FLEXSTAND

FlexStand Operator Interface Reference Manual

Revision 8.0.0

Date: 2014-12-01

Contents

1 Preface

1.1 General

The flexible TestStand Operator 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 Software Development Kit (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.

The FlexStand Operator interfaces comes in two packages: One for developing the FlexStand Operator interfaces and one for use during deployment (Run-Time version). Refer to section 9 for a description of the FlexStand OI Run-Time for deployment.

1.2 Abbreviations

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

1.3 References

Ref.	Name	Location
[1]	NI TestStand Reference Manual	www.ni.com
[2]	NI TestStand Help	Start menu
[3]	FlexStand home page	www.flexstand.eu

2 System requirements for FlexStand Operator Interface

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

Operating system	Windows 7 (32 bit or 64 bit)
TestStand version	2014 (14.0)
LabVIEW version (Development system only required for development of operator interface components)	2014 (14.0)

When using a 64 bit operating system the LabVIEW version must be LabVIEW 32 bit.
The installer supports windows XP but FlexStand has not been fully tested on Windows XP.

3 Overview

The FlexStand operator interface consists of three main parts: Top plug-in, Main plug-in and bottom 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 is shown in Figure 1.

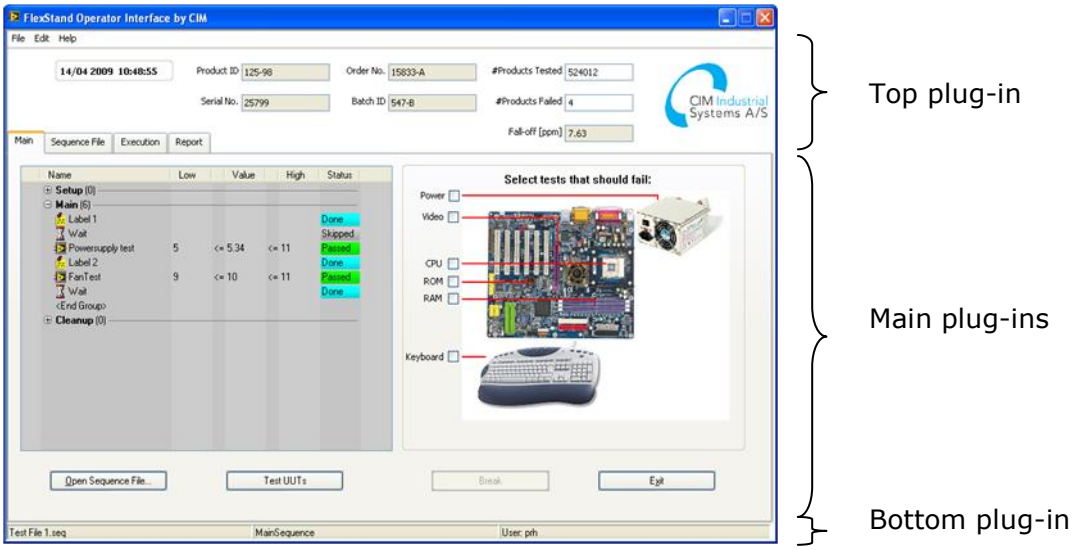


Figure 1: Plug-ins overview

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.

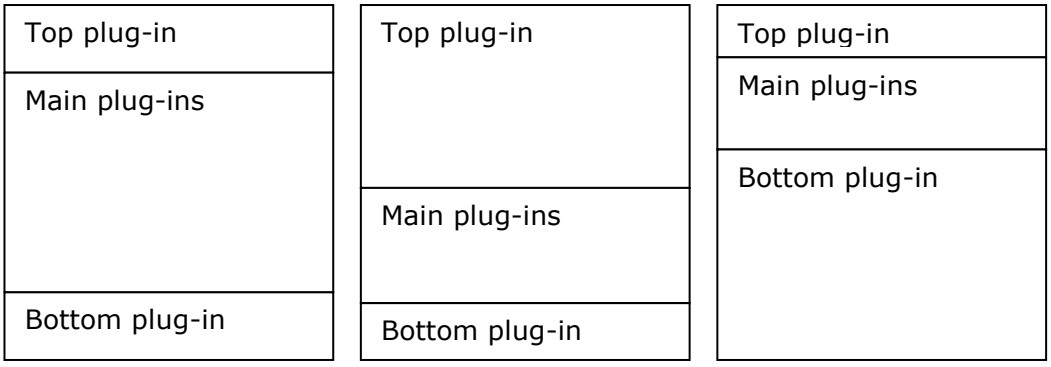


Figure 2: FlexStand operator interface layout

When FlexStand starts it first reads the Flexstand_OI.ini file. This file is located in the same directory as FlexStand_OI.exe.

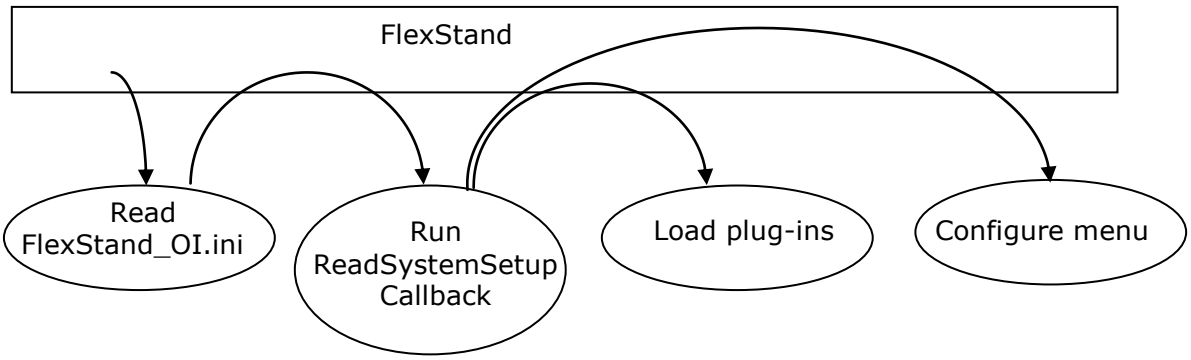


Figure 3: FlexStand start-up sequence

As shown in Figure 3 FlexStand read 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.

4 Getting started

This chapter contains a short description of how to install FlexStand and run the advanced example shipped with the Editor and Developer versions.

4.1 Download

FlexStand can be downloaded from www.flexstand.eu. FlexStand is available in three versions:

- FlexStand OI Developer
- FlexStand OI Editor
- FlexStand OI Run-Time

The following list outlines the difference between the versions.
Normally you would use Editor or Developer for creating your own operator interface and use a Run-Time version for the production lines.

Feature	**FlexStand OI Run-Time**	**FlexStand OI Editor**	**FlexStand OI Developer**
Plug-in architecture: Top plug-in, bottom plug-in, Supports up to 10 main plug-ins	√	√	√
Language localization	√	√	√
Automatic resize of GUI	√	√	√
TestStand Deployment Engine support	√	√	√
LabVIEW Run-Time support	√	√	√
Custom menu: Supports TestStand standard menu and custom menu entries	√	√	√
Tab control including automatic tab change and show/hide tabs	√	√	√
Callbacks	√	√	√
Command line parameters: Supports TestStand standard parameters and Custom defined parameters	√	√	√
FlexStand Configuration Tool		√	√
LabVIEW API palette			√
Simple example	√		√
Standard example			√
Advanced example includes Editor		√	√
Programmers Reference manual			√
Editor Reference manual		√	

4.2 Upgrade

If you need to upgrade an older version of FlexStand OI to a new version please refer to section 10 for upgrade instructions.

4.3 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 extract the downloaded .zip file and run the setup.exe. Read and accept the license agreement. Normally you should press next in all dialogs during the installation.

4.4 Running FlexStand

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

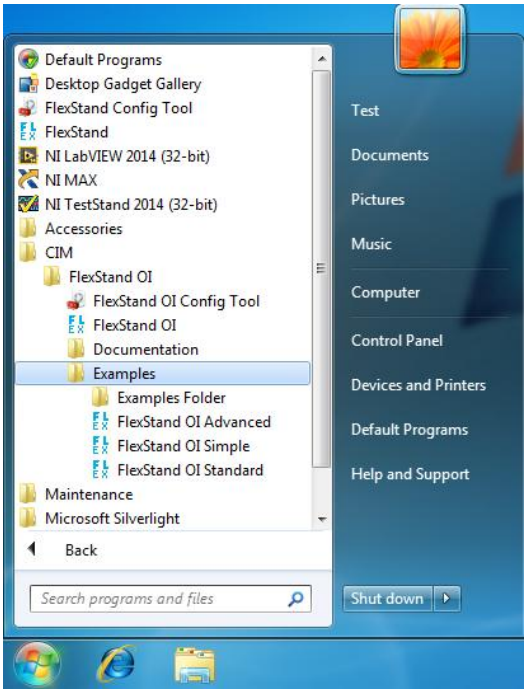


Figure 4: FlexStand Operator Interface in Start menu

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

The first time FlexStand is started the license dialog is shown. By following the instructions in the license dialog you can purchase a license or activate a license. If you want to run a trial version you can press "Evaluate FlexStand Operator Interface".

