

# **Modular Test Sequencer MTS**

---

**Operating Manual**  
Rev. 2.3

**Trademarks**

LabVIEW is a registered trademark of National Instruments.

**Notice**

The information in this document is subject to change without notice and should not be construed as a commitment by PI Electronics AG. PI Electronics AG assumes no responsibility for any errors that may appear in this document.

In no event shall PI Electronics AG be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall PI Electronics AG be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without PI Electronics AG's written permission, and the contents thereof must not be imparted to a third party nor be used for any unauthorized purpose.

*Copyright © PI Electronics AG 2024*

**Document information:**

Document: PI506-MTS\_Operating-Manual\_EN.docx  
 Date: 2024/11/22  
 Author: Ramon Dätwyler  
 Revision: 2.3

**Revision record sheet:**

Revision	Date	Change, issued by	Description
1.0	2008/12/17	R. Dätwyler	Operating manual for MTS version 1.0.0 - 1.0.2
1.1	2009/02/04	R. Dätwyler	Operating manual for MTS version 1.0.3
1.2	2009/04/22	R. Dätwyler	Operating manual for MTS version 1.0.4
1.3	2009/10/14	R. Dätwyler	Operating manual for MTS version 1.0.5 - 1.1.1
1.4	2010/09/02	R. Dätwyler	Operating manual for MTS version 1.1.2 - 1.1.6
1.5	2011/09/02	R. Dätwyler	Operating manual for MTS version 1.1.7
1.6	2011/09/08	R. Dätwyler	Operating manual for MTS version 1.1.8
1.7	2012/03/21	R. Dätwyler	Operating manual for MTS version 1.1.9
2.0	2019/04/18	R. Dätwyler	Operating manual for MTS version 2.0.0 - 2.0.2
2.1	2022/12/16	R. Dätwyler	Operating manual for MTS version 2.0.3 - 2.0.4
2.2	2023/09/12	R. Dätwyler	Operating manual for MTS version 2.0.5 – 2.0.7
2.3	2024/11/22	R. Dätwyler	Operating manual for MTS version 2.1.0 or later

**Contents:**

1	Terms .....	7
2	Introduction .....	8
3	Software operation .....	9
3.1	Installation .....	9
3.2	Program Start .....	9
3.3	License Manager .....	9
3.3.1	License types .....	9
3.3.2	Development License .....	9
3.3.3	Runtime License .....	9
3.4	Login .....	10
3.5	User Management .....	11
3.5.1	Privilege levels .....	11
3.5.2	Start User Manager .....	11
3.5.3	User Groups .....	11
3.5.4	User Accounts .....	12
3.6	Main Panel .....	13
3.7	Preconditions .....	14
3.8	New project/sequence .....	14
3.9	Save/Save as... sequence .....	14
3.10	Open project/sequence .....	14
3.11	Sequence File Revisions .....	14
3.12	Project structure .....	15
3.13	Modules .....	16
3.14	Miscellaneous .....	16
3.15	Variables .....	16
3.15.1	Variable data types .....	16
3.15.2	System variables .....	17
3.15.3	User variables .....	17
3.15.4	Set variable value .....	18
3.16	Sequence Table .....	18
3.16.1	Table Columns .....	18
3.16.2	Main Sections .....	19
3.17	Sequence Compilation .....	19
3.18	Step Configuration .....	19
3.18.1	Common items .....	19
3.18.2	Module .....	20
3.18.3	Section .....	22
3.18.4	Variable .....	23
3.18.4.1	Formulas .....	24
3.18.4.1.1	Numbers .....	24
3.18.4.1.2	Variables .....	24
3.18.4.1.3	Operators .....	24
3.18.4.1.4	Functions .....	25
3.18.4.1.5	Argument separator .....	25
3.18.5	Loop .....	26
3.19	Test Operation .....	27
3.19.1	Start .....	27
3.19.2	Step .....	27
3.19.3	Stop .....	27
3.19.4	Results Table .....	27
3.20	Commissioning and debugging options .....	28
3.20.1	Debugging log .....	28
3.20.2	Breakpoints .....	28
3.20.3	Debugging steps .....	29
3.20.4	Slow down execution speed .....	29
3.20.5	Force sequence to continue in any case .....	29
3.21	Result Files .....	30

3.21.1	File storage .....	30
3.21.2	Format .....	30
3.21.3	Edit header / footer .....	30
3.22	Program options .....	31
3.23	Program exit .....	31
3.24	Program shortcuts .....	32
4	Appendix .....	33
4.1	Guidelines for LabVIEW modules .....	33
4.1.1	Using the variable Step.Attachment .....	33
4.2	Supported interface data types for LabVIEW modules .....	33
4.2.1	Boolean (BOOL) .....	33
4.2.2	Signed Integer 64 Bit (I64) .....	33
4.2.3	Double Precision Floating-Point Number (DBL) .....	33
4.2.4	String (STR) .....	34
4.3	LabVIEW Format Specifiers .....	35
4.3.1	Format Specifiers Syntax Elements .....	35
4.3.1.1	Format Codes for the Time Format String .....	36
4.3.2	Format Specifier Examples .....	37
4.4	How to create a Source Distribution or a Packed Library? .....	38
4.4.1	Source Distribution .....	38
4.4.2	Packed Library .....	40
4.5	MTS application files .....	45
4.6	Report Example .....	46

**List of figures:**

Figure 1:	System overview.....	8
Figure 2:	License Manager Window .....	9
Figure 3:	Login .....	10
Figure 4:	Change Password .....	10
Figure 5:	User Manager main panel .....	11
Figure 6:	Group overview & Create new group .....	12
Figure 7:	Create new user .....	12
Figure 8:	Front Panel .....	13
Figure 9:	New Test Sequence .....	14
Figure 10:	Project structure .....	15
Figure 11:	Add Variable .....	17
Figure 12:	Overwrite a variable value .....	18
Figure 13:	Sequence table.....	18
Figure 14:	Common configuration items.....	19
Figure 15:	Module configuration: General .....	20
Figure 16:	Module configuration: Pre Action .....	20
Figure 17:	Module configuration: Module controls.....	21
Figure 18:	Module configuration: Post Action.....	21
Figure 19:	Module configuration: Evaluation .....	22
Figure 20:	Variable configuration.....	23
Figure 21:	Formula expression .....	23
Figure 22:	Loop configuration .....	26
Figure 23:	Results Table.....	28
Figure 24:	Debugging log field.....	28
Figure 25:	Set/clear breakpoint.....	28
Figure 26:	Select steps to run in debugging mode .....	29
Figure 27:	Flag to temporary ignore errors or failed status .....	29
Figure 28:	Header/Footer definition .....	31
Figure 29:	Options dialog.....	31
Figure 30:	New Source Distribution .....	38
Figure 31:	Distribution Settings.....	38
Figure 32:	Additional Exclusions .....	39
Figure 33:	Test modules arranged as a project library .....	40
Figure 34:	Create a new Packed Library .....	40
Figure 35:	Packed Library naming and destination paths .....	41
Figure 36:	Packed Library source files selection .....	41
Figure 37:	Packed Library build destination path .....	42
Figure 38:	Packed Library source file settings.....	42
Figure 39:	Packed Library advanced build options .....	43
Figure 40:	Packed Library file exclusions .....	43
Figure 41:	Packed Library pane compatibility.....	44
Figure 42:	Packed Library version definition .....	44
Figure 43:	Packed Library Pre/Post build actions.....	45

## 1 Terms

<i>Term</i>	<i>Description</i>
MTS	<b>M</b> odular <b>T</b> est <b>S</b> equencer
UUT	<b>U</b> nit <b>U</b> nder <b>T</b> est
VI	<b>V</b> irtual <b>I</b> nstrument (LabVIEW function)
LVLIB	Labview library file
LVLIBP	Packed LabVIEW library containing all member VI's of the library
EOL	<b>E</b> nd- <b>o</b> f- <b>l</b> ine
OS	<b>O</b> perating <b>S</b> ystem

## 2 Introduction

The Modular Test Sequencer is a framework to create and run test sequences as well as to protocol the test results in a given, table-oriented format. The test sequence can be compiled by drag-and-dropping items from the selection boxes. Each test step within the sequence consists of either a LabVIEW VI (module) or another functional item. The step can be individually configured according to its type. A variable engine allows creating user defined variables in order to pass data from one step to another.

