FLEXSTAND

CIM Industrial Systems A/S

FlexStand Operator Interface
Reference Manual

Contents

1 Revision history

Description	Version	Date	Initials
First issue	1.0	2009-04-20	HMo
Description of how to create a new customized operator interface added to Getting Started section. Advanced example has been modified. The Getting started section has been corrected accordingly. Pictures updated to v 1.1	1.1	2009-07-15	HMo

2 Preface

2.1 General

The flexible TestStand interface makes it possible to significantly reduce the development time of operator interfaces. Only basic Labview and TestStand skills are required.

By using FlexStand you can create simple or advanced operator interfaces for TestStand using only basic Labview programming. A specially designed interface (API) hides all the complex TestStand properties and methods that usually make the task of creating operator interfaces hard. FlexStand integrates fully into the Labview development environment including a tools palette and examples.

FlexStand allows you to create dynamic operator interfaces that fit the tasks for the device to be tested. The operator interface can even be changed during the test, for instance when a barcode is scanned.
A number of tabs, that can change automatically or by user control, provide the operator with more or less information based on your immediate requirements.

As the FlexStand operator interface automatically resizes to the size of the Labview front panels, the programmer has full control over the overall layout. This helps the programmer to allocate more space for some information and less for other.
By using a plug-in structure all code can be shared between test stations, which make reuse a simple and natural task.

This manual describes how to use the SDK for the FlexStand Operator Interface for TestStand. The manual describes in details how to customize the FlexStand operator interface. This also includes a description of the frameworks and event handling used in the Labview plug-ins.

2.2 Abbreviations

Abbreviation	Description
OI	Operator Interface
SDK	Software Development Kit
TS	TestStand sequencer by National Instruments

2.3 References

Ref.	Name	Location
[1]	NI TestStand Reference Manual	www.ni.com
[2]	NI TestStand Help	Start menu

3 System requirements for FlexStand Operator Interface

The following system components must be installed on the system before FlexStand is installed.

Operating system	Windows XP
TestStand version	4.1.1
Labview version (Development system only required for development of operator interface components)	8.6 (8.6.1 recommended)

4 Overview

The FlexStand operator interface consists of three main parts: Top plug-in, Main plug-in and buttom plug-in. The main plug-ins can contain more plug-ins organized in tabs.
Each plug-in is a Labview VI where the front panel is shown as a plug-in and the code is executed concurrently. The plug-ins are shown in Figure 1.
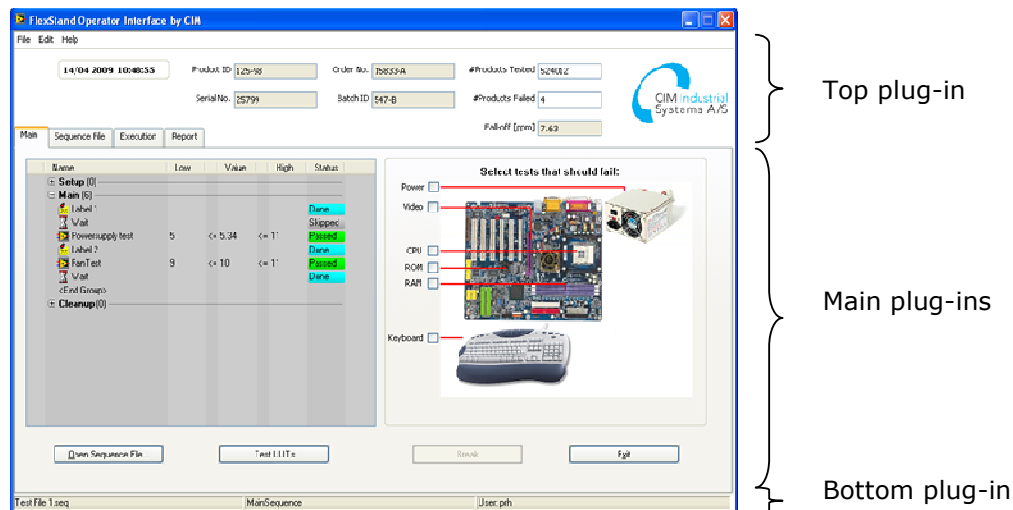

Figure 1: Plug-ins overview

FlexStand automatically adapts the size of the plug-in to the size of the front panel of the VI. By using different sizes of the front panels the FlexStand operator interface can be modified as shown in Figure 2.
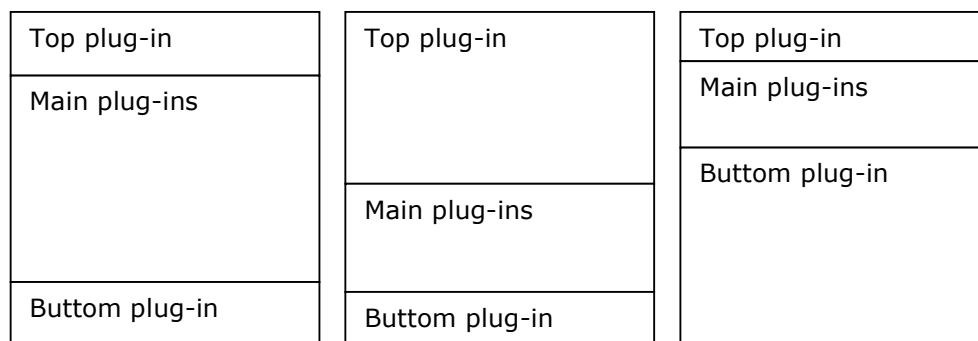

Figure 2: FlexStand operator interface layout

When FlexStand starts it first reads the Flexstand_OI.ini file. This file is located in the same directory as FlexStand_OI.exe.
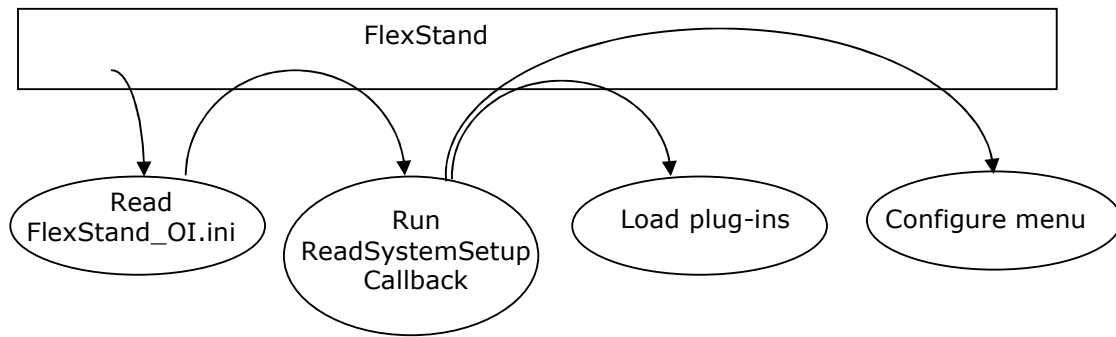
Figure 3: FlexStand start up sequence

As shown in Figure 3 FlexStand reads the paths to the callbacks from the .ini file. The most important callback is the ReadSystemSetup callback. This callback is a Labview VI that must contain a number of calls to the API. These calls will configure FlexStand to run as required by the current solution. The FlexStand examples can be used as templates for the ReadSystemSetup.