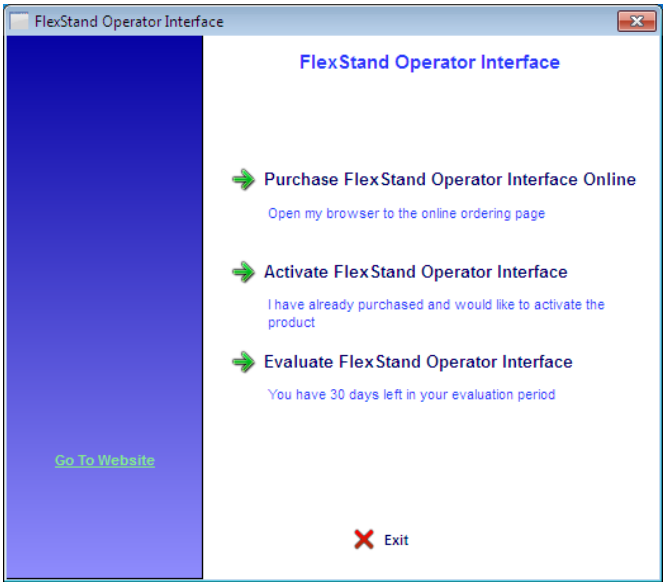


Figure 5: FlexStand license dialog

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

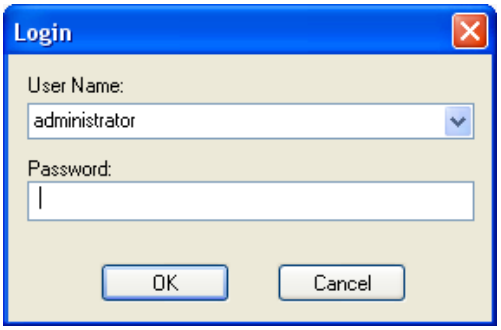


Figure 6: TestStand login prompt

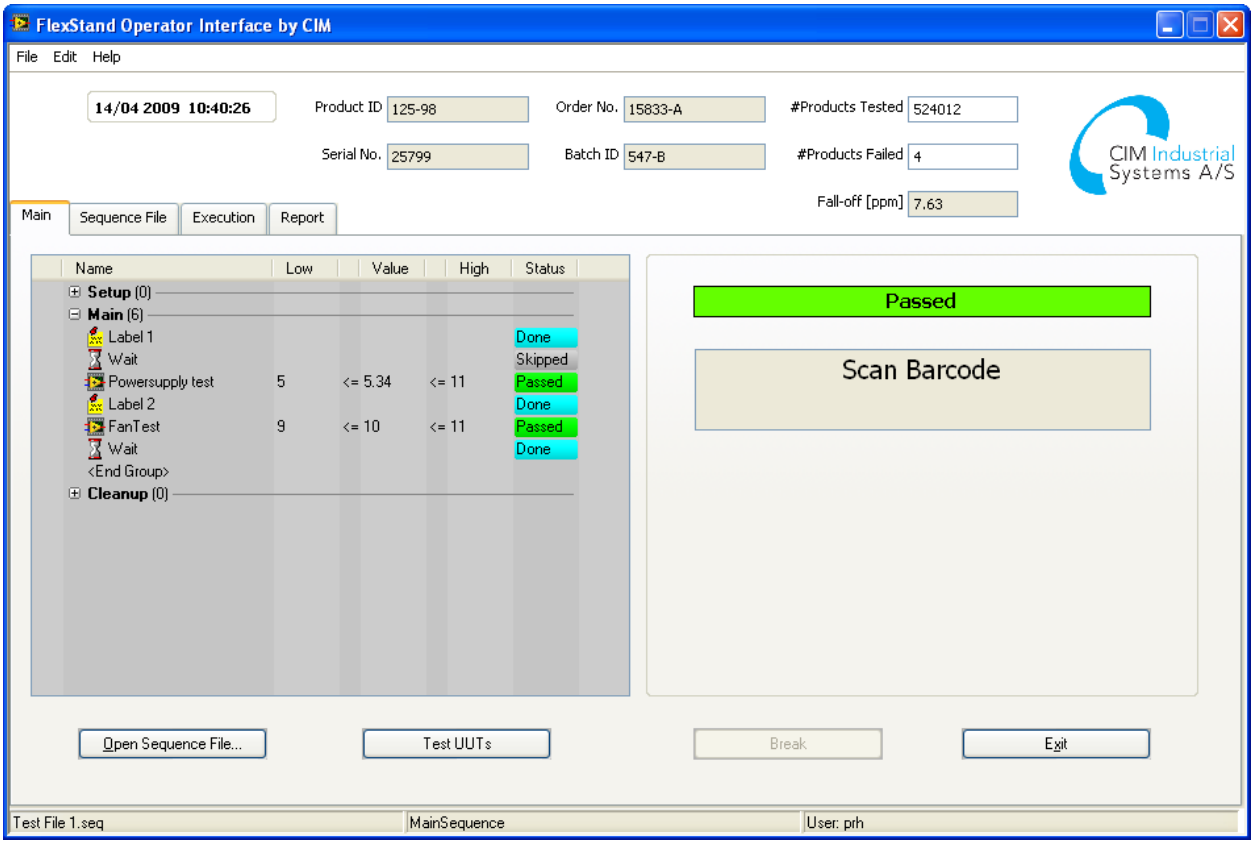


Figure 7: CIM Operator Interface loaded.

From the main plug-in 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:

Windows 7:
C:\Users\Public\Documents\National Instruments\TestStand <version>\UserInterfaces\FlexStand OI Examples\Advanced\DemoSequence

Windows XP:
C:\Documents and Settings\All Users\Documents\National Instruments\TestStand <version>\UserInterfaces\FlexStand OI Examples\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.

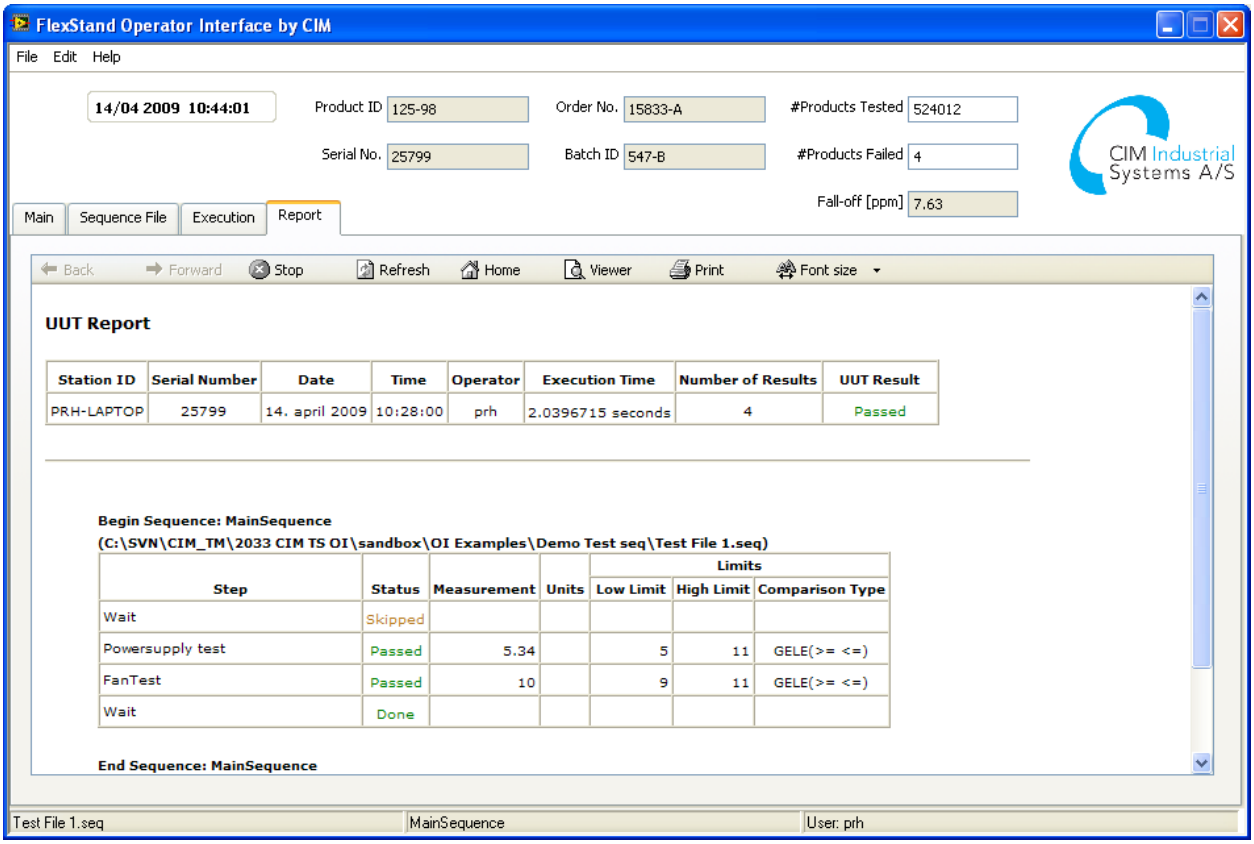


Figure 9: Report view.

4.5 Editing Steps in a Sequence

You can add a step to a sequence, configure the step to call a code module or subsequence, and configure the properties of a step.

The Insertion Palette contains a set of predefined step types you can add to sequences. Step types define the standard behavior and a set of step properties for each step of that type. All steps of the same type have the same properties, but the values of the properties can differ.

When you build sequence files, you can also use the Templates list in the Insertion Palette. Use the Templates list to hold copies of steps, variables, and sequences you reuse during the development of sequence files. For example, you can add a step that calls a specific LabVIEW VI you typically use or a sequence that contains common setup steps, cleanup steps, and local variables.

Refer to the "Getting Started with TestStand" for more information about all the features covered in this chapter (http://www.ni.com/pdf/manuals/373436f.pdf).

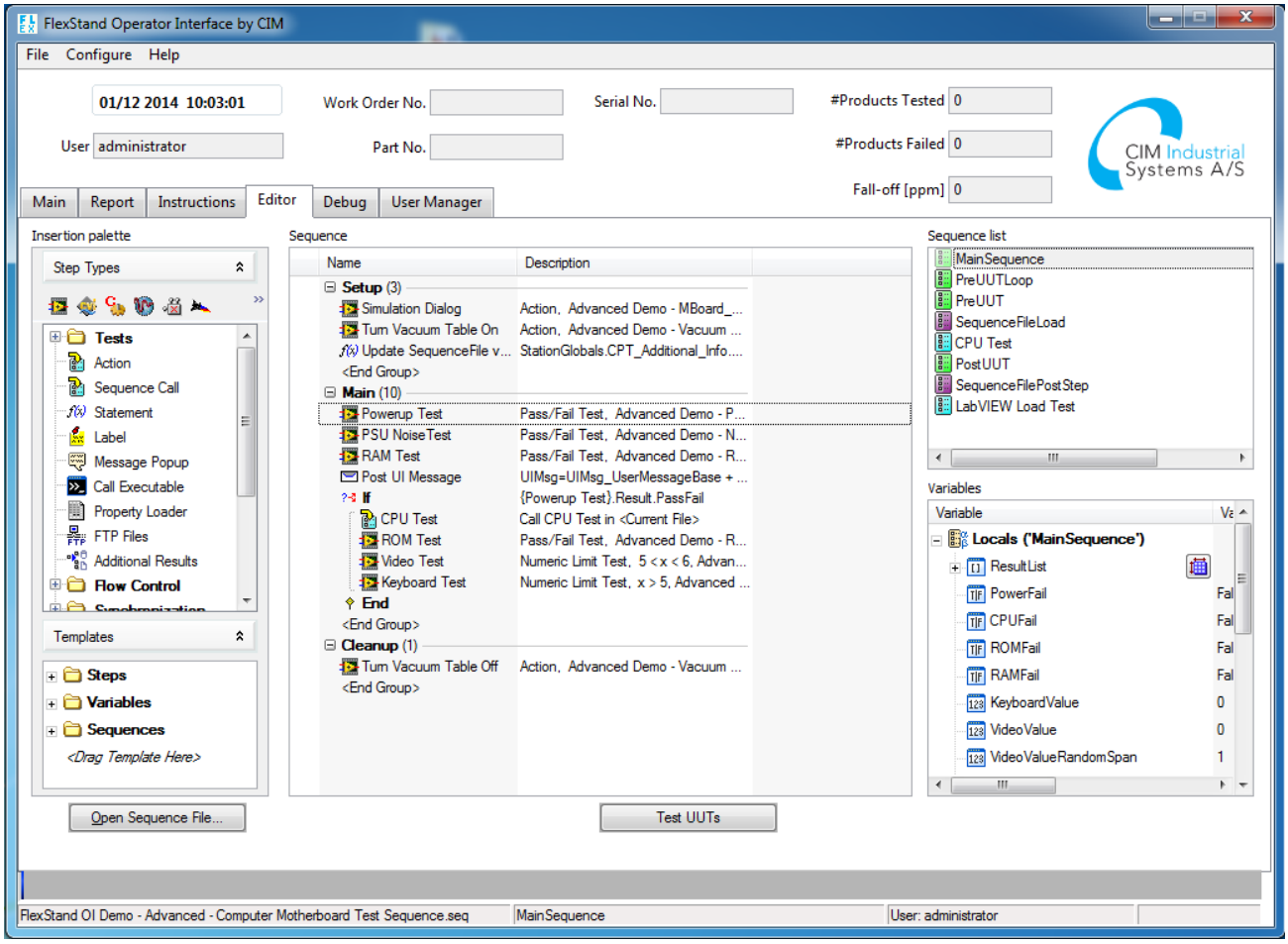


Figure 10: The FlexStand Sequence Editor

4.5.1 Adding a new step

Steps can be added to the sequence by drag and drop from the Insertion Palette. The example in Figure 11 shows how to add a delay after the Powerup Test.