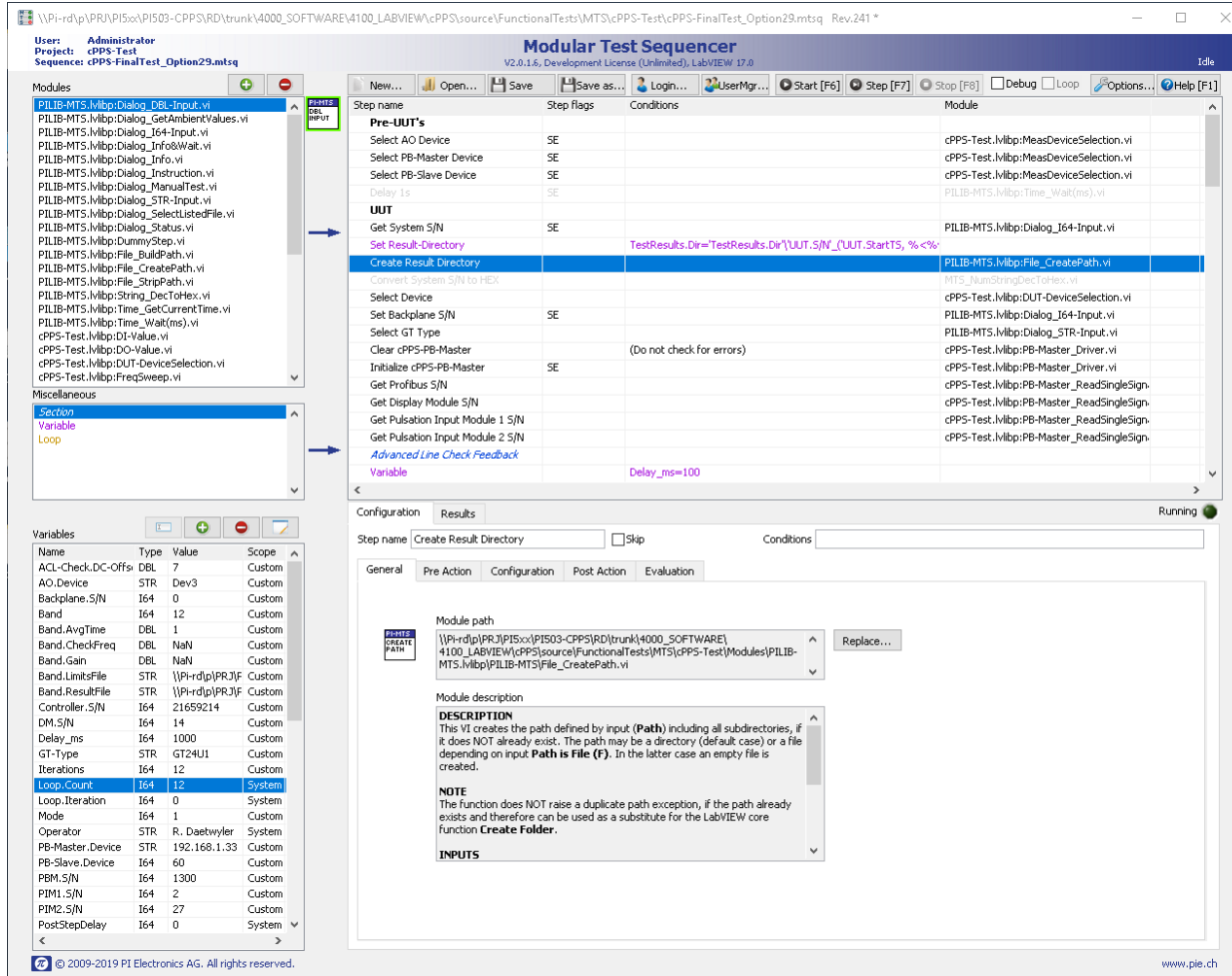


Figure 1: System overview

## 3 Software operation

### 3.1 Installation

To install the Modular Test Sequencer software run the executable “setup.exe” from the MTS Installation media and follow the instructions on the screen.

### 3.2 Program Start

Start the program by using the link PI Electronics AG -> Modular Test Sequencer -> Modular Test Sequencer in the Windows Start Menu. If you start the software for the first time it's necessary to activate the program by entering a valid license code.

### 3.3 License Manager

To protect the software for illegal copies a license check is active. If the program is not activated yet the License Manager pops up:

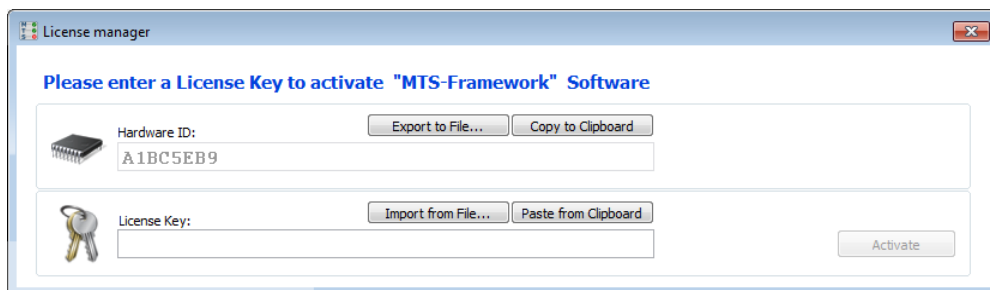


Figure 2: License Manager Window

The ‘Hardware ID’: shows a unique ID of your computer system. To obtain a valid license key please send this 8-digit hex number to PI Electronics AG (E-mail: [pisales@pie.ch](mailto:pisales@pie.ch)). If you are an authorized user, PI Electronics AG will return you a valid license key for this computer. Optionally and to avoid typing errors you can save the ‘Hardware ID’ to an ASCII file by using the button ‘Export to File...’ and send the generated file to PI Electronics AG.

After you received a valid license key enter the numbers into the mask or import it from a file by pressing ‘Import from File...’. Then press ‘Activate’ to license the product.

#### 3.3.1 License types

MTS software includes two license types with different functional scope: Development License and Runtime License. The type of license which is activated and the license expiration date is shown in the header bar of the main window.

#### 3.3.2 Development License

Development License includes the full range of functionality, i.e. create, run, edit and debug of a MTS project.

#### 3.3.3 Runtime License

Runtime License only allows the user to run a sequence of given MTS project.

### 3.4 Login

After the program start, the operator is prompted to enter the user name and password. After the installation, the two system user accounts 'Administrator' (full access) and 'Guest' (minimal access) are available (Default passwords: ""). Additional user accounts and passwords can be set up in the User Manager (chapter 3.5) by a user with administrator privileges.

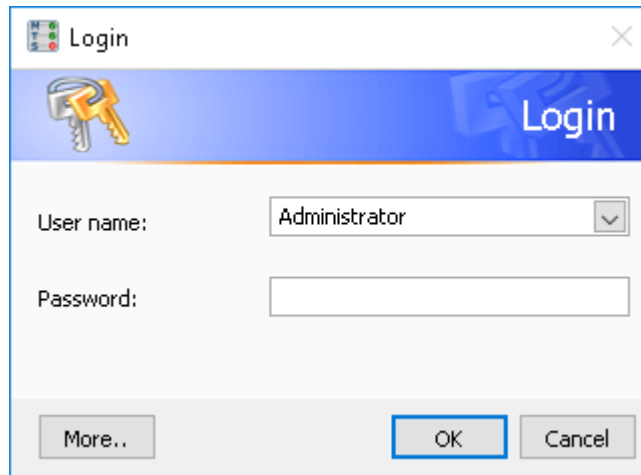


Figure 3: *Login*

To change a password press button 'More...' and enter your old and new passwords:

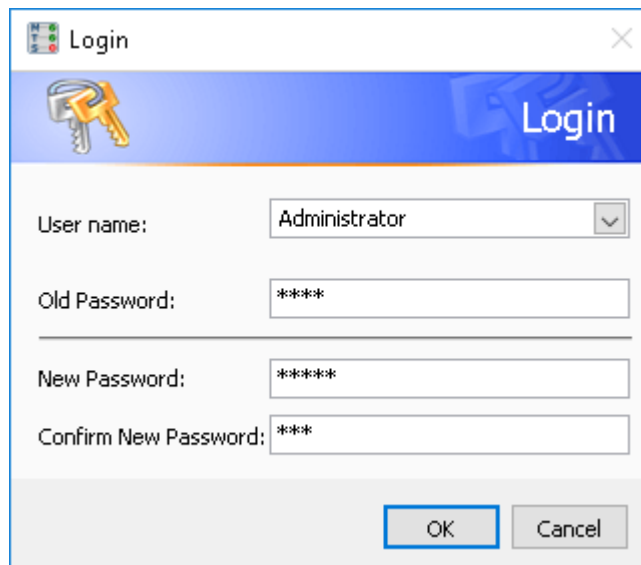


Figure 4: *Change Password*

## 3.5 User Management

### 3.5.1 Privilege levels

The user management distinguishes between different levels of privileges. Each user belongs to a group with a privilege level in the range of 1...10. Members of groups of level 1 have the least privileges, those of level 10 the most. The privilege level controls the user access to certain parts of the program:

Level	10	9	8	7	6	5	4	3	2	1
Change user accounts	X									
Create new sequence/project	X	X	X	X	X					
Open sequence	X	X	X	X	X	X	X	X	X	X
Save sequence	X	X	X	X	X					
Edit sequence	X	X	X	X	X					
Configure steps	X	X	X	X	X					
Add/Remove modules	X	X	X	X	X					
Add/Remove variables	X	X	X	X	X					
Edit Header/Footer	X	X	X	X	X					
Start sequence	X	X	X	X	X	X	X	X	X	X
Stop sequence	X	X	X	X	X	X	X	X	X	X
Step sequence	X	X	X	X	X					
View Results	X	X	X	X	X	X	X	X	X	X
Open Help	X	X	X	X	X	X	X	X	X	X
Change options	X	X	X	X	X					
Breakpoint/Debugging options	X	X	X	X	X					

Remark: If a user cancels the login procedure, then level 0 is applied and all parts of the program are inaccessible.

### 3.5.2 Start User Manager

To open the user manager, press button 'UserMgr...' on the front panel. Since this part of the program requires administrator privileges (Level 10) you are prompted to login as user of a group of this level, unless you already logged in.