- CIM_TS_API - SetPluginPath (with a list of plug-in paths),
- CIM_TS_API - Set TAB Names (with a list of tab names for the main plug-ins),
- CIM_TS_API - Set Sys Err Config (with a path to the system error log file),
- CIM_TS_API - SetMenubarFileSpecification (with a path to the XML file that defines the menubar).

Note: In the FlexStand examples the information listed above is read from an .ini file. This is a solution proposal but not a requirement.

5 Getting started

This chapter contains a short description of how to install FlexStand and run the advanced example shipped with the Developer version.
It also contains a short introduction on how to create your own operator interface.

5.1 Installation

Before installing the FlexStand operator interface first check the system requirements listed in the Requirements table above. A Labview development system is required for modifying the FlexStand Operator Interface.

To install the FlexStand Operator Interface simply run the setup.exe. Read and accept the license agreement. Normally you should press next in all dialogs during the installation.

5.2 Running FlexStand

When the FlexStand Operator Interface has been installed it can be started by selecting FlexStand Operator Interface from the start menu. See Figure 4.
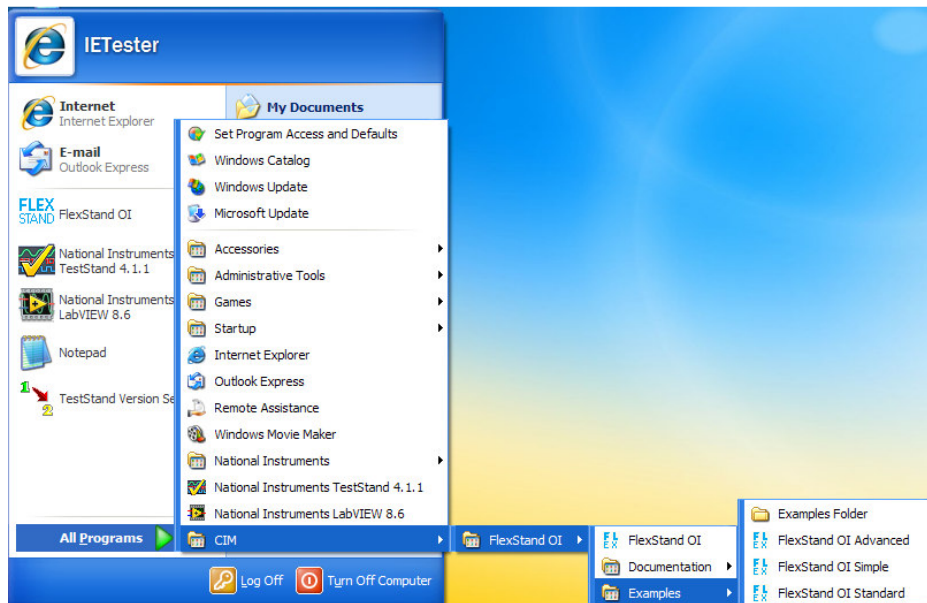

Figure 4: CIM Operator Interface in Start menu

The FlexStand Operator Interface will now start and show the default operator interface.

The first time FlexStand is started the license dialog is shown. Enter a valid license number in the License Key field. A valid license key can be obtained from www.flexstand.eu or by contacting CIM Industrial Systems.

Figure 5: FlexStand license dialog

TestStand will now show its login prompt. Log in as Administrator with an empty password.
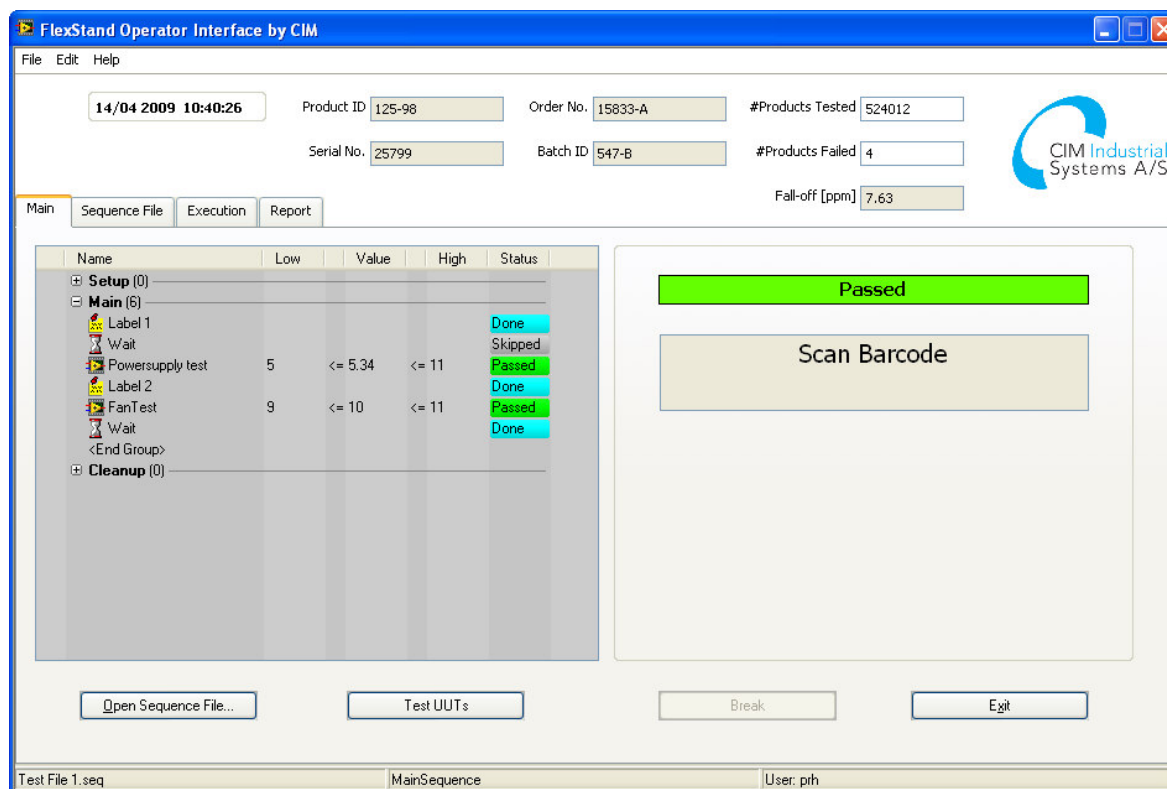

Figure 6: TestStand login prompt

Figure 7: CIM Operator Interface loaded.