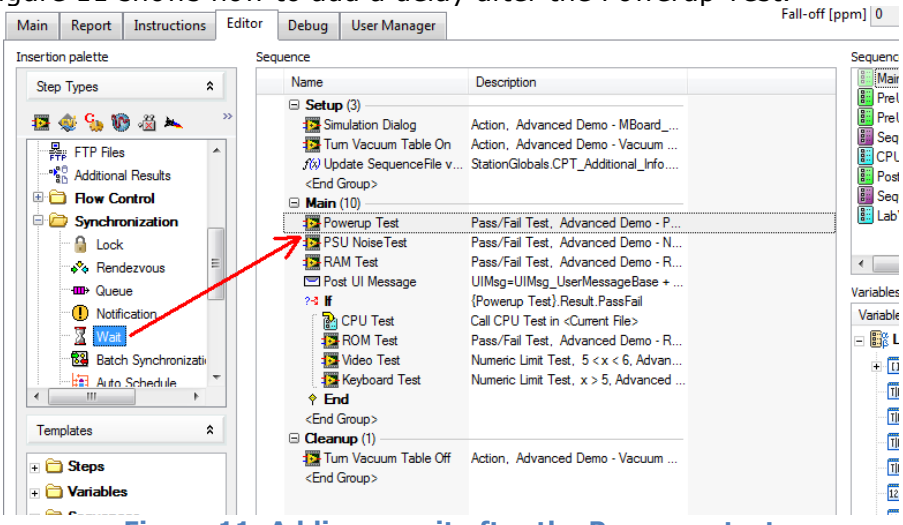


Figure 11: Adding a wait after the Power up test

4.5.2 Edit the step settings

To edit the step settings right click on the step in the sequence and select "Configure wait…" (For other types of steps this may sometimes be called "Edit…" or "Configure…").

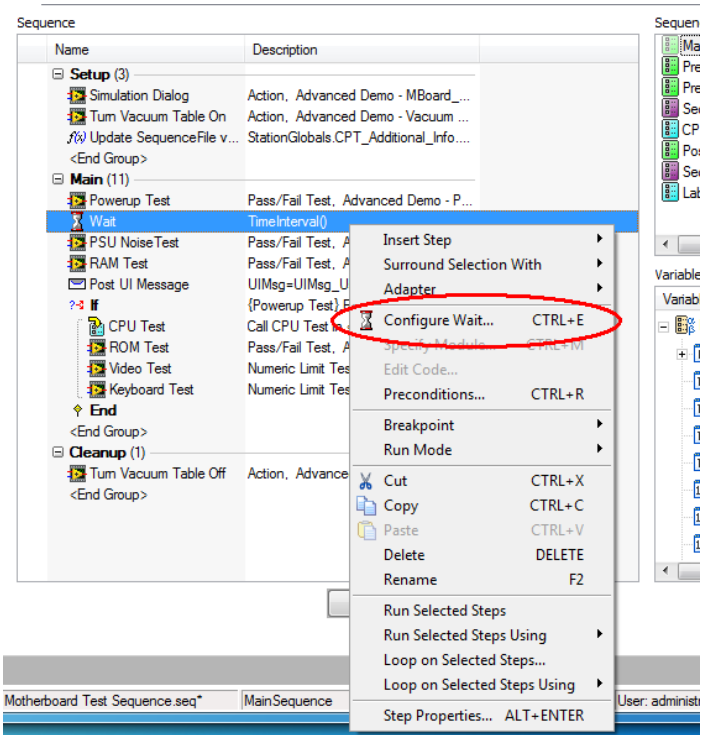


Figure 12: Configure a step

Selecting ”Configure Wait…” will open the configuration dialog for this type of step. The settings for this step can then be set. I this case a delay of 1.5 seconds. Pressing OK will save this setting in the sequence for this specific step.

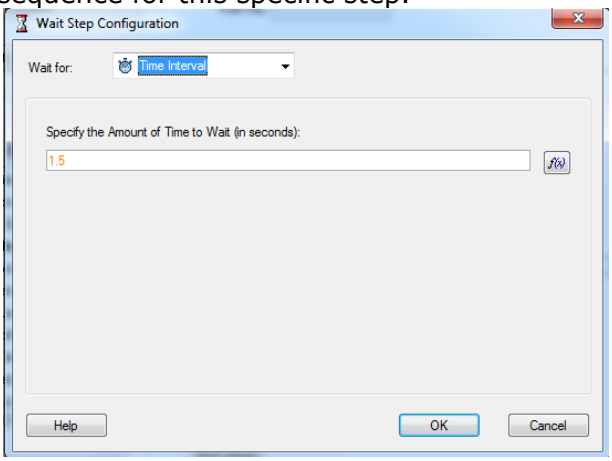


Figure 13: Configure the settings of a step.

4.5.3 Steptypes installed with FlexStand

Two steptypes are installed with the FlexStand OI. The steptypes are located in the palette under the CIM menu.

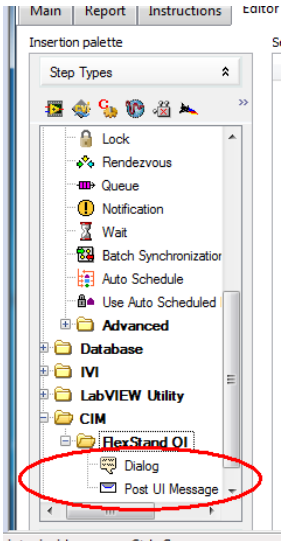


Figure 14: FlexStand OI steptypes

The Dialog steptype can be used to insert a step that will show a custom dialog in the FlexStand OI.

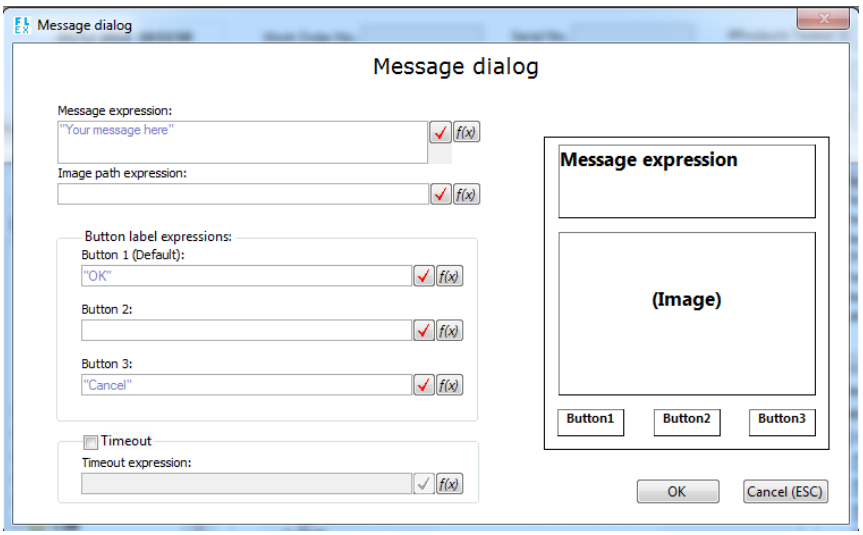


Figure 15: Configuring a dialog for FlexStand.

The Post UI Message steptype can be configured to send a UI message to the operator interface. The UI message can contain various information for the operator interface. The example sequence shown above calls this steptype for setting the document to show in the Instructions tab. In this case the FlexStand reference manual. Refer to the TestStand Help for more information about posting UI messages.

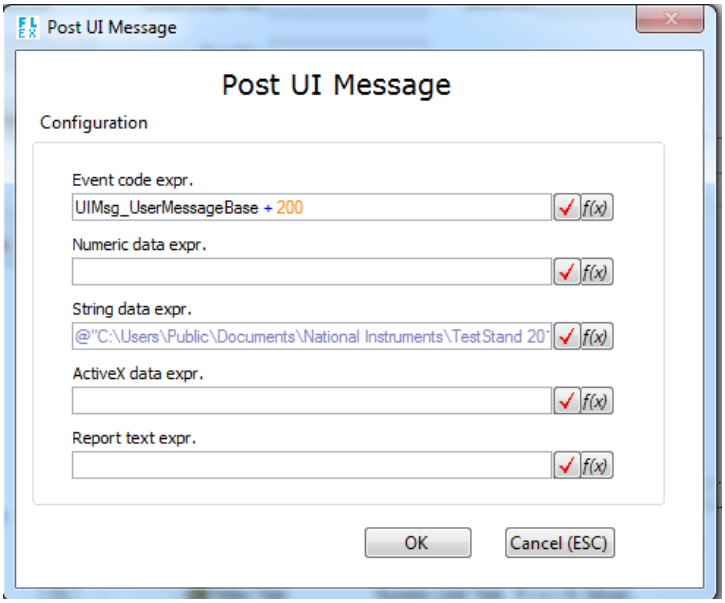


Figure 16: Posting a UI message to the operator interface.

4.5.4 Preconditions and step properties

The context menu shown in Figure 12 contain items for running and modifying the step. "Preconditions…" is a common setting to specify conditions for running the step. "Step Properties…" is another common setting where looping, pre/post expressions and other settings can be configured.