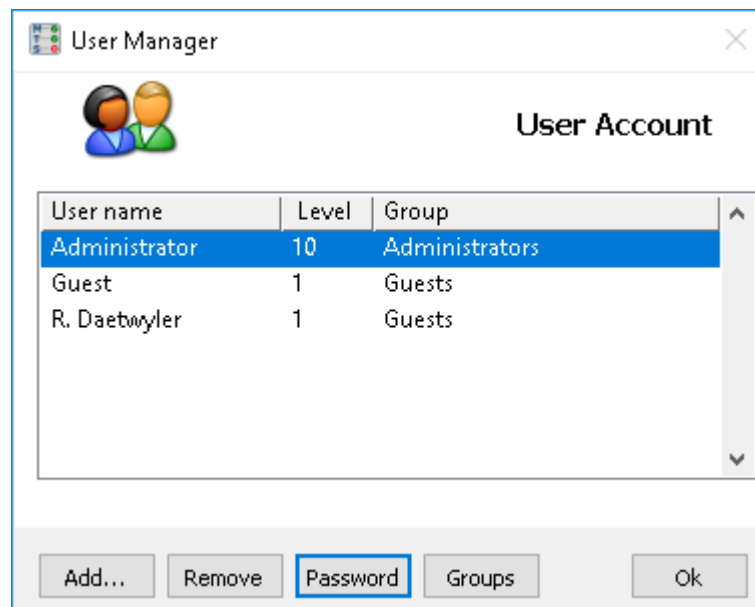


Figure 5: *User Manager main panel*

### 3.5.3 User Groups

Before setting up new user profiles the user groups must be defined. Press the button 'Groups' to open the group editor.

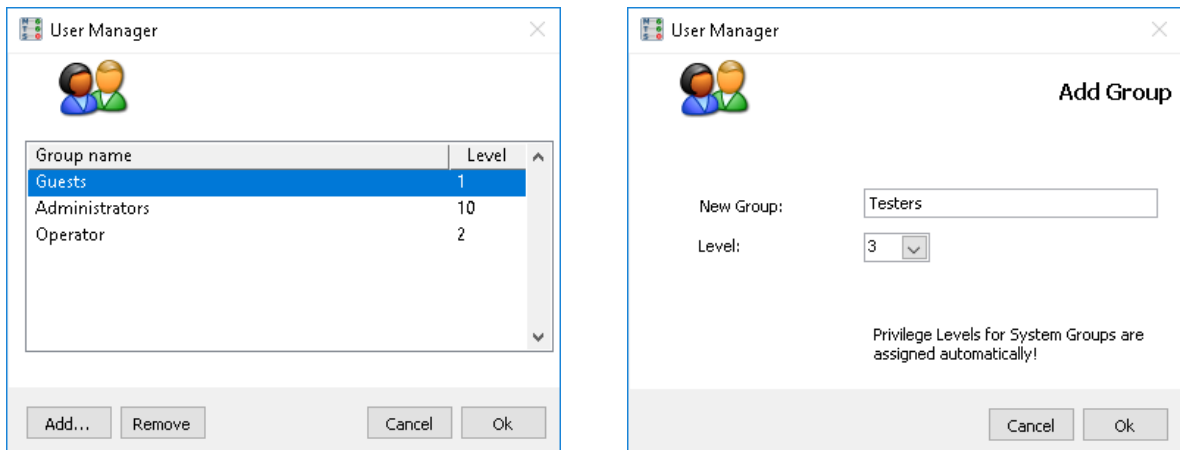


Figure 6: *Group overview & Create new group*

Press button 'Add...' to create a new group and assign a privilege level to the group. To delete a group, press 'Remove'.

Remark: Group Administrators & Guests are system groups and cannot be deleted. Press 'Ok' to return to the User Manager Main Panel.

### 3.5.4 User Accounts

After all groups are set up, new users can be created by pressing 'Add...'. Enter a new user name and assign it to a group. Then enter and confirm a password for this user. After creating all users press 'Ok' to return to the User Manager main panel.

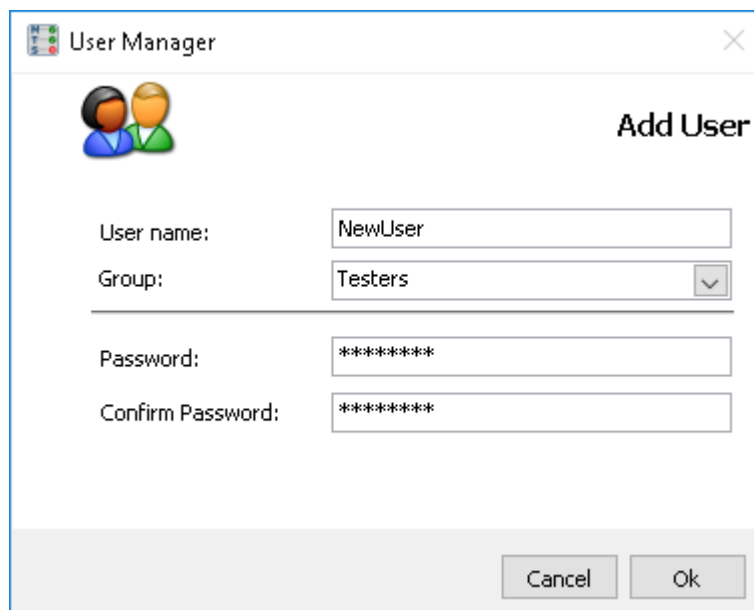


Figure 7: *Create new user*

To edit the group or password of an existing user press button 'Password' in the User Manager main panel and change the settings for that user.

### 3.6 Main Panel

Both test sequence configuration and execution can be done from the following user interface:

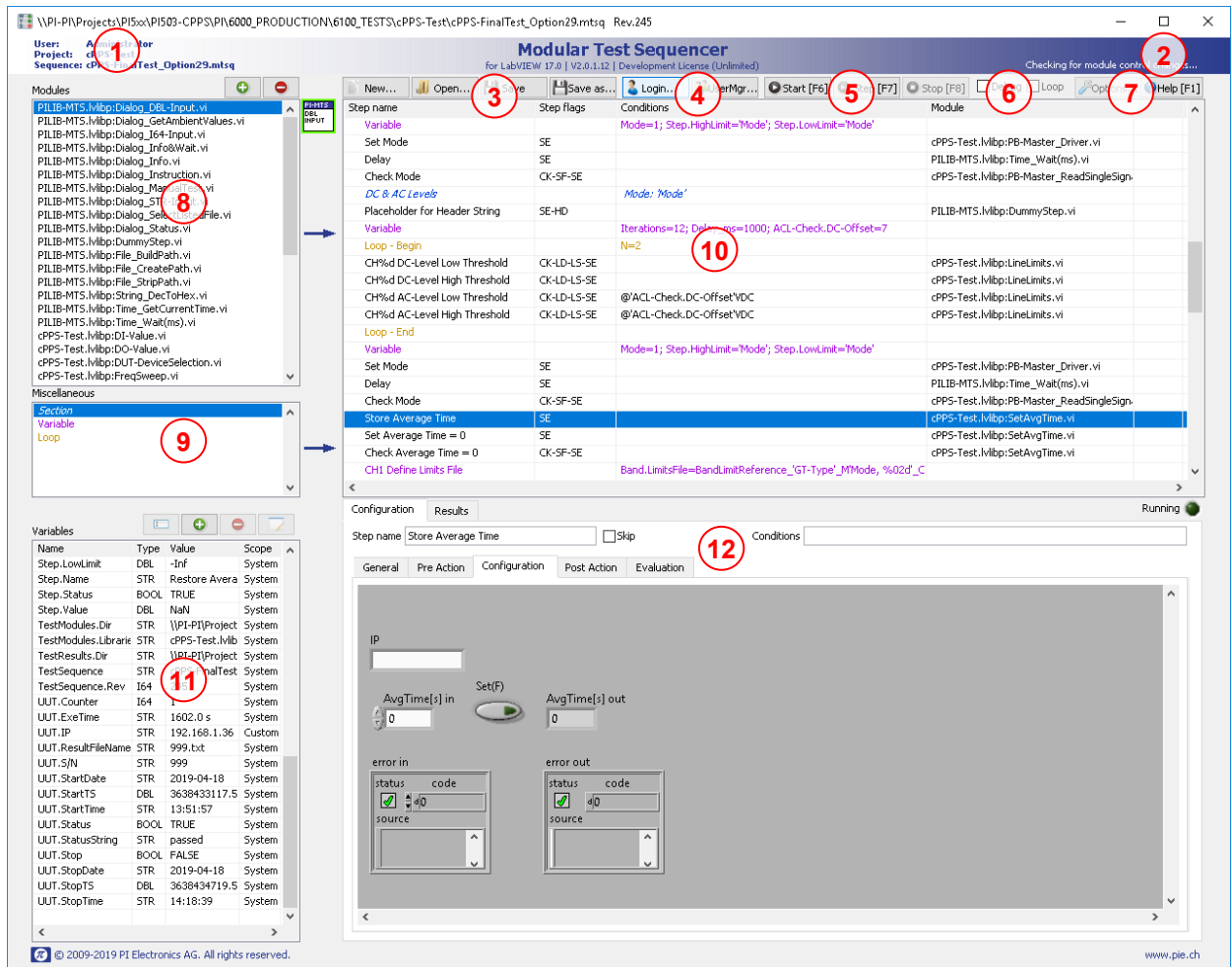


Figure 8: Front Panel

1	Test Info: Shows the current user name, project name and sequence name
2	Status Info: Shows the current state of the sequencer
3	File Handling: Buttons to open/save sequences and to create new sequences
4	User Management: Buttons to log in or user management
5	Run Controls: Buttons to run, step or stop the test sequence
6	Checkboxes for debugging during commissioning
7	Program options and help
8	Modules: Shows the available LabVIEW modules
9	Miscellaneous: Shows the available non-module step types
10	Sequence: Table to compile the test sequence
11	Variables: Shows the available system and user variables
12	Configuration/Results: Panel to configure a single test step or to show test results

### 3.7 Preconditions

Before creating or executing a test sequence, the following preparations must have been made:

- The modules are created and distributed (see chapter 4.4)
- All necessary drivers for the above modules are installed (e.g. DAQmx, VISA, ...)