From the main plugin a sequence file can be opened by pressing the "Open Sequence File…" button. Open the 'FlexStand OI Demo - Advanced - Computer Motherboard Test Sequence.seq' example sequence normally found in the folder:

C:\Documents and Settings\All Users\Documents\National Instruments\TestStand 4.1.1\UserInterfaces\FlexStand OI\Example\Advanced\DemoSequence

When the sequence has been loaded the 'Test UUTs' button will be visible as shown in Figure 8.
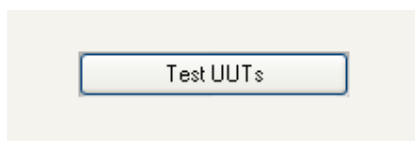

Figure 8: Test UUTs button

Press the Test UUTs button to start the sequence.
When the sequence finishes the operator interface changes to the report tab and displays the final test report as shown in Figure 9.
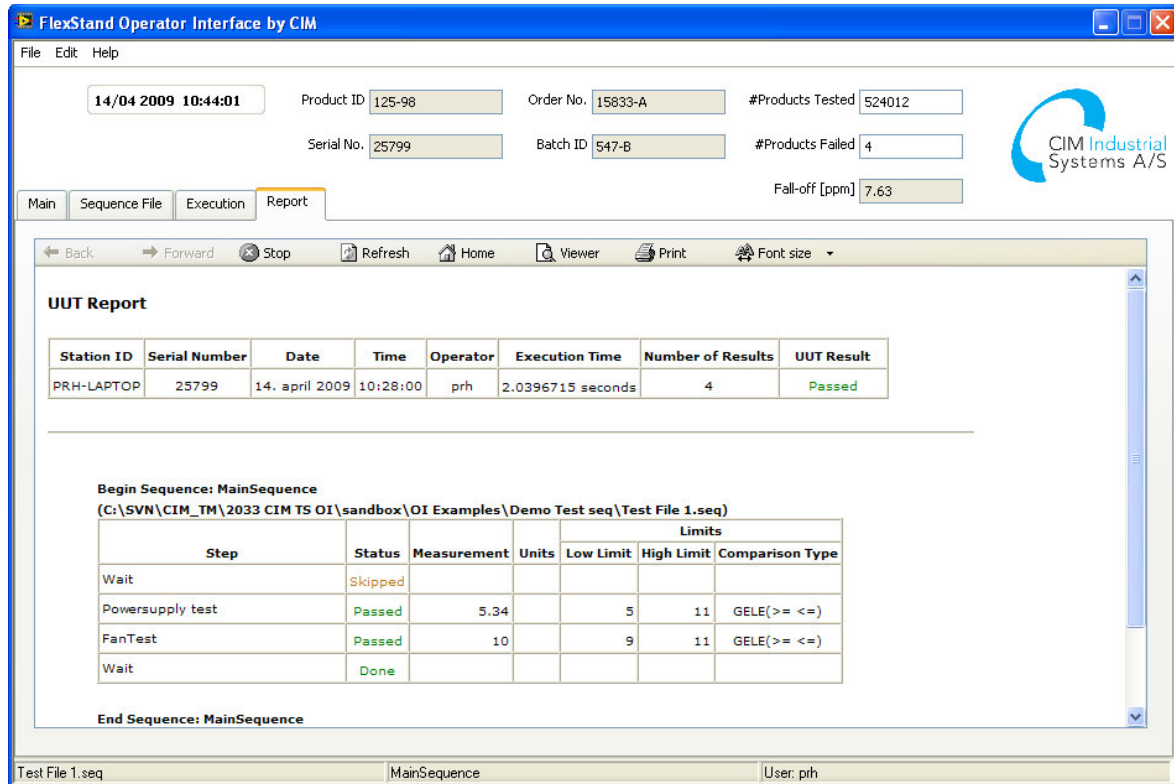
Figure 9: Report view.

5.3 Creating a custom operator interface

If you want a different operator interface than the examples shipped with FlexStand this can easily be made by using the examples as a starting point.

5.3.1 Copy example

Select the example that is closest to the requirements for your custom operator interface. Copy the folder Advanced, Standard or Simple with all subfolders to a new directory. When creating the first operator interface we recommend using the Simple example. Place the new directory at the same level in the tree structure of Documents and settings. Call the new directory a name describing the operator interface you are creating, for example "myOI". Refer to Figure 10.

Figure 10: New folder for custom operator interface.
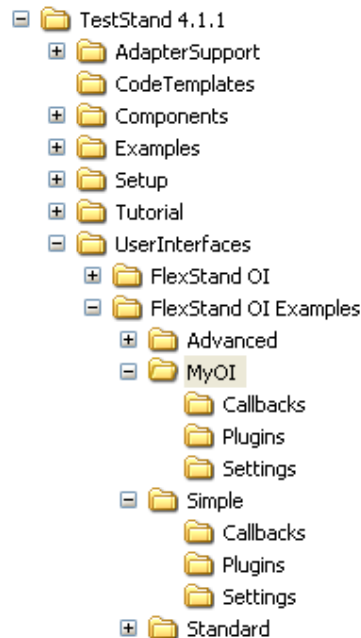
5.3.2 Modify FlexStand_OI.ini

To run the new operator interface FlexStand must know where to find the new files. FlexStand reads the folder with the callbacks from the FlexStand_OI.ini file. The FlexStand_OI.ini file is located in the 'FlexStand OI' folder. Open this file in notepad and correct the path to point to your new folder. The FlexStand_OI.ini could look like this after modification:

[Callbacks]
Init=""
Close=""
SystemError=""
ReadSystemSetup="C:\Documents and Settings\All Users\Documents\National Instruments\TestStand 4.1.1\UserInterfaces\FlexStand OI Examples\MyOI\Callbacks\CIM_TS_OI CALLBACK ReadSystemSetup.vi"
ReadCustomSetup=""

[SplashScreen]
Path="C:\Documents and Settings\All Users\Documents\National Instruments\TestStand 4.1.1\UserInterfaces\FlexStand OI Examples\MyOI\Plugins\Advanced_SplashScreenPlugin.vi"

5.3.3 Modify Plugins

The new folder 'Plugins' contains the plug-ins used in the example. These plug-ins can now be modified to the new requirements using Labview.
Save the plug-ins as VIs with a front panel of the desired layout and size. FlexStand will automatically adapt the size of the operator interface to the size of the plug-in VIs.