5 Creating a custom operator interface

If you want a different operator interface than the examples shipped with FlexStand, you can use the examples as a starting point. This section describes the procedure for creating a new operator interface. For configuration of the FlexStand OI we recommend using the FlexStand OI Configuration Tool instead of direct modification of the ini files. Refer to section 6 for a description of the configuration tool.

5.1 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. Place the new directory in a folder under <TestStand public>\UserInterfaces\.. Rename this new directory to a descriptive name, for example "myOI". Refer to Figure 17.

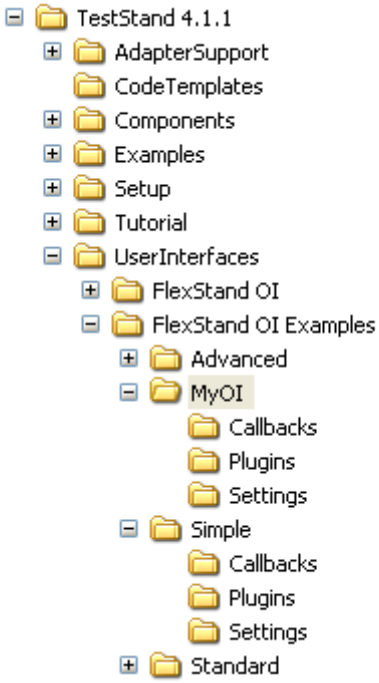


Figure 17: New folder for custom operator interface.

5.2 Selecting the new Operator Interface

To run the new operator interface, FlexStand must know where to find the new files. To set the location of the Operator Interface files use the FlexStand configuration Tool. Refer to section 6 for a description of how to use the configuration tool.

5.3 Modify Plugins

The new folder 'Plugins' contains the plug-ins used in the example. Using LabVIEW the plug-ins can be modified to meet the specific requirements.
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 bottom plug-in and one main plug-in defined.

5.4 Modify plug-in list

Use the FlexStand Configuration Tool if you want to change the list of plug-ins. You can add new or remove some of the example plugins. Refer to section 6.3 for a description of how to use the configuration tool to modify the list of plug-ins.

5.5 Run new operator interface

Save all open files and close the FlexStand Configuration Tool. Run FlexStand OI by selecting Start>>All Programs>>CIM>>FlexStandOI>>FlexStand OI. Your new FlexStand operator interface will now start.

6 FlexStand OI Configuration Tool

By using the FlexStand OI Configuration Tool you can easily modify the FlexStand OI settings.
The configuration tool can be found on the desktop or in the Start menu:
All Programs->CIM->FlexStand OI->FlexStand OI Config Tool.

When launching the configuration tool a dialog will ask for the directory of your FlexStand files. This usually points to a custom directory created in the same folder as the FlexStand examples. If you have created a modified copy of one of the shipped examples you should select this new folder in the dialog box.

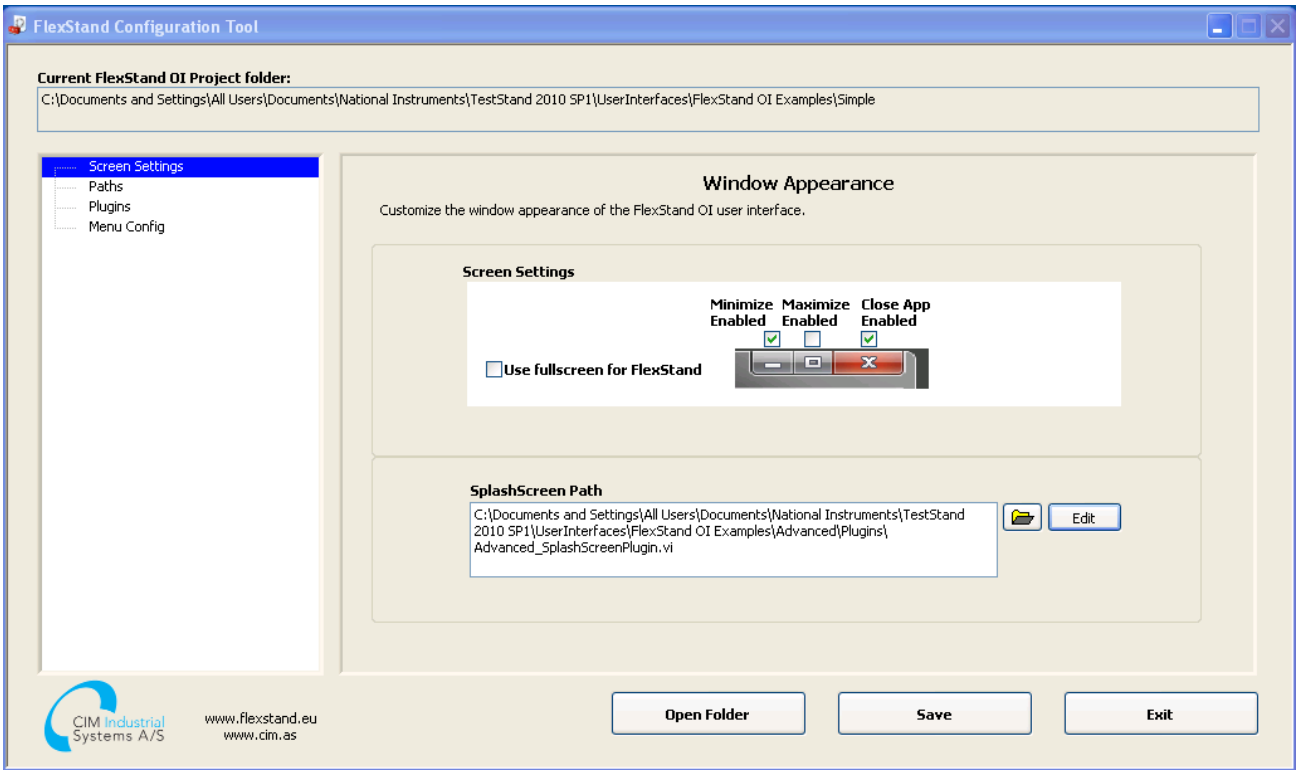


Figure 18: FlexStand Configuration Tool

6.1 Screen settings

On the screen shown in the figure above the overall look of FlexStand OI can be configured. It is also possible to change the splash screen shown at startup. The Edit button can be used to edit the vi for the splash screen. This requires LabVIEW development to be installed.

6.2 Paths

The Path settings lets you define the various paths used be FlexStand OI.

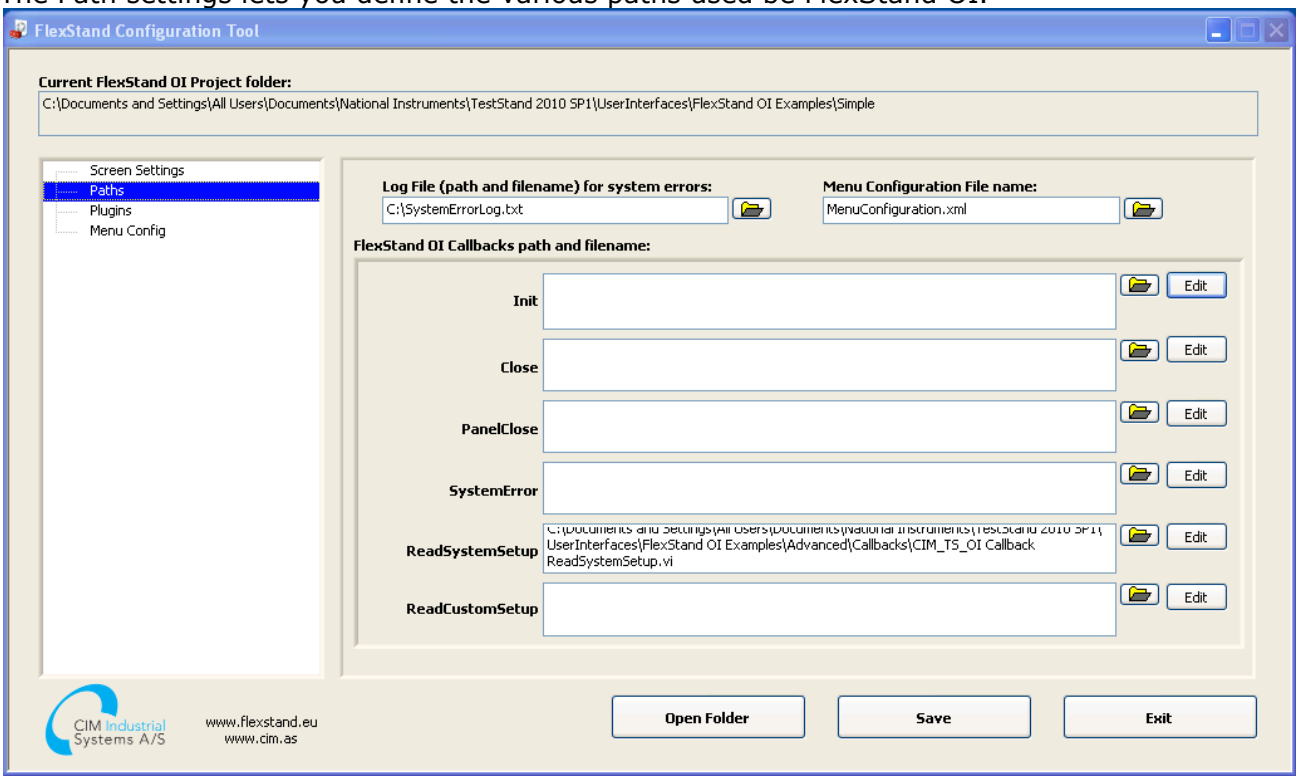


Figure 19: FlexStand Configuration Tool, Paths setting

6.3 Plugins

The plugins used in the current FlexStand OI can be configured here.