### 3.8 New project/sequence

To create a new project or sequence press button 'New...' (Shortcut: Ctrl+N). The following window will pop up:

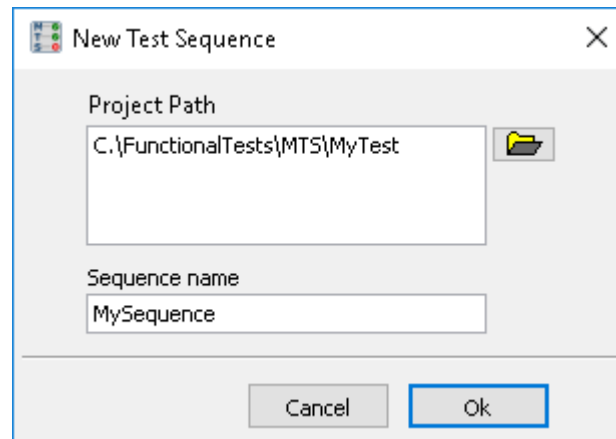


Figure 9: *New Test Sequence*

Use the Browse-Button to select or create a project folder and define a name for the test sequence. This will create a new project structure (if not existing) and an empty test sequence within the project.

### 3.9 Save/Save as... sequence

To save the current test sequence, press 'Save' (Shortcut: Ctrl+S). To save a copy of the sequence in the project press 'Save as...' and enter a new name.

### 3.10 Open project/sequence

To open an existing sequence, press 'Open...' (Shortcut: Ctrl+O) and navigate to the desired sequence. The MTS framework will automatically switch to the navigated project and load its modules into memory.

### 3.11 Sequence File Revisions

Each sequence file is provided with an internal revision number in order to exactly identify the version of the file. This is necessary since changes to the sequence might be made even after series of units have already been tested, while the file name of the test sequence remains the same. When you create a new sequence, the revision will be 1 and increases by 1 each time modifications are saved. The revision of the file is visible in the title bar following the file name. The revision information is also available as a predefined variable called 'TestSequence.Rev', which should be included in the header/footer of each Test Report (see 3.21.3).

### 3.12 Project structure

When a new project is initiated, a project folder (project name) and the two subfolders 'Modules' and 'Results' are created. The test sequences are stored in the project directory. The folder 'Modules' is used to store the test modules from a LabVIEW distribution. The test results (e.g. TAB separated ASCII file) are stored in the subfolder which is named after the sequence name in the 'Results' directory. Alongside a copy of the current sequence file is saved, in order to retain the dedicated version which produced the result files. Thus, the file name of the copy is supplemented with the revision of the sequence file <sequence name>\_Rev<revision>.mts. If the test was running with unsaved sequence changes (which should be avoided) the name gets an additional 'm' following the revision number <sequence name>\_Rev<revision>m.mts.

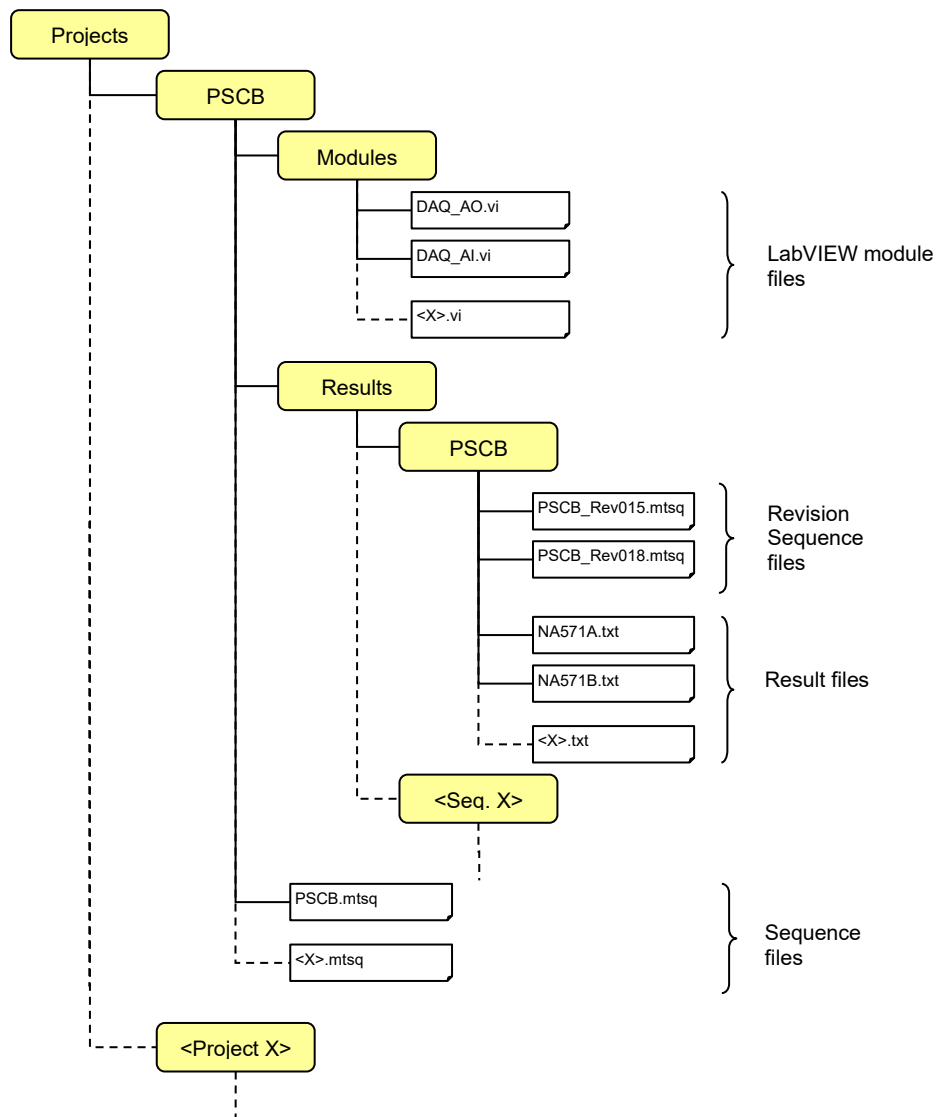


Figure 10: Project structure

### 3.13 Modules

There are two possibilities to add LabVIEW functions from the distribution to the 'Modules' selection box (see chapter 4.4 for more information about creation of a source distribution or packed library):

- a) Copy the VI's and packed libraries (\*.lvlibp) from the distribution manually to the project's 'Modules' folder. Then reopen the sequence file to load the copied modules into the memory by using 'Open...'
- b) Press (+) to browse to the distribution and select all the VI's in the folder or select a packed library (\*.lvlibp). The modules are automatically copied to the project and loaded into memory.

To remove a module from the 'Modules' selection box press (-). The file will be deleted in the project's 'Modules' folder (not possible with modules inside packed libraries).

### 3.14 Miscellaneous

Beside the module steps there are some other step types which can be inserted into the sequence. These are used to structure the test sequence, to control the sequence execution or for reporting reasons:

Type	Description
Section	Opens a new section in the test sequence. All steps below this step down to the next section step belong to this section (members). It adds an additional row to the report which shows the overall status (passed/failed) of all the member steps.
Variable	Possibility to assign a value, another variable or a formula to a variable.
Loop	Possibility to repeatedly execute parts of the sequence for a defined number of iterations.

### 3.15 Variables

In order to pass data from one step to another or to control the sequence execution flow there's a variable engine available. The available variables are listed in the 'Variables' list box. Besides a set of predefined system variables, the user has the possibility to create its own variables.

#### 3.15.1 Variable data types

The variable engine supports 4 data types of variables:

Type	Description	Range
BOOL	Boolean	T/F
I64	Signed Integer 64Bit	$-2^{32} \dots 2^{32}-1$
DBL	Double Floating-Point Number	Sign: 1Bit, Exponent: 11Bit, Mantissa: 52Bit
STR	String	-

### 3.15.2 System variables

The variable engine provides a set of predefined system variables which can be accessed in each step:

Name	Type	Description
Step.Name	STR	Name of the current step
Step.Value	DBL	Value of the current step
Step.HighLimit	DBL	High Limit parameter which can be used for result evaluation
Step.LowLimit	DBL	Low Limit parameter which can be used for result evaluation
Step.Status	BOOL	Status of the current step (TRUE=passed, FALSE=failed)
Step.Attachment	STR	Report attachment of the current step
Section.Status	BOOL	Status of the current section (TRUE=passed, FALSE=failed)
Section.Stop	BOOL	Flag to exit current section and proceed to the next step after it
UUT.Status	BOOL	Status of the current UUT (TRUE=passed, FALSE=failed)
UUT.StatusString	STR	UUT.Status as passed/failed string
UUT.Stop	BOOL	Flag to exit current unit test and proceed to the next UUT
UUT.Counter	I64	Number of currently tested UUT's
UUT.S/N	STR	Serial number of the current UUT
UUT.StartDate	STR	Date of the Test-Begin for the current UUT
UUT.StartTime	STR	Time of the Test-Begin for the current UUT
UUT.StopDate	STR	Date of the Test-End for the current UUT
UUT.StopTime	STR	Time of the Test-End for the current UUT
UUT.ExeTime	STR	Test execution time in seconds for the current UUT
UUT.ResultFileName	STR	Name of the result file
Operator	STR	Name of the logged-in user
StationID	STR	Name of the current computer
Project.Dir	STR	Directory path of the sequence
TestResults.Dir	STR	Directory path where results are stored
TestSequence	STR	Name of the current test sequence
TestSequence.Rev	I64	Revision of the current test sequence
TestSequence.Stop	BOOL	Flag to exit test sequence and proceed to the Post-UUT's steps
TestModules.Dir	STR	Directory path of the modules
TestModules.Libraries	STR	List of loaded libraries with versions (comma separated)
Loop.Count	I64	Number of total iterations for the current loop
Loop.Iteration	I64	Iteration index of the current loop
Loop.Iteration.Stop	BOOL	Flag to exit current loop iteration and proceed to the next iteration
Loop.Stop	BOOL	Flag to exit current loop structure and proceed to the next step after it
PostStepDelay	I64	Global wait time in milliseconds after a module has been executed