NOTE: There must be at least one top plug-in, one buttom plug-in and one main plug-in defined.

5.3.4 Modify plug-in list

The example code uses a SystemSetup.ini file for defining the names of the plug-ins. The method is reused here but can be changed by modifying the ReadSystemSetup callback. Modify the systemSetup.ini file to match the new plug-ins. The following figure shows an example of a SystemStup.ini file with a top and bottom plug-in and two main plug-ins.

```
[Plugins]
TopPlugin="Simple_TopPlugin.vi"
MainPlugin1="MyMainPlugin.vi"
MainPlugin2="MyExecutionViewPlugin.vi"
MainPlugin3=""
MainPlugin4=""
MainPlugin5=""
MainPlugin6=""
MainPlugin7=""
MainPlugin8=""
MainPlugin9=""
MainPlugin10=""
BottomPlugin="Simple_BottomPlugin.vi"


[TabNames]
Tab1="TS_Main"
Tab2="Execution view"
Tab3=""
Tab4=""
Tab5=""
Tab6=""
Tab7=""
Tab8=""
Tab9=""
Tab10=""

[SystemErrorLog]
Path="C:\SystemErrorLog.txt"

[MenuConfig]
Path="MenuConfiguration.xml"
```

5.3.5 Run new operator interface

Save all open files and run FlexStand by selecting Start>>All Programs>>CIM>>FlexStandOI>>FlexStand OI. Your new FlexStand operator interface will now start.

6 Templates

The FlexStand OI SDK comes with a number of fully functional operator interfaces. These operator interfaces can be used as is or serve as a template for building a customized operator interface.

The files for the example operator interfaces are stored in the TestStand directory which is normally found in the folder:

C:\Documents and Settings\All Users\Documents\National Instruments\TestStand 4.1.1\UserInterfaces\FlexStand OI Examples

The components in the FlexStand OI are described in details in the following chapters.

7 FlexStand Reference

The following sections contain detailed information about the components in FlexStand.

7.1 Labview palette

FlexStand OI SDK will install a FlexStand palette in Labview. This palette contains all the API functions that is needed for creating callbacks and plug-ins.
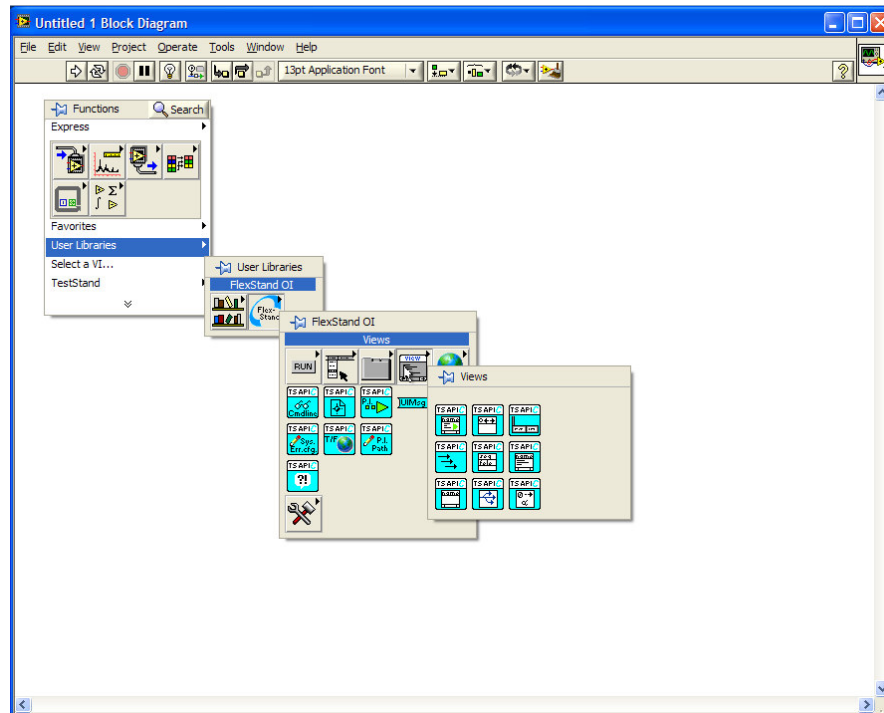

Figure 11: FlexStand palette in Labview

7.2 Callbacks

The Operator Interface can make calls to specific callback functions during execution. The following table shows a list of the callbacks and the sequence in which they are called.

Callback	Called at	Function
Init	Start of Operator Interface	Default empty. Can be used for logging of program start or to start programs that needs to run in parallel with the Operator Interface.
Close	Shut down of the Operator Interface	Default empty. Can for example be used for cleanup of data or to stop programs running in parallel to the Operator Interface.
System error	System error	Default empty. Can be used to log system error in a customer specific system or to modify the system error or even delete the system error for known cases. If the system error is cleared in the callback the system error handling built into the Operator Interface will not execute.
ReadSystemSetup	Start of Operator Interface	Sets the system parameters for: Plug-in Paths, Tab names, Menu specification file, System error handling.
ReadCustomSetup	Start of Operator Interface	Default empty. Can be used to read custom settings that must be available to the plug-ins for later use.

The operator interface is delivered with a set of template callbacks that can be used as a starting point for creating a customized callback.

The location of the callbacks must be listed in the FlexStand_OI ini-file. This file is located in the same directory as the FlexStand_OI.exe.

The ini-file contains a section called [Callbacks]. This section lists the locations of the callbacks.

```
[Callbacks]
Init=" "
Close=""
SystemError=" "
ReadSystemSetup=" C:\Documents and Settings\All Users\Documents\National Instruments\TestStand
4.1.1\UserInterfaces\FlexStand OI Examples\Advanced\Callbacks\CIM_TS_OI CALLBACK ReadSystemSetup.vi"
ReadCustomSetup=" "
```

Figure 12: Typical settings for callbacks

Note that callbacks with empty paths will not be called.