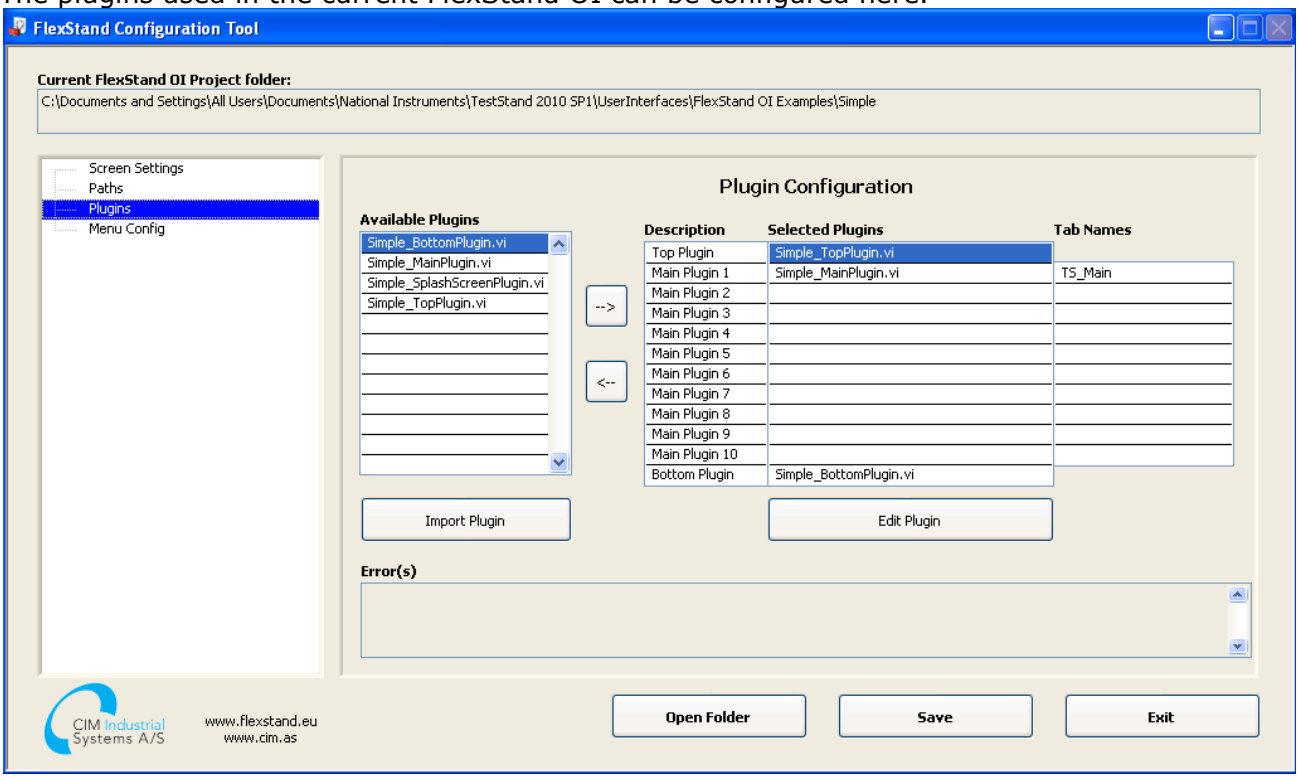


Figure 20: FlexStand Configuration Tool, Plugins

Selected Plugins shows the plugins currently used in FlexStand.

Available Plugins shows a list of plugins currently available in the plugins directory. Plugins from other projects or examples can be added by using the Import Plugin button.

Tab names defines the name shown in the tab. If the name is prefixed with TS_ it can be translated to local language if the entry is added to the language files. Refer to section 8.4 for a description of the localization feature.

By pressing the Edit Plugin button the selected plugin can be opened in LabVIEW for editing.

6.4 Menu config

The pull down menu in FlexStand can be configured to contain both TestStand build in entries and custom entries. The menu config defines the contents of the menus and the tags to be used in the plugins (for custom entries).

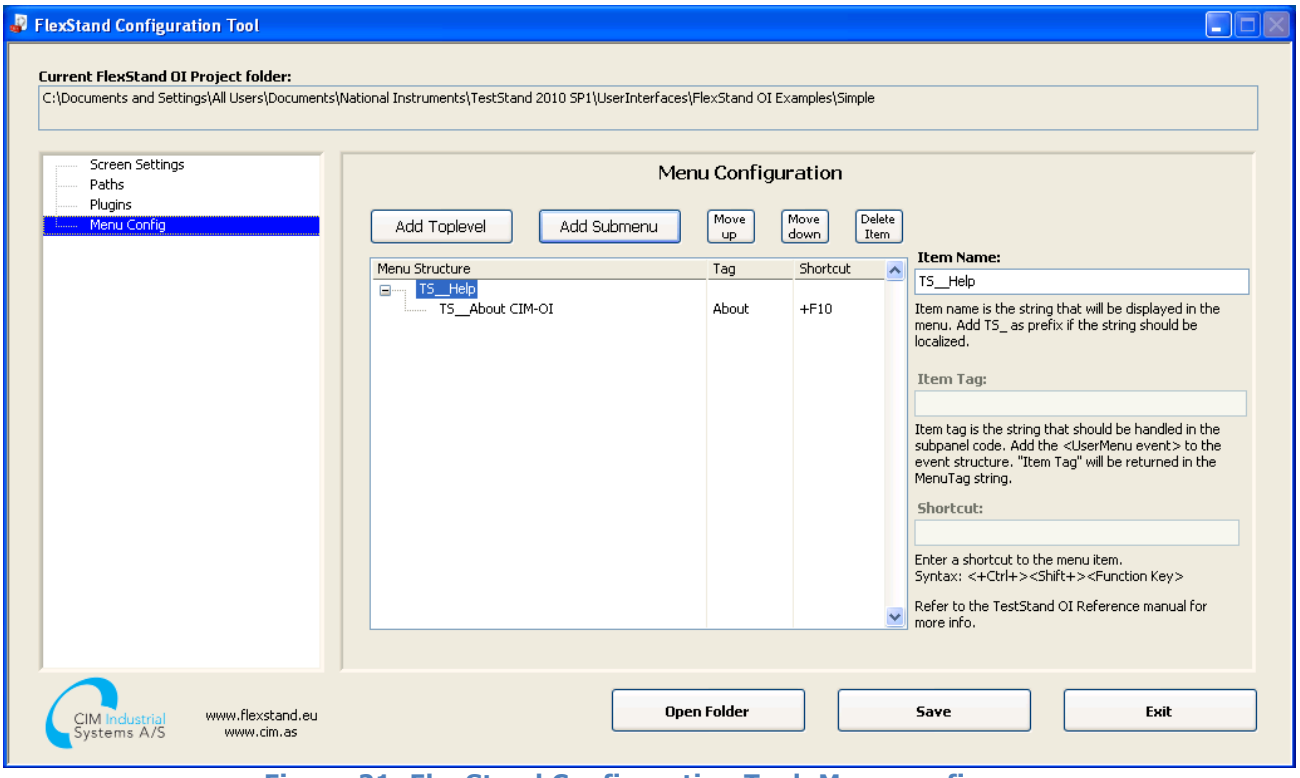


Figure 21: FlexStand Configuration Tool, Menu config

Item Name is the name shown in the menu. If the name is prefixed with TS_ it can be translated to local language if the entry is added to the language files. Refer to section 8.4 for a description of the localization feature. The build-in menus in TestStand can be used as Item name. A list of available build-in TestStand items can be found in the TestStand help by searching for "CommandKinds". Figure 22 shows an example of how to use the default file menu from TestStand in the FlexStand OI menu.

Item Tag is the Menutag (string) that is returned with the UserMenu event when the user selects the menu entry. This is typically included in the event handler of one of the main plug-ins. If a TestStand built-in menu item is used the FlexStand kernel will execute the menu item and a user event is not sent to the event handler. In this case the Item Tag must be empty. Refer to Figure 22.

Shortcut. A keyboard shortcut can be defined for the entry. The shortcut is a text string of the format [+Ctrl][+Shift][+]Shortcutkey.

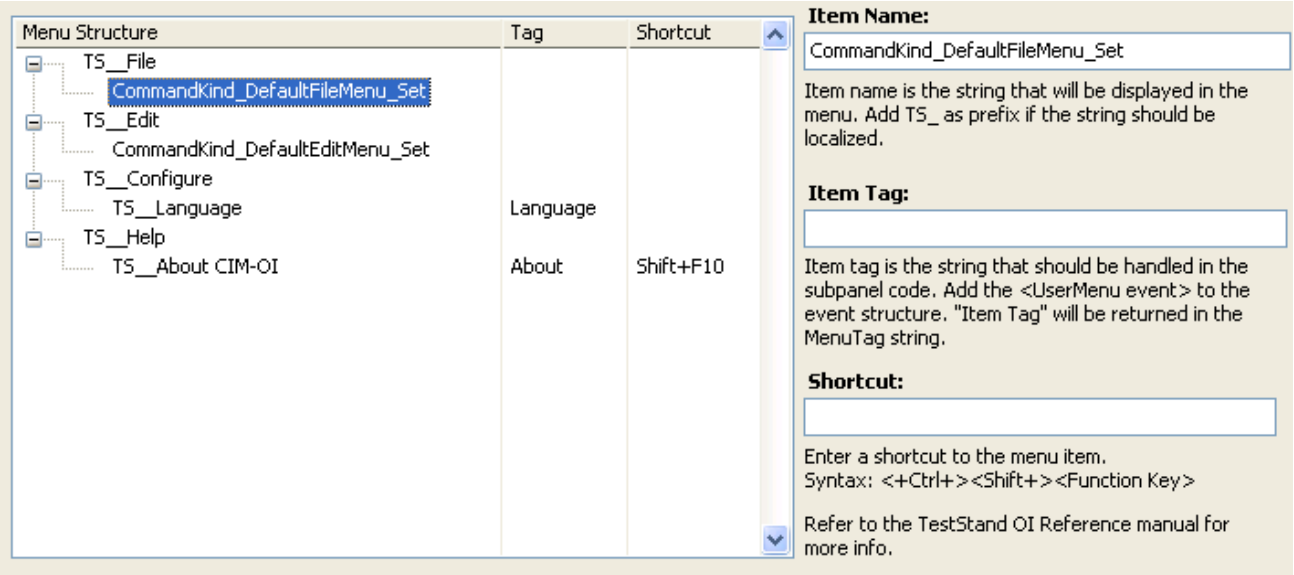


Figure 22: Example of using the TestStand default File menu.