### 3.15.3 User variables

The variable engine allows the user to create its own variables which can be accessed in each step. A new variable can be created using the button (+) or double-click on an empty row:

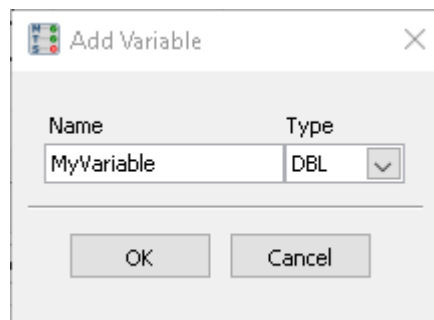


Figure 11: Add Variable

Enter the name for the variable and select its type, then press 'OK'.  
 To remove a variable from the list, select the variable and press (-) or hit 'Delete'.  
 To change a variable, select the variable and press the 'Edit' button or hit 'Space'.

**Note:** System variables cannot be deleted nor changed.

### 3.15.4 Set variable value

The value of the variable are updated during sequence execution. However, for sequence commissioning and debugging it might be necessary to manually set a variable to a certain value. This can be done by pressing button 'Overwrite value':

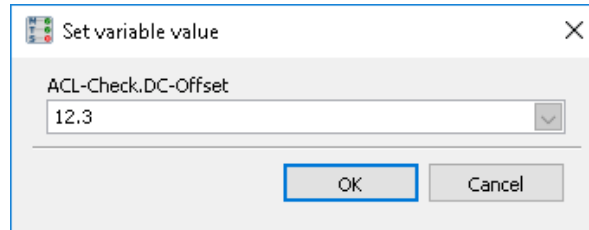


Figure 12: *Overwrite a variable value*

Enter the new value for the variable, then press 'OK'.

### 3.16 Sequence Table

The sequence table is the place where the different test steps are compiled to a sequence. The color of a step indicates its type: Module steps are black, Section steps italic blue, Variable steps magenta and Loop structures brown. Skipped steps appear in grey color.

Step name	Step flags	Conditions	Module
Variable		Mode=1; Step.HighLimit='Mode'; Step.LowLimit='Mode'	
Set Mode	SE		cPPS-Test.Ivlbp:PB-Master_Driver.vi
Delay	SE		PILIB-MTS.Ivlbp:Time_Wait(ms).vi
Check Mode	CK-SF-SE		cPPS-Test.Ivlbp:PB-Master_ReadSingleSign.
<i>DC &amp; AC Levels</i>		Mode: 'Mode'	
Placeholder for Header String	SE-HD		PILIB-MTS.Ivlbp:DummyStep.vi
Variable		Iterations=12; Delay_ms=1000; ACL-Check.DC-Offset=7	
Loop - Begin		N=2	
CH%d DC-Level Low Threshold	CK-LD-LS-SE		cPPS-Test.Ivlbp:LineLimits.vi
CH%d DC-Level High Threshold	CK-LD-LS-SE		cPPS-Test.Ivlbp:LineLimits.vi
CH%d AC-Level Low Threshold	CK-LD-LS-SE	@'ACL-Check.DC-Offset'\VDC	cPPS-Test.Ivlbp:LineLimits.vi
CH%d AC-Level High Threshold	CK-LD-LS-SE	@'ACL-Check.DC-Offset'\VDC	cPPS-Test.Ivlbp:LineLimits.vi
Loop - End			
Variable		Mode=1; Step.HighLimit='Mode'; Step.LowLimit='Mode'	
Set Mode	SE		cPPS-Test.Ivlbp:PB-Master_Driver.vi
Delay	SE		PILIB-MTS.Ivlbp:Time_Wait(ms).vi
Check Mode	CK-SF-SE		cPPS-Test.Ivlbp:PB-Master_ReadSingleSign.

Figure 13: *Sequence table*

#### 3.16.1 Table Columns

The table is divided into the four columns Step name, Step flags, Conditions and Module:

- Step name: Shows the name of the step, which can be modified in the step configuration. Exception: Loop and main section step types.
- Step flags: Shows the settings of the result logging and error handling flags for module step type.

Shortcut	Flag
CK	Check limits
LD	Log data
LS	Log result status
SF	Stop action defined, if test failed
SE	Stop action defined, if the module reports an error
HD	Add column header

- Conditions: Shows the 'Conditions' user text for module step types, the 'Loop Count' for loop step types or the variable value assignments for variable step types.
- Module: Shows the module file name for module step types.

### 3.16.2 Main Sections

Every test sequence is structured in 3 main sections:

- Pre-UUT's: Steps in this section are executed once only at the beginning of the test.
- UUT: This is the core part of the sequence. Steps in this section are repeatedly executed for each UUT. After the last step in the UUT section has been executed it will automatically jump back to the first step in this section. So, ensure that the first step in this section is a dialog step, which allows the user to install the next UUT.
- Post-UUT's: Steps in this section are executed once only at the end of the test (after all UUT's have been tested). The end of the test can be induced by a user stop command (variable TestSequence.Stop=TRUE), a module error or a failed test step.

### 3.17 Sequence Compilation

To insert a new step to the sequence simply drag an item from the 'Modules' or the 'Miscellaneous' selection box and drop it in the sequence table. Then drag the inserted step to the correct position within the test sequence and configure it.

### 3.18 Step Configuration

Depending on the step type, there are different editors to configure the step. However, some configuration items are common for all step types. Select the 'Configuration' tab of the Configuration/Results control.

#### 3.18.1 Common items

The following configuration items are common for all step types:

Configuration item	Description	Default value
Step name	Name of the current step. It will be added as test name in the report. You can use %d to include the iteration index into the name, if the step is placed in a loop structure. Put the text into quotation characters with optional format specifier ('<Variable> [, <format specifier>'] if you want to include a value from a variable.	""
Skip	If Skip is TRUE, the step will not be executed.	OFF
Conditions	Allows the user to compose additional information for the test, which will also appear in the report (not available for Variable or Loop step types). Put the text into quotation characters with optional format specifier ('<Variable> [, <format specifier>'] if you want to include a value from a variable.	""

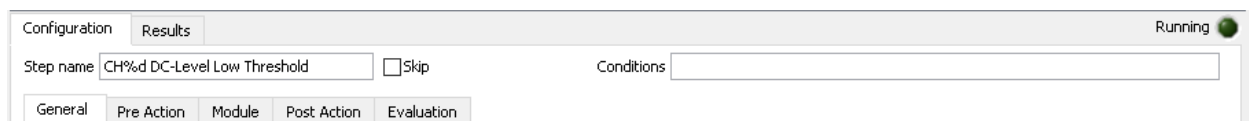


Figure 14: Common configuration items

### 3.18.2 Module

*General:* Shows the full path, icon and description of the module. If you want to substitute the module for this step press 'Replace...' and select the new module. The framework tries to reassign configuration values from the former module by cross-checking label names and types. However, always check the control values of the module as described below when exchanging a module. Checkbox 'Show FP on testing' offers the option to display the front panel of the module while testing.

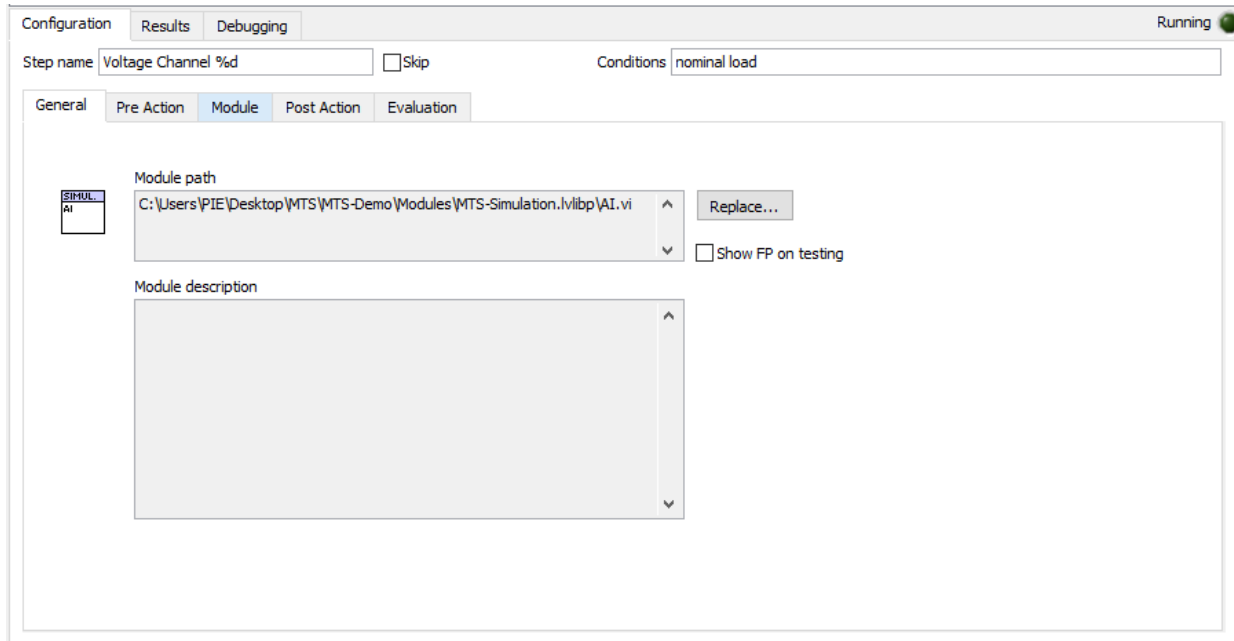


Figure 15: *Module configuration: General*

*Pre Action:* Value assignment from variables to module controls **before** the module is executed. Use the corresponding button to add ('Insert'), remove ('Delete') or edit ('Space') assignments.