7.3 Plug-ins

7.3.1 Plug-in overview

The main user interface is organized in a number of plug-ins. The plug-ins are Labview VIs created to meet the needs for the actual operator interface. The operator interface automatically loads the plug-in at the appropriate locations and resizes the window to fit the size of the plug-in.
Figure 13 shows the layout of the plug-in. The operator interface supports one top panel plug-in, ten main panel plug-ins and one bottom panel plug-in.
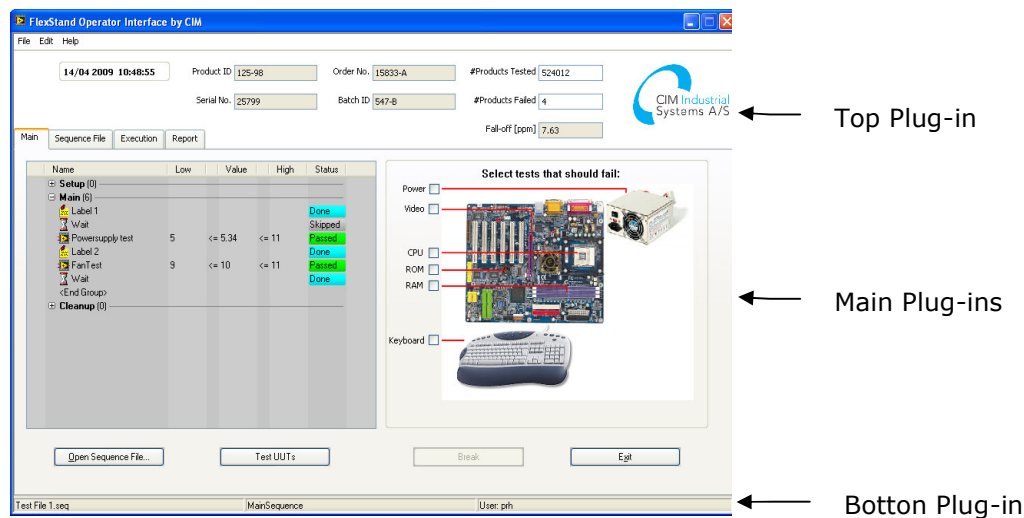

Figure 13: Plug-in layout.

The ten main panel plug-ins will be located on tabs, one plug-in one each tab.

7.3.2 Plug-in contents

The plug-ins are Labview VIs that must contain an event handler and other predefined calls to work properly. The event handler is used to receive events from the plug-in, from the FlexStand engine and from the Teststand engine. The plug-in must also send a message to the Operator Interface in order to tell the Operator Interface when the plug-in is running and ready to receive events.

7.3.2.1 Event handler

Figure 14 shows a typical basic layout of the code in a plug-in. The Get Event function returns a reference to the dynamic events generated by the Operator Interface. The reference must be connected to the dynamic event terminal on the event handler structure.
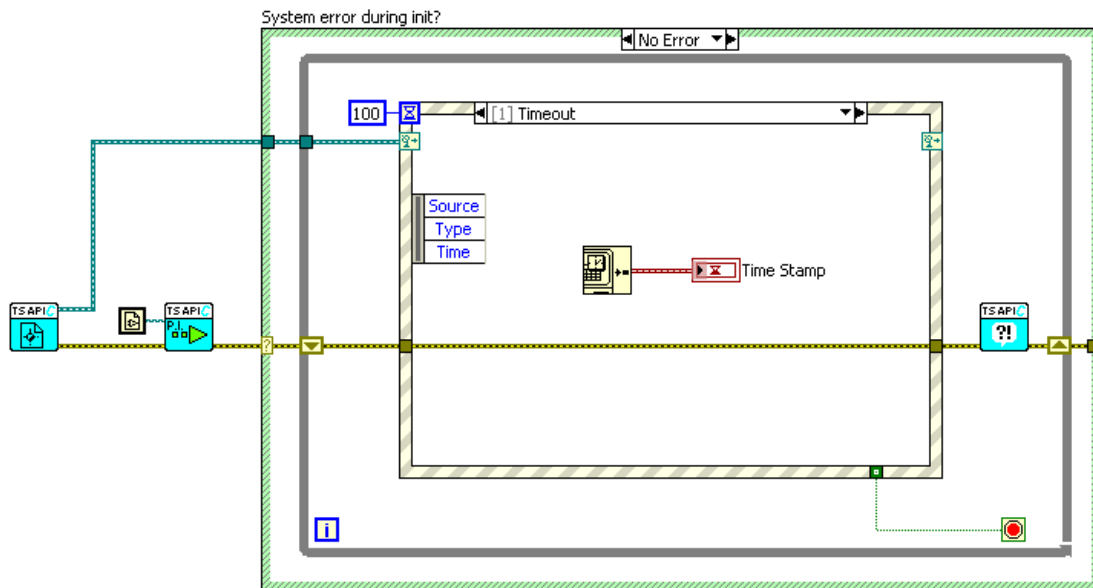
Figure 14: Plugin code.

The function 'Plug-in Started' sends a message to the Operator Interface that indicates that the plug-in has initialized and is ready to receive events.

In the above example the event handler contains a Timeout event case that updates a watch. As shown in Figure 15 the event handler also contains a Quit application event. This event is triggered when the Operator Interface is shutting down. The Quit application event must stop the execution of the plug-in. The Operator Interface waits for the plug-ins to stop before it finishes. The call to System error is a general API function to handle system errors.
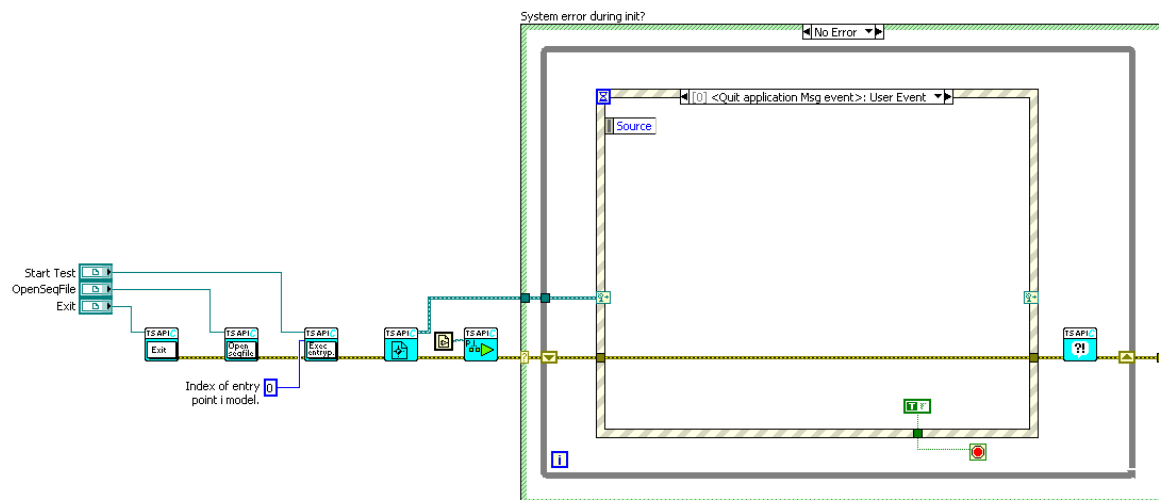

Figure 15: Quit application event.

7.3.2.2 TestStand buttons

A plug-in will typically contain TestStand elements like buttons and views. Using the FlexStand API these elements are connected to the TestStand kernel. This connection will ensure that the content of the button or view is continuously updated by the TestStand kernel.