7 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:

Windows XP:
C:\Documents and Settings\All Users\Documents\National Instruments\TestStand <version>\UserInterfaces\FlexStand OI Examples

Windows 7:
C:\Users\Public\Documents\National Instruments\TestStand <version>\UserInterfaces\FlexStand OI Examples

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

8 FlexStand Reference

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

8.1 LabVIEW palette

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

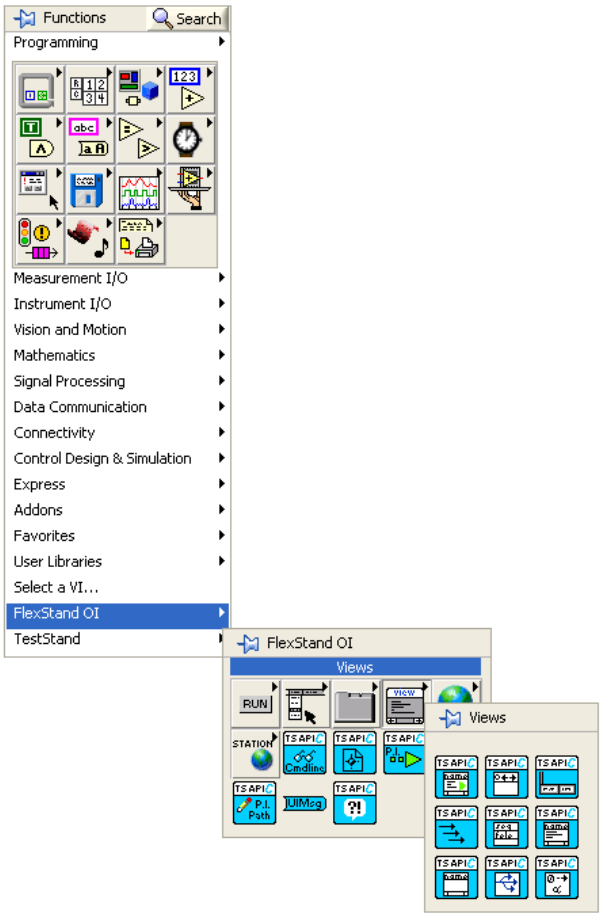


Figure 23: FlexStand palette in LabVIEW

8.2 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. Called before the TS kernel is loaded.	Default empty. Can be used for logging of program start or to start programs that needs to run in parallel with the Operator Interface.
ReadSystemSetup	Start of Operator Interface. Called after the TS kernel is loaded.	Sets the system parameters for: Plug-in Paths, Tab names, Menu specification file, System error handling.
ReadCustomSetup	Start of Operator Interface. Called after the TS kernel is loaded and immediately after the ReadSystemSetup callback.	Default empty. Can be used to read custom settings that must be available to the plug-ins for later use. See example below.
PanelClose	Called when the operator presses the Windows Close function. The callback is called before the TestStand engine is shut down.	Default empty. Use the callback for closing running executions and clean-up of the application before exit. This callback is only used if the CloseAppEnabled is set to True in the FlexStand_OI.ini file in the [Settings] section.
Close	Shut down of the Operator Interface. The callback is called after the TestStand engine has been shut down and immediately before FlexStand closes.	Default empty. Contains example code of how to shut down the computer. 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.

The operator interface is delivered with a set of template callbacks that can be used as a starting point for creating a customized callback.
By using the FlexStand Config Tool, the location of the callbacks can be changed. Refer to section 6.2.
Note that callbacks with an empty paths will not be called.

8.2.1 ReadCustomSetup callback

The ReadCustomSetup callback can be used to execute code at the beginning of the FlexStand execution. The ReadCustomSetup callback is called after the TS kernel has been loaded but before the login. An example of functionality could be to set some settings that must be available during the execution of the current FlexStand session.

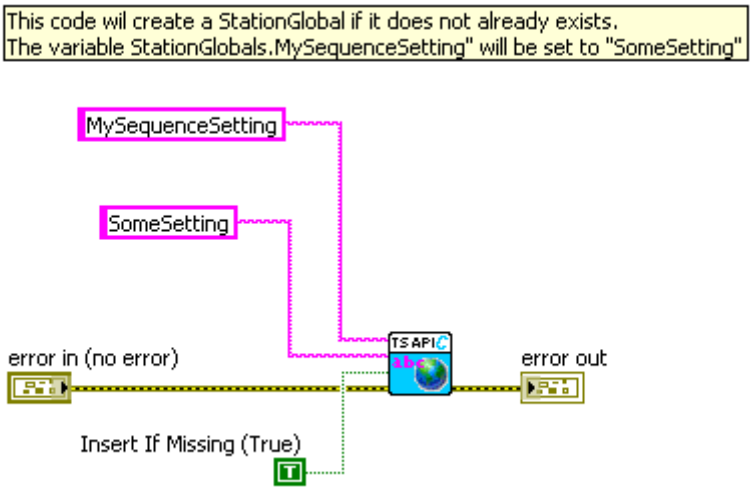


Figure 24: Example of ReadCustomSetup callback.

8.3 Plug-ins

8.3.1 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 25 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.

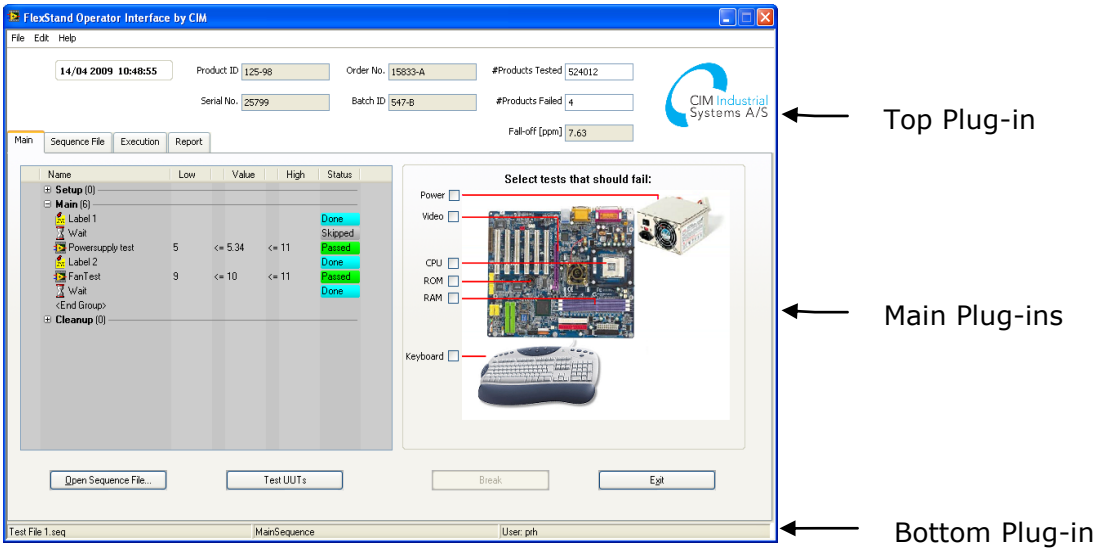


Figure 25: Plug-in layout.

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

8.3.2 Plug-in contents

The plug-ins is 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.

8.3.2.1 Event handler

Figure 26 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.

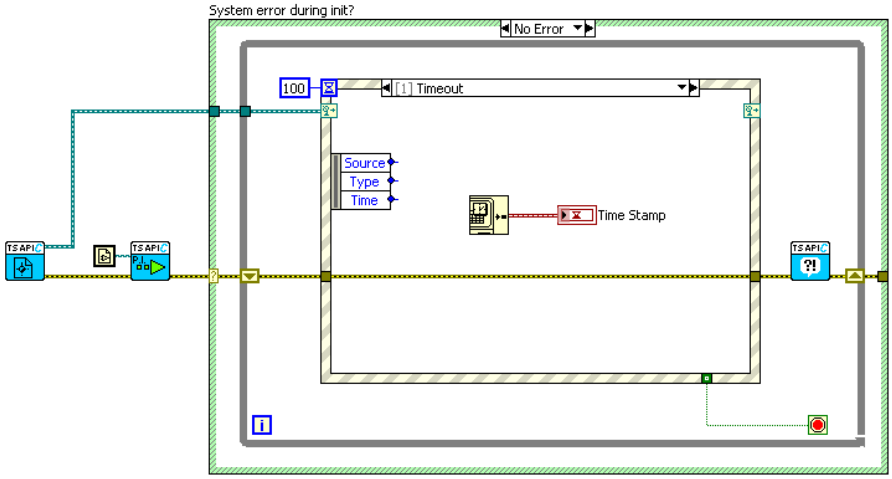


Figure 26: Plug-in code.

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 27 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.

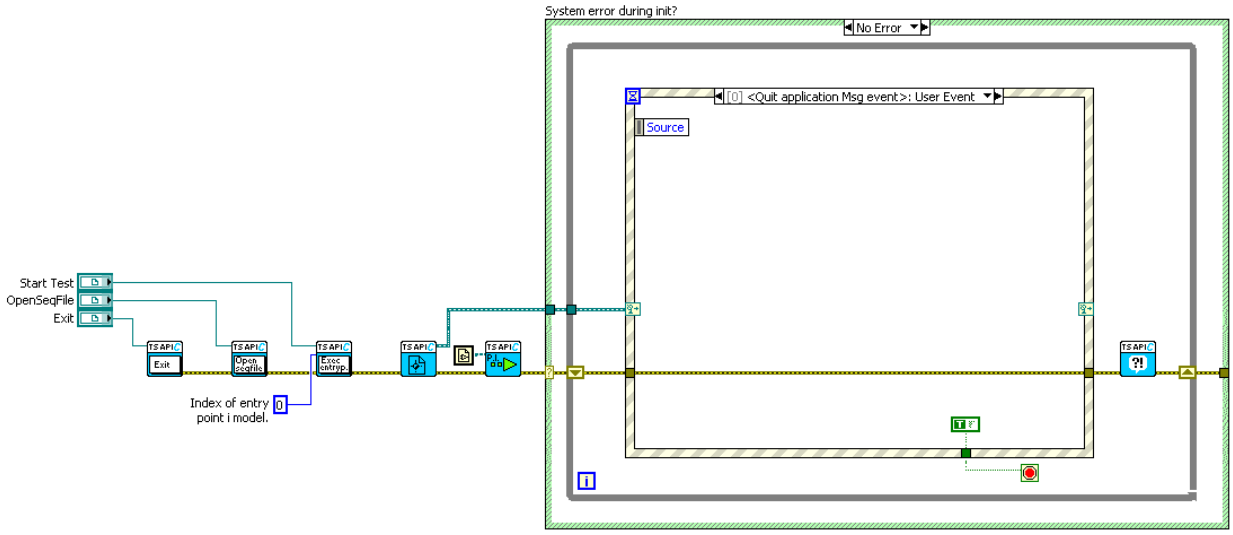


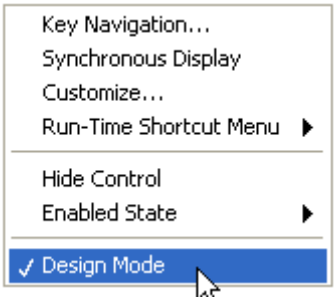
Figure 27: Quit application event.

8.3.2.2 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.

8.3.2.3 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.