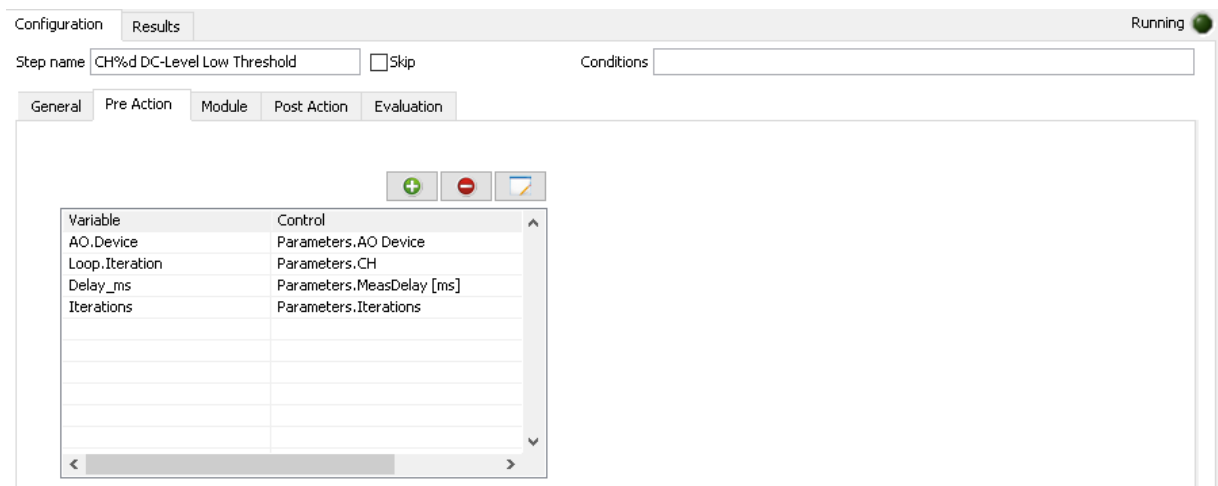


Figure 16: *Module configuration: Pre Action*

**Note:** Depending on the type of the selected variable only type-compatible controls are listed in the control selection box.

**Module:** Change the controls directly on the module's front panel. The module is executed with these control settings unless a certain control is changed by a Pre-Action.

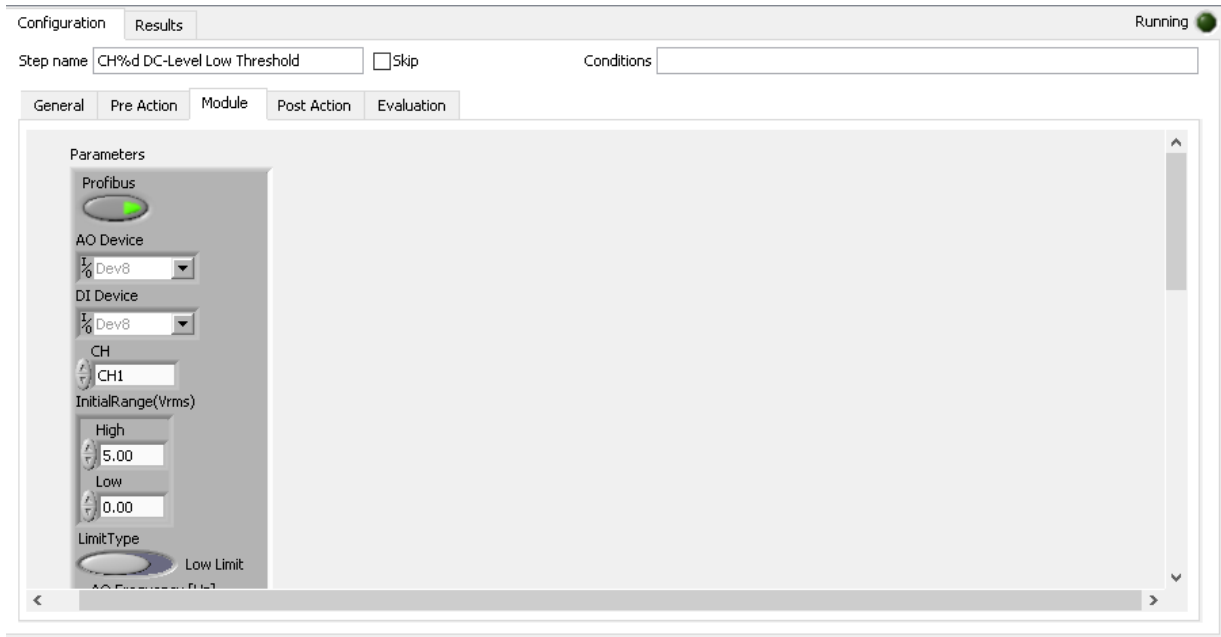


Figure 17: *Module configuration: Module controls*

**Post Action:** Value assignment from module indicators to variables **after** the module has been executed. Use the corresponding button to add ('Insert' key), remove ('Delete' key) or edit ('Space' key) assignments. Consider: Depending on the type of the selected indicator only type-compatible variables are listed in the variable selection box.

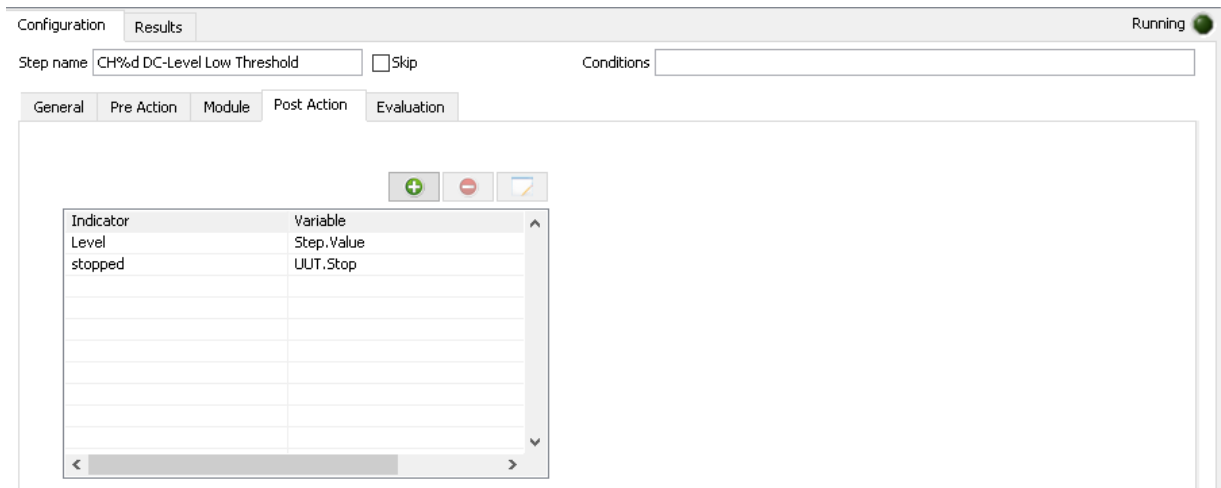
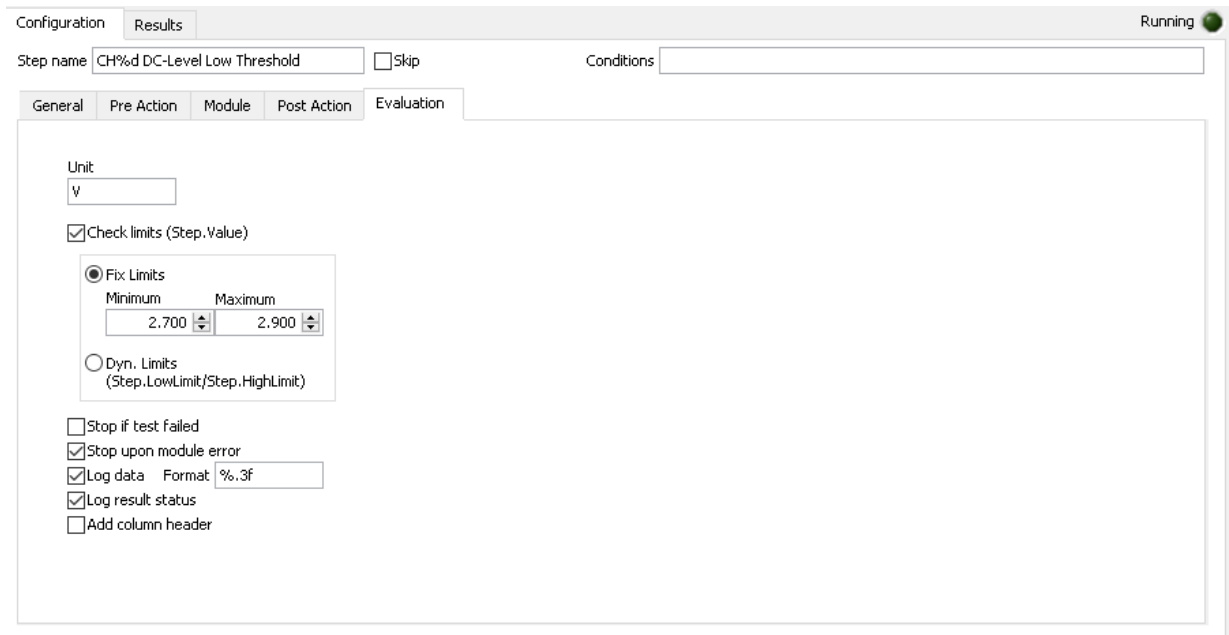



Figure 18: *Module configuration: Post Action*

**Note:** Depending on the type of the selected indicator only type-compatible variables are listed in the variable selection box.