7.3.2.3 Views
FlexStand OI supports a number of views that can be inserted in the plug-ins. These views can be configured using a dialog box. To open the dialog for configuration right click on the view in edit mode and select Advanced >> Design mode:
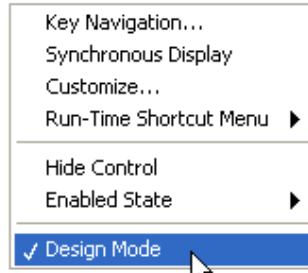
Right click again on the view and select for example SequenceView>>Properties.
In the dialog box shown the contents of the view can be fully configured.

Examples of configurations:
To create an Execution view that shows the status of the step add the status column in the SequenceView. If you want the status to be colored green and red for Pass and Fail insert the following expression in the dialog.
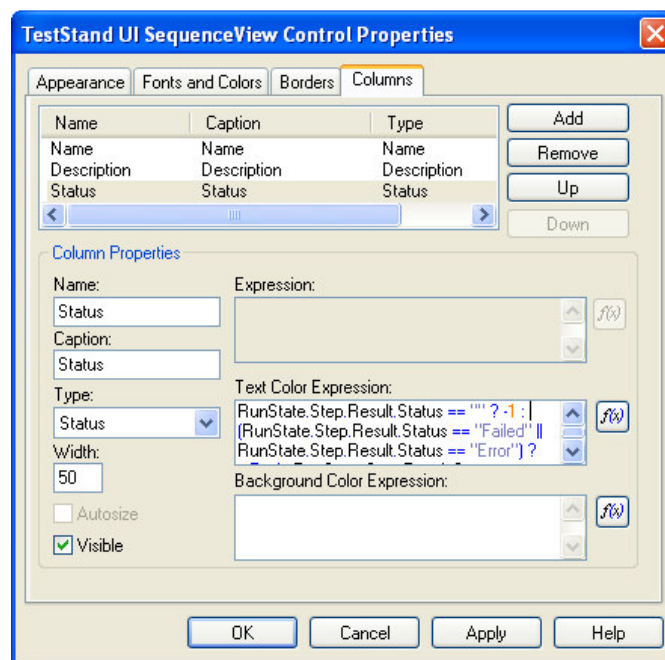
Figure 16: Expression for colored Pass/Fail status

The expression used is:

RunState.Step.Result.Status == "" ? -1 : (RunState.Step.Result.Status == "Failed" ||
RunState.Step.Result.Status == "Error") ? tsRed : RunState.Step.Result.Status == "Passed"
? tsGreen : RunState.Step.Result.Status == "Skipped" ? tsLightGray :
(RunState.Step.Result.Status == "Running"
|| RunState.Step.Result.Status == "Looping") ? tsMagenta : RunState.Step.Result.Status ==
"Terminated" ? tsBlue : tsCyan

The expression can be copied from the text above or from the example shipped with FlexStand OI.

7.3.2.4 Status bar
The standard TestStand status bar can be used on the front panels of the plug-ins. In the examples shipped with FlexStand the status bar is located in the bottom plugin. The content of the status bar is configured in edit mode by using the following procedure:

Add the TestStand UI StatusBar to the front panel of the vi.
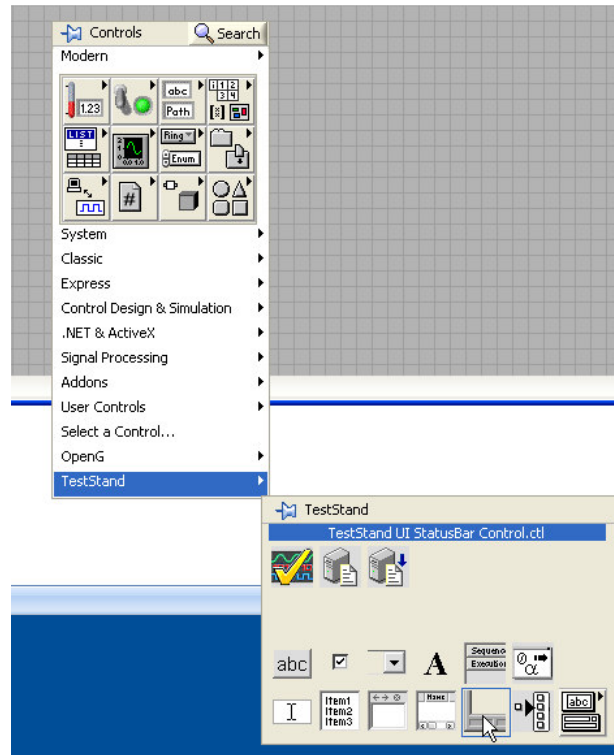
Right click the StatusBar and select Advanced. Make sure Design Mode is checked:
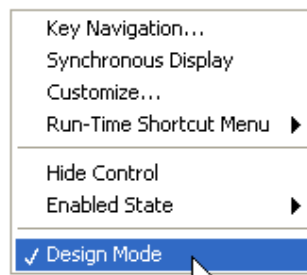
Right click on the StatusBar and open the editor by selecting Statusbar>>Properties. The properties editor now opens. Select the Panes tab as shown in the following figure.

NOTE: Remember to uncheck Design Mode before executing the plug-in otherwise the StatusBar will not be shown.

In the example above the StatusBar contains tree elements: Number of tests in the sequence, UUT status and the user name. The name used for the entry tells the FlexStand engine which elements to insert in the StatusBar. The names listed in following table can be used. The names must be spelled exactly as shown for the FlexStand engine to recognize the entry.
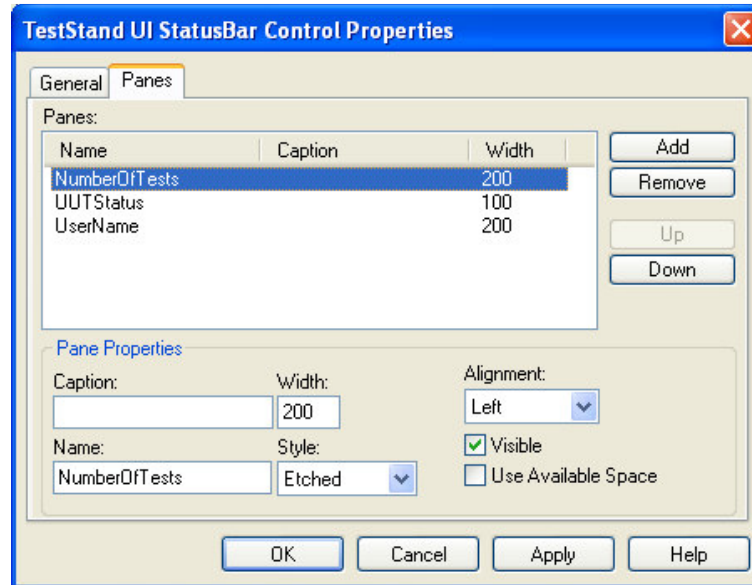
Some items will only be updated if a UIMessage Event is send from the model or test sequence. For those items where the UI Message is needed the UIMsg event code to use is shown in the following list.

