFrame	The actual bytes of the frame.
	iDrf	Any errors or other data related flag.
	iStream	Which side these data come from.
	i64TimeStamp	The timestamp that should be placed on this frame.
	ptcComment	A pointer to an array of characters that holds the desired comment.
	uiCommentLength	The length of the comment passed in this call.
<b>Remarks</b>	Call this function to send a frame and an accompanying comment to ComProbe software.	

Table 4.7 - SendFrameWithCommentFromDatasource Function

<b>SendFrameWithCommentFromDatasource</b>
---



Table 4.7 - SendFrameWithCommentFromDatasource Function(continued)

HRESULT g_pSendFrameWithCommentFromDatasource(int iDatasourceId, int iOriginalLength, int iIncludedLength, const BYTE* pbytFrame, int iDrf, int iStream, __int64 i64Timestamp, const TCHAR* ptcComment, unsigned int uiCommentLength);		
<b>Return Value</b>	A Standard HRESULT value.	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	iDatasourceId	Index of datasource sending the frame; 0 if only one datasource.
	iOriginalLength	The "real" length of a frame. Some frames may be truncated, so this may not be the same as included length.
	iIncludedLength	The size of the data passed in this call.
	pbytFrame	The actual bytes of the frame.
	iDrf	Any errors or other data related flag.
	iStream	Which side these data come from.
	i64TimeStamp	The timestamp that should be placed on this frame.
	ptcComment	A pointer to an array of characters that holds the desired comment.
uiCommentLength	The length of the comment passed in this call.	
<b>Remarks</b>	Call this function to send a frame and an accompanying comment to ComProbe software.	

Table 4.8 - SendEvent Function

SendEvent		
HRESULT g_pSendEvent (BYTE bytData, int iDrf, int iStream, __int64 i64Timestamp);		
<b>Return Value</b>	A Standard HRESULT value.	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	bytData	The actual event (BYTE) to be sent to ComProbe software.
	iDrf	Any errors or other data related flag.
	iStream	Which side these data come from.
	i64TimeStamp	The timestamp that should be placed on this byte.
<b>Remarks</b>	Call this function to send a single byte event to ComProbe software.	



Table 4.9 - UpdateStat Function

UpdateStat		
HRESULT g_pUpdateStat (int iStream, int iStatNumber, __int64 i64IncrementAmount);		
<b>Return Value</b>	A Standard HRESULT value.	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	iStream	Which side these data come from.
	iStatNumber	A number that indicates which statistic to update.
	i64IncrementAmount	A number that indicates how much to increment the statistic. To decrement, the number should be negative.
<b>Remarks</b>	Call this function to increment or decrement a statistic on the statistics screen of ComProbe software.	

Table 4.10 - Release Function

Release		
HRESULT g_pReleaseLiveImport (void);		
<b>Return Value</b>	A Standard HRESULT value.	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	none	
<b>Remarks</b>	Call this function when you are closing the Datasource to release the connection	

Table 4.11 - Cleanup Function

Cleanup		
void NullLiveImportFunctionPointers(void);		
<b>Return Value</b>	none	
<b>Parameters</b>	<b>Parameter</b>	<b>Description</b>
	none	
<b>Remarks</b>	Call this function when you are closing the Datasource to release the Live Import API function pointers.	

Table 4.12 - Unload Function

Unload		
BOOL FreeLibrary(HMODULE hModule);		
<b>Return Value</b>	A standard BOOL value.	



Table 4.12 - Unload Function(continued)

Parameters	Parameter	Description
	A standard HMODULE value	should be g_hLiveImportAPI.
<b>Remarks</b>	This function must be called just prior to exiting the Datasource with g_hLiveImportAPI as the argument.	

---





## Chapter 5 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.

Web: <http://www.fte.com>, click Support

Email: [tech\\_support@fte.com](mailto:tech_support@fte.com)

If you need to talk to a technical support representative, support is available between 9am and 5pm, U.S. Eastern time, Monday through Friday. Technical support is not available on U.S. national holidays.

Phone: +1 (434) 984-4500

Fax: +1 (434) 984-4505



---

## Index

---

### D

data related flags 11

Data Source 7

### E

Event Display 11

### L

Live Import 1

API

    Clean 3

    Close 3

    Initialize 2

    Load 2

    Send 2

Liveimport.ini 7

    configuration string 9

LiveImportAPI 7

### N

non-data related flags 11

### P

percent 13

Protocol Stack 12

### R

Row 13

rows 13

### S

styles 13

### T

Table 12-13

    new 14

totals 13

---

### V

Virtual Sniffing 1