*Evaluation:* In this page the settings for the evaluation of the measured value and the report entry are made.

Configuration item	Description	Default value
Unit	Unit of the measured value	""
Check limits	Checks the measured value (Step.Value) against a range	OFF
Limit Mode	Limit range can be fix using Minimum/Maximum parameter or dynamically by using the System variable Step.HighLimit/Step.LowLimit	Fix Limits
Minimum	Lower limit of the range when using Fix Limits mode	-Inf
Maximum	Upper limit of the range when using Fix Limits mode	+Inf
Stop if test failed	Aborts the execution if the limit check failed	OFF
Stop upon module error	Aborts the execution if a module error occurred (error out)	ON
Log data	Adds the measured value to the report	OFF
Format	Sets the display format for the reported value and limits	%.2f
Log result status	Adds the result status to the report	OFF
Add column header	Adds a column header row to the report (before the result row of the current step)	OFF



Configuration Results Running 

Step name: CH%d DC-Level Low Threshold  Skip Conditions:

General Pre Action Module Post Action **Evaluation**

Unit:

Check limits (Step.Value)

Fix Limits

Minimum	Maximum
<input type="text" value="2.700"/>	<input type="text" value="2.900"/>

Dyn. Limits (Step.LowLimit/Step.HighLimit)

Stop if test failed

Stop upon module error

Log data Format:

Log result status

Add column header

Figure 19: Module configuration: Evaluation

### 3.18.3 Section

Besides the common configuration no other settings have to be made. Sections are always added to the report (incl. section status).

### 3.18.4 Variable

This step type is used to assign constant values, variables or a formula to variables. Use the corresponding button to add ('Insert' key), remove ('Delete' key) or edit ('Space' key) assignments.

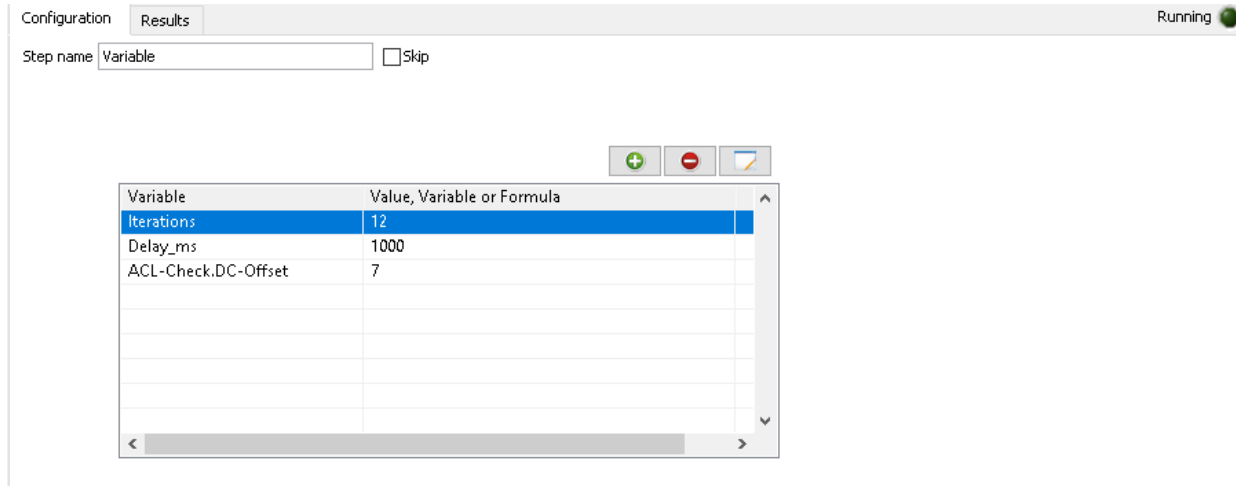


Figure 20: Variable configuration

**Value:** To assign a constant value, simply enter it as normal text. The value will be interpreted according to the assigned variable.

**Variable:** To assign a value containing from another variable use quotation characters ('<Variable>') to reference the variable. For strings (STR) it's possible to concatenate constant text parts with variable values.

**Formula:** To assign a value resulting from a formula the expression must start with the prefix "f=" to distinguish it from a simple assignment.

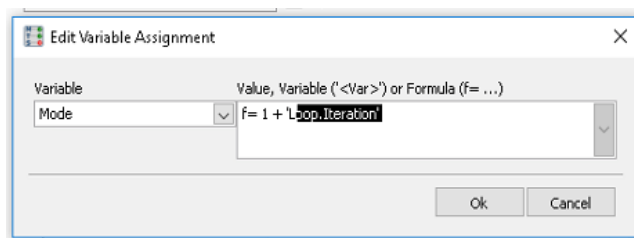


Figure 21: Formula expression

The result value will be interpreted and formatted according to the type of the assigned variable.

### 3.18.4.1 Formulas

#### 3.18.4.1.1 Numbers

Numbers as operands or arguments in formulas are considered in decimal format. Hexadecimal formatted numbers can be represented by a prefix "0x" (e.g. 0xF344AB7C), binary formatted numbers by a prefix "0b" (e.g. 0b10010001).

#### 3.18.4.1.2 Variables

Variables as operands or arguments in formulas can be referenced by using quotation characters ('<Variable>').

#### 3.18.4.1.3 Operators

Following operators can be used in formulas:

<i>Operation</i>	<i>Description</i>
+ x0	Computes unary + of the following term x0.
- x0	Computes unary - of the following term x0.
! x0	Computes logical NOT of x0.
~ x0	Computes bitwise NOT of x0.
x0 ^ x1	Computes x0 raised to the x1 power.
x0 * x1	Computes product of x0 and x1.
x0 / x1	Computes quotient of x0 and x1.
x0 + x1	Computes sum of x0 and x1.
x0 - x1	Computes difference of x0 and x1.
x0 << x1	Shifts x0 left the number of bits specified by x1 as uint32.
x0 >> x1	Shifts x0 right the number of bits specified by x1 as uint32.
x0 < x1	Returns 1 if x0 < x1, otherwise 0.
x0 > x1	Returns 1 if x0 > x1, otherwise 0.
x0 <= x1	Returns 1 if x0 <= x1, otherwise 0.
x0 >= x1	Returns 1 if x0 >= x1, otherwise 0.
x0 == x1	Returns 1 if x0 == x1, otherwise 0.
x0 != x1	Returns 1 if x0 != x1, otherwise 0.
x0 & x1	Computes bitwise AND of x0 and x1 as uint32.
x0 xorb x1	Computes bitwise XOR of x0 and x1 as uint32.
x0 xnorb x1	Computes bitwise XNOR of x0 and x1 as uint32.
x0   x1	Computes bitwise OR of x0 and x1 as uint32.
x0 && x1	Computes logical AND of x0 and x1.
x0 xor x1	Computes logical XOR of x0 and x1.
x0 xnor x1	Computes logical XNOR of x0 and x1.
x0    x1	Computes logical OR of x0 and x1.