- **BatchSerialNumber**: Displays the batch serial number for the batch that contains the current execution
- **BatchState**: Displays a description of the execution state of the batch that contains the current execution. For example, this caption might display Initializing, Testing, Waiting, or Completed [Passed].
- **BatchStatus**: Displays the batch result status for the batch that contains the current execution.
- **CurrentCallStackEntry**: Displays a name that identifies the current call stack item in the foreground thread in the current execution.
- **CurrentExecution**– Displays a name that identifies the current execution.
- **CurrentProcessModelFile**: Displays the path name of the process model file for the currently executing or currently selected sequence file.
- **CurrentSequence:** Displays the name of the currently executing or currently selected sequence.
- **CurrentSequenceComment:** Displays the comment for the currently executing or currently selected sequence.
- **CurrentSequenceFile:** Displays the path name of the currently executing or currently selected sequence file.
- **CurrentSequenceFileComment:** Displays the comment for the currently executing or currently selected sequence file.
- **CurrentStep:** Displays the name of the current step in the current execution.
- **CurrentStepComment:** Displays the comment for the current step in the current execution.
- **CurrentStepGroup:** Displays the name of the currently executing or currently selected step group.

- **CurrentStepIndex_ZeroBased:** Displays the zero-based index of the current step in the current execution.
- **CurrentTestIndex_OneBased:** Displays the one-based index of the current step in the current execution.
- **CurrentThread:** Displays a name that identifies the foreground thread in the current execution.
- **Date:** Displays the current date.
- **ModelState:** Displays a description of the execution state of the current execution. For example, this caption might display Initializing, Testing, Waiting, or Completed [Passed].
- **NotASource:** Guaranteed to never be a valid caption source specifier.
- **NumberOfSteps:** Displays the number of steps in the currently executing or currently selected step group.
- **NumberOfTests:** Displays the number of steps in the currently executing or currently selected step group. The description uses the word *test* in place of the word *step*.
- **ProgressPercent**: Displays the progress percentage information for the current execution. This caption indicates the progress of operations for which the execution chooses to report the amount of progress. It does not necessarily reflect the progress of the execution as a whole. The progress percent is sent by a process model through the UIMsg_ProgressPercent UIMessageCode.
- **ProgressText:** Displays the progress message for the current execution. The progress text is sent by a process model through the UIMsg_ProgressText UIMessageCode.
- **ReportLocation:** Displays the location of the report for the current execution. The display value for a report is specified by the Report.Location property.
- **TestSocketIndex**: Displays the zero-based index of the test socket for the current execution. Displays nothing if the execution is not participating in a multi-socket test process. The text socket index is sent by a process model through the UIMsg_ModelState_BeginTesting and UIMsg_ModelState_Waiting UIMessageCodes.
- **UserName:** Displays the name of the current user.
- **UUTStatus**: Displays the result status of the UUT for the current execution. The UUT status is sent by a process model through the UIMsg_ModelState_TestingComplete UIMessageCode.

7.4 Menubar

The menubar in the FlexStand Operator Interface can be customized to contain both TestStand and user defined entries.
The specification of the menubar is done using an XML file located in the settings folder. The contents of the XML file can be modified in a simple text editor like Notepad. Figure 17 shows the connection between the menubar and the 'Toplevel' elements of the XML file.
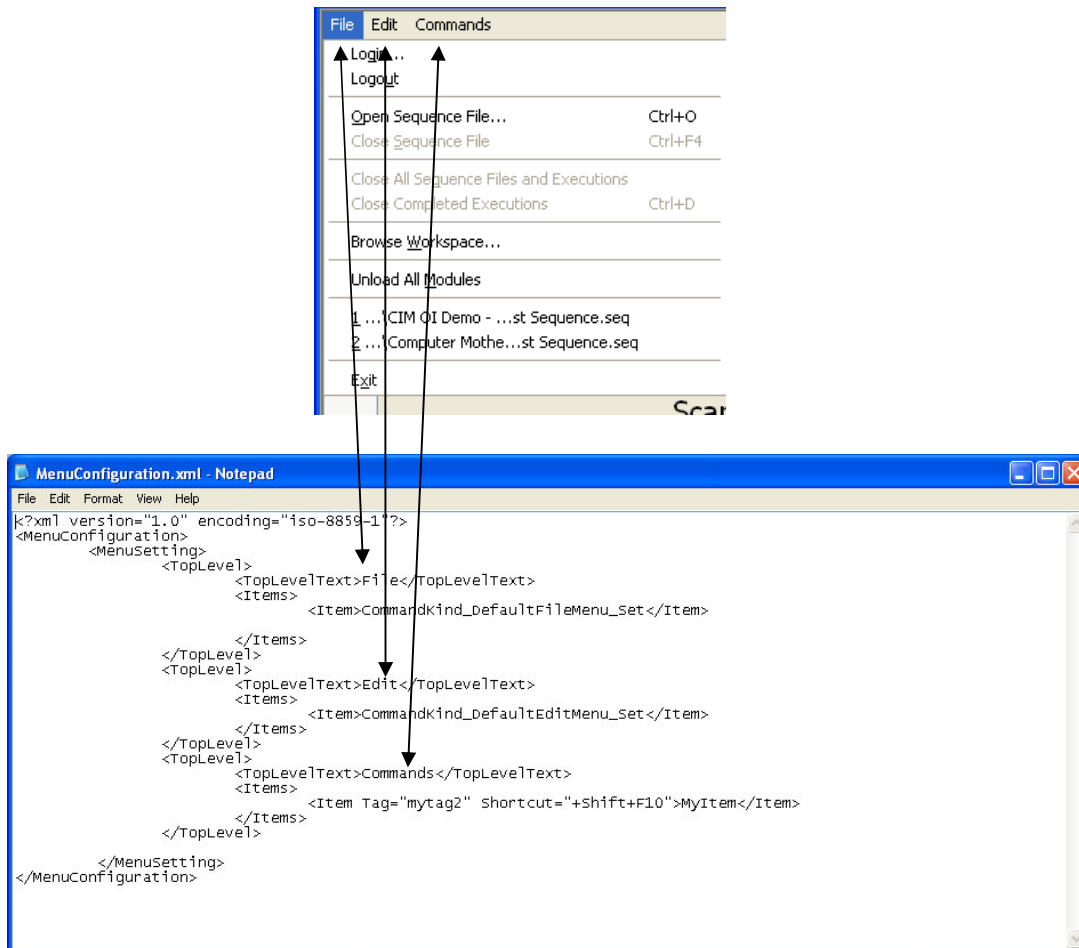

Figure 17: Menubar XML file.