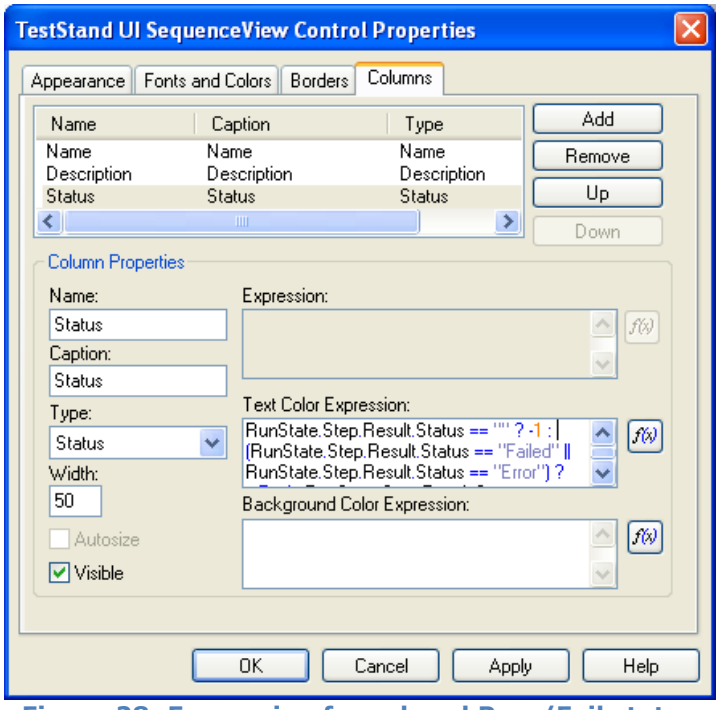


Figure 28: Expression for colored Pass/Fail status

The expression used is:

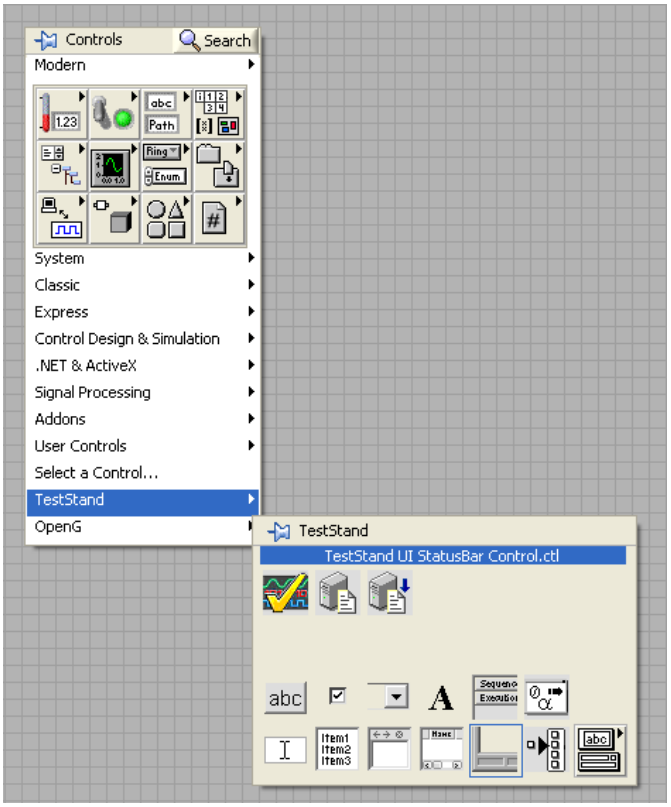
```
RunState.Step.Result.Status == "" ? -1 : (RunState.Step.Result.Status == "Failed" ||
RunState.Step.Result.Status == "Error") ? tsRed : RunState.Step.Result.Status == "Passed"
? tsGreen : RunState.Step.Result.Status == "Skipped" ? tsLightGray :
(RunState.Step.Result.Status == "Running"
|| RunState.Step.Result.Status == "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.

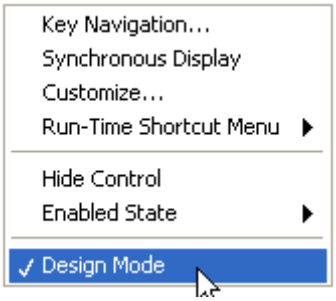
8.3.2.4 Status bar

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 plug-in. 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.

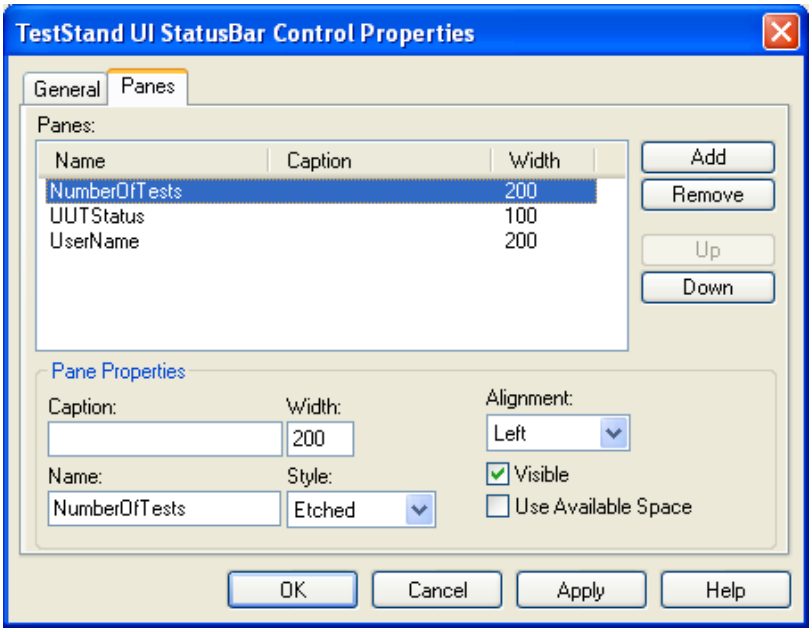


Right click the StatusBar and select Advanced. Make sure Design Mode is checked:



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.

8.4 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:

```
<TestStand Public Dir>\Components\Language\English\FlexStand OI CustomStrings -
<language>.ini
```

The file contains two main sections: [CIM-OI-Menu] and [CIM-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 8.4.5 for a description of the resource file format. Example of custom strings:

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

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

8.4.1 Menu translation

To translate the menu entries the names must have a prefix of TS__ (NOTE: Double underscore).

You may want to use shortcuts for menu entries like pressing Alt+F for File. In this case the entry must be named TS__File (NOTE: Double underscore) as the Item name defined in the FlexStand configuration Tool. In the translation file, the menu entry must be defined in the [CIM-OI-Menu] section:

```
[CIM-OI-Menu]
File="&File"
```

The ampersand (&) must be placed before the letter used for the shortcut. In this case pressing Alt+F will activate the File menu.

8.4.2 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".

8.4.3 Plug-in string translation

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

- LabVIEW control and indicator captions will be translated if the label 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 8.4.4.

8.4.4 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_". For plug-ins the text must be added to the section [CIM-OI-Main] in the language files.

8.4.5 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.

9 Deployment of FlexStand OI

When you need to deploy your FlexStand Operator Interface on a production line you can chose to install a FlexStand Developer or Run-Time license. We recommend you use a Run-Time license for the production line. The license for the Run-Time version of FlexStand OI is cheaper but does not allow you to develop FlexStand OI plug-ins. All software developed on the FlexStand Developer will run on the Run-Time option of FlexStand without limitations.

Summary of license requirements:

- You will need one license for each machine running FlexStand OI. This can be FlexStand OI Developer or FlexStand OI Run-Time.
- It is a requirement that you buy at least one Developer License.

See www.flexstand.eu for a product overview and how to buy a license.

9.1 Deployment methods

We recommend one of two methods for deploying FlexStand to a production line. Choosing a method depends on the number of deployments needed. Other methods are available, refer to the Deploying TestStand Systems in the TestStand Reference Manual.

9.1.1 Deployment using TestStand Deployment Utility

If you need to distribute your sequences and VIs to a large number of production lines you could consider using the TestStand Deployment Utility. Using this method you can create a complete install package that includes TestStand Run-Time, LabVIEW Run-Time, drivers, sequences and all the VIs needed by your sequences.

This method is a simple way of distribution but the drawback is that you will need to make a complete new install package if you have changes to sequences or VIs. You will also need to run the complete installer on the production line for updates.

9.1.2 Deployment by file copying

If you need to distribute your sequences to a limited number of production lines and/or you need to have full control of the update of single files like sequences or VIs consider updating the production lines by copying files.
This method is normally used in conjunction with a revision control system like Subversion (Tortoise SVN). The revision control system can then be used to keep the production lines up to date to a specific release of the test software.

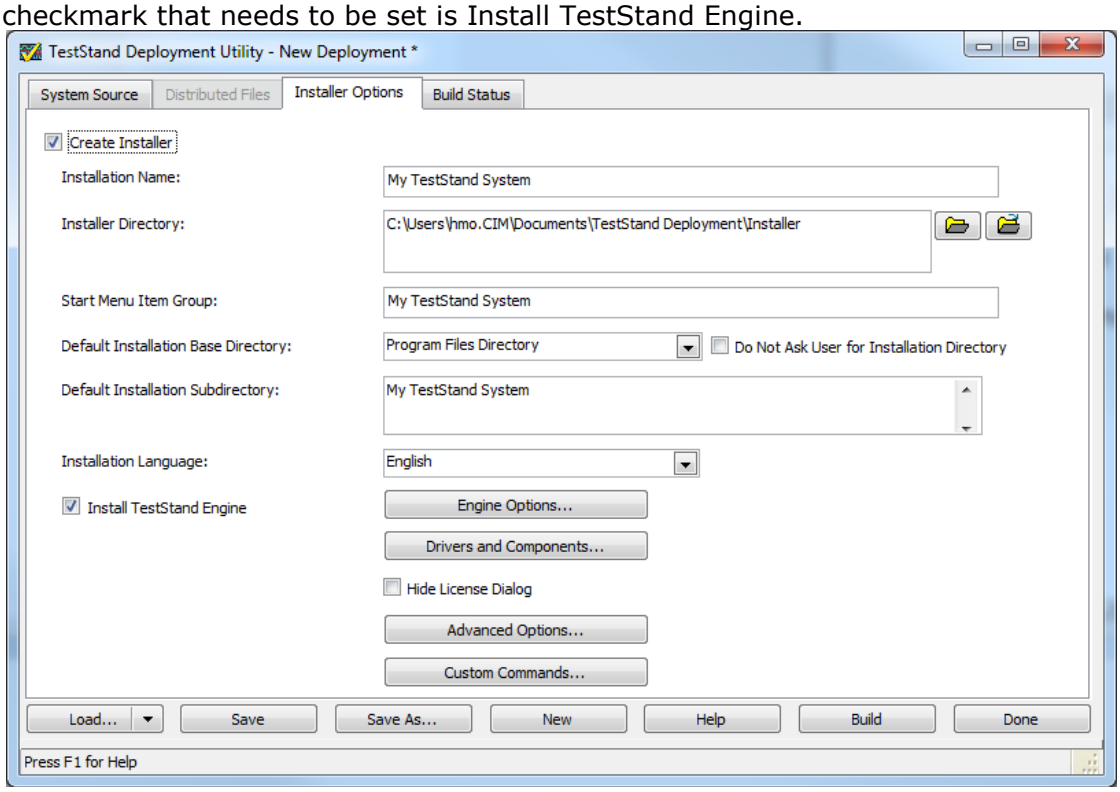
This method gives you full control of the source code and you can replace or update any files individually. The drawback is that you need to make sure you have all the VI's needed by the steps in your sequence. This may include a mass compiled version of vi.lib, instr.lib and other components required by the code.

In order to use this method you will still need to build an installer with TestStand and LabVIEW Run-Time as a minimum. If you need other drivers you could also include these in the deployment. The following chapter describes how to create a base installer using the TestStand Deployment Utility.

9.2 TestStand Deployment

In order to run TestStand on a production line PC a TestStand deployment needs to be installed. The deployment installer is created from the TestStand Editor using the TestStand Deployment Utility.

The TestStand Deployment Utility can be found in the Tools menu -> Deploy TestStand Engine.
For a standard FlexStand OI installation an "empty" deployment is needed. The only checkmark that needs to be set is Install TestStand Engine.



You may include other drivers and components if needed. You may also include the LabVIEW Run-Time version(s) needed by your sequences.

Build the deployment.

Note: During the build process you may need the TestStand and drivers install disks.

Run the deployment installer created above on the PC for the production line.

9.3 TestStand Settings

To copy the TestStand settings (for example Station Globals, User logins and Report Options) copy the relevant files from the <TestStand>\cfg to the Run-Time PC in the same folder.

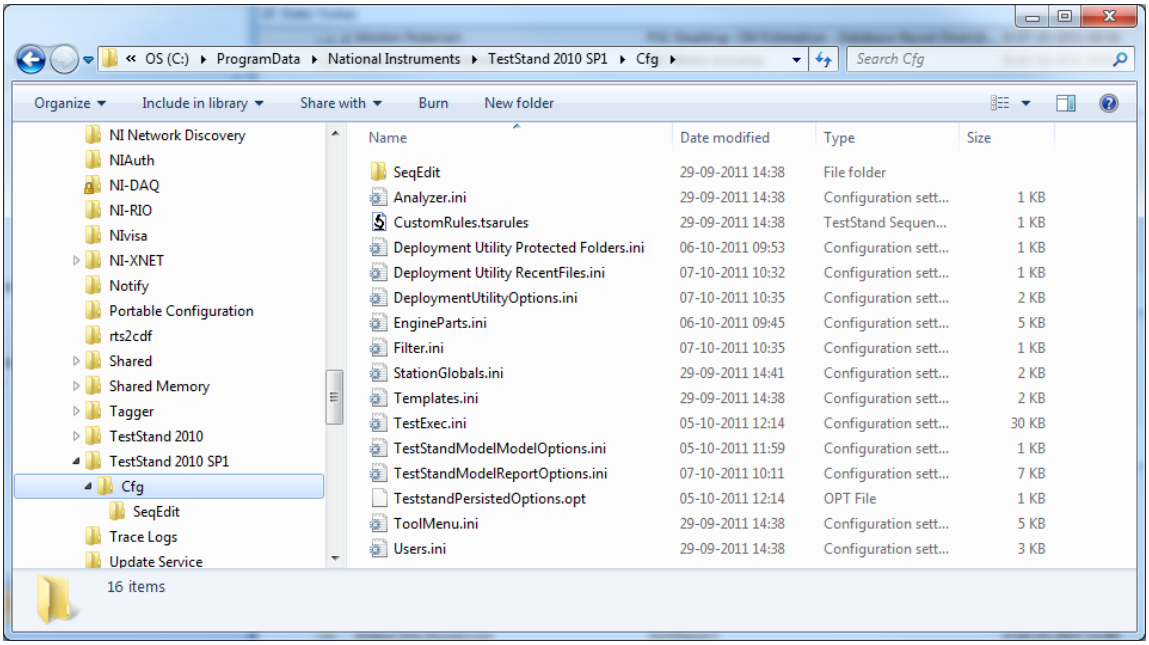


Figure 29: cfg settings on a Windows 7 PC.

9.4 FlexStand OI Run-Time

Install the FlexStand Operator Interface Run-Time on the Run-Time PC. The FlexStand OI Run-Time can be downloaded from www.flexstand.eu.

Start FlexStand and check that the simple example starts. The Simple example is provided as a default Operator Interface in the FlexStand Run-Time installation.

Activate licenses if requested or select the evaluation option.

9.5 FlexStand Operator Interface and Sequence files

Note: Before copying any LabVIEW files to the Run-Time PC all LabVIEW code must be mass compiled. This is necessary because the Run-Time PC only has a LabVIEW Run-Time installed.

Copy the mass compiled files from the <TestStandPublic>\Userinterfaces folder to the Run-Time PC at the same locations.

Depending on the LabVIEW code, components from the LabVIEW vi.lib, instr.lib or other LabVIEW basic components may also need to be copied to the Run-Time PC.

9.6 Configuring FlexStand

On the Development PC the FlexStand OI can be configured using the Configuration Tool. The configuration tool can be found on the desktop or in the Start menu:
All Programs->CIM->FlexStand OI->FlexStand OI Config Tool.
It is highly recommended to use the configuration tool for making modifications to the FlexStand OI environment but the files can also be modified in a text editor like notepad.

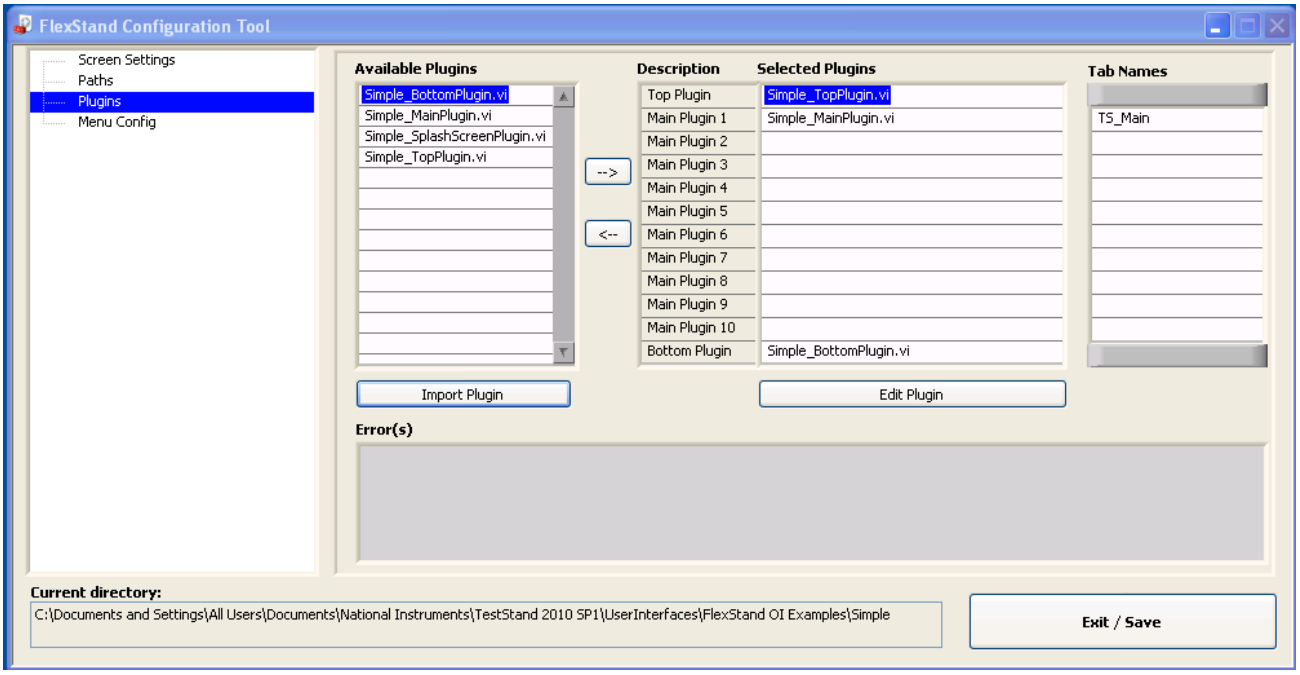


Figure 30: The FlexStand OI Configuration Tool.

In order for the Run-Time PC to execute your Operator Interface you need to copy the FlexStandOI.ini file to the Run-Time PC or modify the local version of the file on the Run-Time PC.
'FlexStandOI.ini' is located in the folder: <TestStandPublic>\UserInterfaces\FlexStand OI.

All other components should already have been copied as described in section 9.5.

9.7 Shared variables in a TestStand project

If you use shared variables you must ensure that library is deployed before you use it. This can be done from TestStand using the Deploy Library steptype (located in the LabVIEW Utility group). The Deploy Library step can for example be added to the FrontEnd Callback to ensure that the library is loaded before you use it in FlexStand or in your sequences. For more information refer to the TestStand help and search for Deploy Library.

10 FlexStand Command Line Switches

FlexStand_OI.exe supports command line switches. This is often used for desktop shortcuts or entries in the Start menu.

Some typical usages for command line switches are:

- Automatic load of a sequence
- Automatic load and execution of a sequence
- Automatic change between different operator interfaces
- Custom arguments

FlexStand supports the same switches as TestStand but also supports custom designed switches for control of the user interface.
Command line switches provided by TestStand are processed by TestStand while custom switches are available to the programmer of the FlexStand Operator Interface by using the FlexStand API (Get Command Line Arguments).

Some typical examples of using command line switches are:

To load a sequence:

FlexStand_OI.exe c:\mysequence.seq

To load a sequence and run the main sequence:

FlexStand_OI.exe –runEntryPoint Mainsequence c:\mysequence.seq

To pass a user-defined switch to a FlexStand callback (e.g. CIM_TS_OI CALLBACK ReadCustomSetup.vi):

FlexStand_OI.exe -myargument

(The argument can be read by calling the FlexStand API function 'Get Command Line Arguments')

Note: You can use / or - to specify a switch. Quotation marks are required for arguments that contain a space, such as "Test UUTs" and "C:\My Documents\Test Sequence.seq". Depending on the operating system there may be some limitations to the length of the entire command string.

More information about command line switches supported by TestStand can be found in the TestStand help by searching for "CommandLineArguments".

11 Upgrade from old versions

If you already have a FlexStand OI license you can upgrade to the latest version at a reduced cost. Please send an e-mail with your current license number to support.flexstand@cim.as and please indicate the version number of the FlexStand OI you want to upgrade to.

12 Support

For support information, refer to www.flexstand.eu for access to Support, FAQ, download and additional information about FlexStand.

Registered users may access their account by logging in on www.flexstand.eu. When logged into your account you will have access to your customer information and lists of licenses and activations.

For support available at CIM Industrial Systems offices, please send an email to support.flexstand@cim.as.

You may also contact CIM Industrial Systems directly at:

CIM Industrial Systems
Jegstrupvej 96a
DK-8361 Hasselager
Denmark

Phone: +45 96 84 05 00
e-mail: support.flexstand@cim.as