Each <Toplevel> element must contain an entry for the menu text <TopLevelText> and an <items> section. In the <Items> section the elements of the toplevel menu are defined. The tag <Item> defines each element in the menu. If the Item is set to a built-in TestStand element like for example 'CommandKind_DefaultFileMenu_Set' this element will automatically be added to the menu. A user defined item can be added also by using the <Item> tag but the tag must also contain two attributes for the menu entry: Tag-name and shortcut. Shortcut is optional. The attributes must be entered as shown in the above example:

<Item Tag="mytag2" Shortcut="+Shift+F10">MyItem</Item>

"mytag2" is the tag text that will be returned to the event handlers in the plug-ins when the menu entry is activated. Menu entries are returned using the event 'UserMenu event'.

"Shortcut" is the optional shortcut key to use for the entry. The shortcut must have the format:

> [+Ctrl][+Shift][+]Shortcutkey.

'+Ctrl' and '+Shift' must be added if the shortcut needs to use Shift, Ctrl or both Shift and Ctrl as modifiers. If both Ctrl and Shift must be used the modifiers must be written in the sequence '+Ctrl+Shift'.
The plus in front of the Shortcut key is optional but is recommended to ease reading of the shortcut.

7.5 Language translation

Strings used in the FlexStand Operator Interface can be translated to a local language. The strings used in the FlexStand Operator Interface are defined in the file:

```
C:\Documents and Settings\All Users\Documents\National Instruments\TestStand
4.1.1\Components\Language\English\FlexStand OI CustomStrings.ini
```

The file contains two main sections: [FlexStand-OI-Menu] and [FlexStand-OI-Main]. The first section contains the strings for the user defined entries in the menu. The second section contains strings for the tab names and for strings used in the plug-ins. See section 0 for a description of the resource file format.

```
[FlexStand-OI-Menu]
;; Put entries for the menu here.
About FlexStand-OI="About FlexStand OI"
Help="Help"
File="File"

[FlexStand-OI-Main]
;; Put entries for the plugins here.
Main="Main"
Execution="Execution"
Report="Report"
F6 Buttontext="Serial no queue [F6]"
F7 Buttontext="N-loop [F7]"
Serialnumber="Serial number"
```

7.5.1 Menu translation

To translate the menu entries the names must have a prefix of TS__ (NOTE double underscore). TestStand uses TS_ (single underscore) as prefix for strings to be looked up but for menu items an underscore denotes a shortcut to the following letter. Therefore the lookup string must contain two underscores for menu items.
In the above example the menu File (…\FlexStand_OI\Settings\MenuConfiguration.xml) should have the text TS__File to be looked up correct. For this example the menu file will look like the following:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<MenuConfiguration>
          <MenuSetting>
                    <TopLevel>
                              <TopLevelText>TS__File</TopLevelText>
                              <Items>
                              …
                              </Items>
                    </TopLevel>
          </MenuSetting>
</MenuConfiguration>
```

7.5.2 Tab names translation

The texts in the tabs on the FlexStand Operator Interface can be translated by entering the strings in the [FlexStand-OI-Main] section of the file. The tab names used when calling the API function "Set Tab Names" should start with TS_.
To translate the Main tab in the example above the name should be set to TS_Main when calling the API function "Set Tab Names".

7.5.3 Plug-in string translation

Strings used in the plug-ins can be translated by using the following rules:

- Labview control and indicator <u>captions</u> will be translated if the <u>label</u> is set to a name with the format TS_<label>. <label> must be entered in the resource file mentioned above in the [FlexStand-OI-Main] section. The caption must be visible.
- For Labview Boolean buttons and indicators the Boolean text will be translated if the Boolean text starts with TS_.
- Labview free labels will be translated if the contents contains TS_. Note that the VI should be saved with the TS_<text> as free label. Use "Set current value as default" to make sure the VI is not saved with the translated text.
- TestStand controls and indicators are translated by using the default TestStand resource string method. See section 7.5.4.

7.5.4 Translation during execution

During execution of FlexStand strings can be translated by calling the API function "CIM_TS_API - Localize String.vi". This function can be used to convert user messages and other text during execution of FlexStand. Strings to be translated by using the API function must start with "TS_".

7.5.5 TestStand resource string method

TestStand uses a resource string method to obtain the string messages to display in the FlexStand Operator Interface, sequence editor and user interface windows and dialog boxes. The method uses a string category and a tag name and searches for the string resource in all string resource files in the following predefined order of directories:

1. `<TestStand Public>\Components\Language\<current language>`
2. `<TestStand Public>\Components\Language\English`
3. `<TestStand Public>\Components\Language`
4. `<TestStand>\Components\Language\<current language>`
5. `<TestStand>\Components\Language\English`
6. `<TestStand>\Components\Language`

Select **Configure»Station Options** in the TestStand editor to change the current language setting.
To customize a string resource file for a supported language or to create a resource file for a new language, copy an existing language file from the `<TestStand>\Components\Language\<language>` directory, place the file in the `<TestStand Public>\Components\Language\<language>` directory, and modify the file. To create a resource string file that applies to all languages, place the resource file in the base `<TestStand Public>\Components\Language` directory.

Note The TestStand Engine loads resource files when you start the TestStand engine. If you make changes to the resource files, you must restart FlexStand for the changes to take effect.

 String Resource File Format
String resource files must use the `.ini` file extension and use the following format:

```
[category1]
tag1 = "string value 1"
tag2 = "string value 2"
[category2]
tag1 = "string value 1"
tag2 = "string value 2"
```

When you create new entries in a string resource file or create a string resource file for custom components, use unique category names to avoid conflicts with the default names TestStand uses. For example, begin new category names with a unique ID, such as a company prefix.

You can create an unlimited number of categories and tag names. You can create strings of unlimited size, but you must break a string with more than 512 characters into multiple lines. Each line includes a `lineNNNN` tag suffix, where `NNNN` is the line number with zero padding, as shown in the following example:

```
[category1]
tag1 line0001 = "This is the first sentence of a long "
tag1 line0002 = "paragraph. This is the second sentence."
```

You can use the escape codes in Table 1 to insert unprintable characters.

Table 1: Resource String File Escape Codes

Escape Code	Description
\n	Embedded linefeed character.
\r	Carriage return character.
\t	Tab character.
\xnn	Hexadecimal value that represents the character.For example, `\x1B` represents the ASCII ESC character.
\\	Backslash character.
\"	Double quotation mark.