### 3.18.4.1.4 Functions

Following functions can be used in formulas:

<i>Function</i>	<i>Description</i>
abs(x0)	Returns the absolute value of x0.
acos(x0)	Computes the inverse cosine of x0 in radians.
acosh(x0)	Computes the inverse hyperbolic cosine of x0.
asin(x0)	Computes the inverse sine of x0 in radians.
asinh(x0)	Computes the inverse hyperbolic sine of x0.
atan(x0)	Computes the inverse tangent of x0 in radians.
atan2((x0;x1)	Computes the arctangent of x1/x0 in radians, where x0 is the x-axis and x1 is the y-axis.
atanh(x0)	Computes the inverse hyperbolic tangent of x0.
ceil(x0)	Rounds x0 to the next higher integer (smallest integer >= x0).
cos(x0)	Computes the cosine of x0, where x0 is in radians.
cosh(x0)	Computes the hyperbolic cosine of x0.
cot(x0)	Computes the cotangent of x0 (1/tan), where x0 is in radians.
csc(x0)	Computes the cosecant of x0 (1/sin), where x0 is in radians.
exp(x0)	Computes the value of e raised to the x0 power.
expm1(x0)	Computes one less than the value of e raised to the x0 power ((e^x0) - 1).
floor(x0)	Truncates x0 to the next lower integer (largest integer <= x0).
getexp(x0)	Returns the exponent of x0 with base 2 (x0 = mantissa * 2^exponent).
getman(x0)	Returns the mantissa of x0 with base 2 (x0 = mantissa * 2^exponent).
if(x0;x1;x2)	If condition x0 is 1 it returns x1, otherwise x2.
int(x0)	Rounds x0 to the nearest integer.
intrz(x0)	Rounds x0 to the nearest integer between x0 and zero.
ln(x0)	Computes the natural logarithm of x0 (to the base of e).
lnp1(x0)	Computes the natural logarithm of (x0 + 1).
log(x0)	Computes the logarithm of x0 (to the base of 10).
log2(x0)	Computes the logarithm of x0 (to the base of 2).
max(x0;x1)	Computes the maximum of x0 and x1.
min(x0;x1)	Computes the minimum of x0 and x1.
mod(x0;x1)	Computes the remainder of the division x0/x1 (x0 - x1*floor(x0/x1)).
pi(x0)	Returns the value of x0*π (pi(2) = 2π). 2pi or 2(pi) will return an error
pow(x0;x1)	Computes x0 raised to the x1 power.
rand()	Produces a floating-point number between 0 and 1 exclusively.
sec(x0)	Computes the secant of x0, where x0 is in radians (1/cos).
sign(x0)	Returns 1 if x0 is greater than 0, returns 0 if x0 is equal to 0, and returns -1 if x0 is less than 0.
sin(x0)	Computes the sine of x0, where x0 is in radians.
sinc(x0)	Computes the sine of x0 divided by x0 (sin/x0), where x0 is in radians.
sinh(x0)	Computes the hyperbolic sine of x0.
sqrt(x0)	Computes the square root of x0.
tan(x0)	Computes the tangent of x0, where x0 is in radians.
tanh(x0)	Computes the hyperbolic tangent of x0.

### 3.18.4.1.5 Argument separator

To separate arguments in functions, always use the semicolon character (;).

### 3.18.5 Loop

If you drag a loop step from the 'Miscellaneous' selection box into the sequence table two steps, a 'Loop-Begin' and a 'Loop-End' step are inserted. The steps inside the 'Loop-Begin' and 'Loop-End' steps are repeated for the configured number of iterations ('Loop Count'), which is set in the 'Loop-Begin' step.

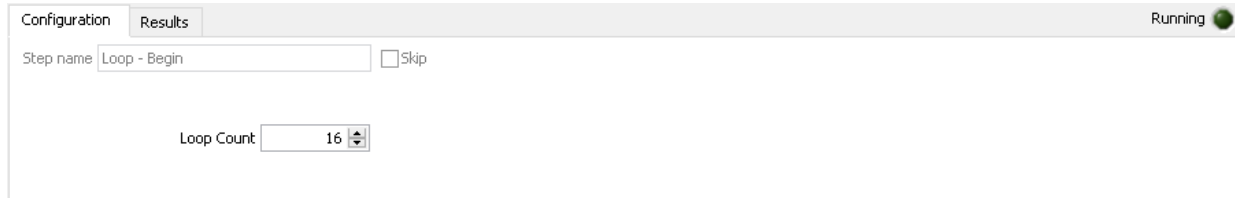


Figure 22: *Loop configuration*

### 3.19 Test Operation

In order to run a test sequence, always ensure that the correct sequence file is loaded, and the hardware test setup is properly installed.

#### 3.19.1 Start

Generally, the sequencer always starts the test sequence from the first step in Pre-UUT's by pressing the button 'Start [F6]'.

**However, if the user has a privilege level above 5 the test execution starts at the currently selected row! Keep in mind that some variables might not be initialized or a device might be still under voltage by previous step executions!**

This feature allows advanced users to start the sequence at a certain point. If no row is selected it starts regularly at the first step in Pre-UUT's.

**Before starting to test series of UUT's always ensure that the last sequence changes are saved in order to be able to associate the generated test results to a certain test sequence revision!**

#### 3.19.2 Step

This feature is for users with privilege level above 5. It allows running the test sequence or parts of it step by step by pressing the button 'Step [F7]'.

**Keep in mind that some variables might not be initialized or a device might be still under voltage by previous step executions!**

#### 3.19.3 Stop

If you press the button 'Stop [F8]' the sequencer always jumps to the Post-UUT's section and executes the steps inside this section. The test sequence is then stopped.

#### 3.19.4 Results Table

During the execution of steps, the current results are displayed in the Results page of the Configuration / Results control according to the step's report configuration. A result line is always displayed as 7 columns:

<i>Column</i>	<i>Description</i>
Name	Name of the test step
Conditions	Conditions text of the module configuration as explained in chapter 0
Unit	Unit of the measured value
Min. Value	Lower limit for the test
Max. Value	Upper limit for the test
Act. Value	Actual measured value
Status	Test status (passed/failed)

The current UUT status is displayed within the 'Overall UUT Status' field. The test and error status of the last executed test step are displayed within the 'Last Step Status' field. If flag 'Show result popup after each UUT' is set, the sequencer shows the result status of the UUT at the end of the test sequence. After one UUT has been completely tested, the sequencer will immediately loop back to the first step in the UUT section, without giving the user the possibility to have a look at the test results. Therefore, by setting the flag 'Pause execution after each UUT' the run process can be halted at this point. To continue with the next UUT simply press 'Start [F6]' again.

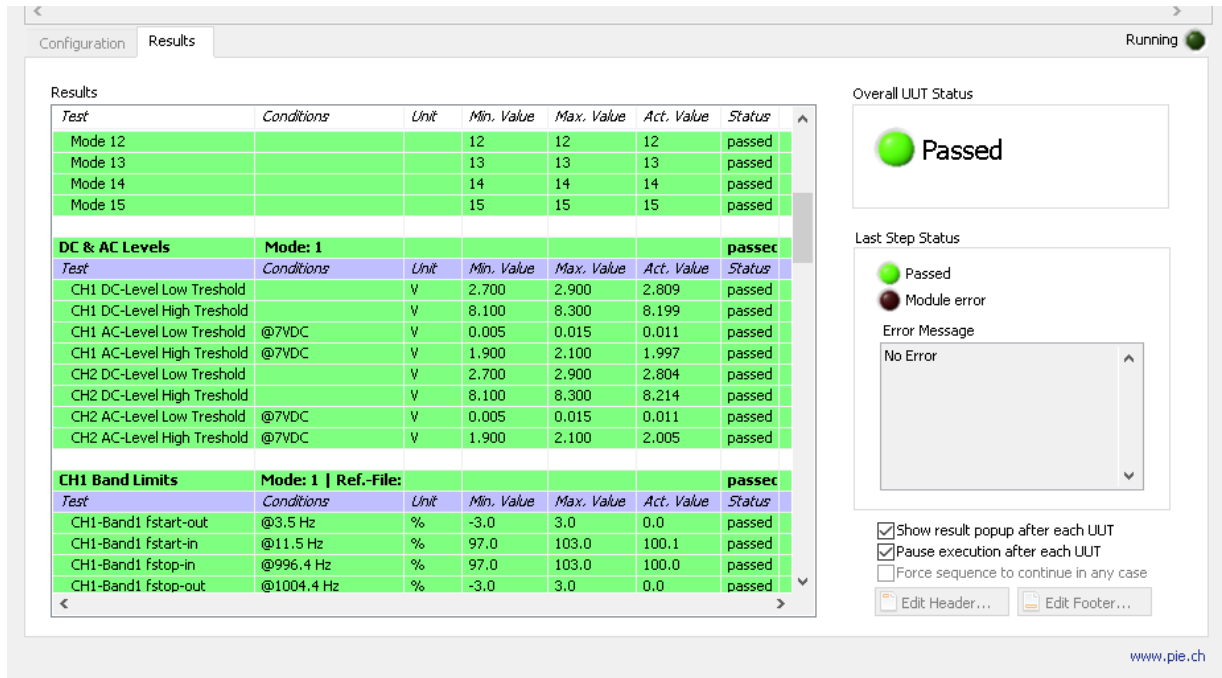


Figure 23: Results Table

### 3.20 Commissioning and debugging options

The following options are for users with privileges above 5. All commissioning/debugging settings are temporary and are not stored in the sequence file.

#### 3.20.1 Debugging log

During debugging, it is possible to view the results of all steps in the debugging log field. To do this, switch to the "Debugging" page and check the "Log" option.

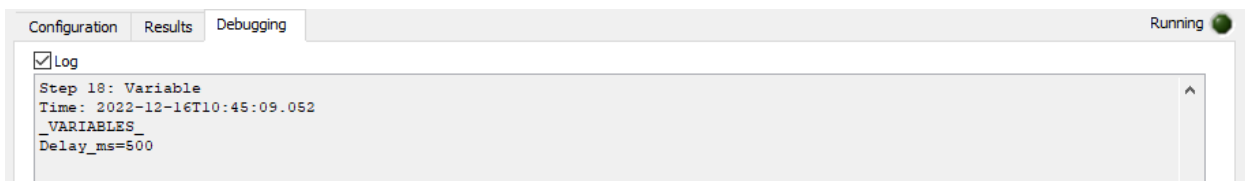


Figure 24: Debugging log field

#### 3.20.2 Breakpoints

In order to pause test execution at a certain step, one can set/clear a breakpoint by clicking the corresponding row left to the step name.

<input type="checkbox"/>	Get Profibus S/N		
<input type="checkbox"/>	Get Display Module S/N		
<input checked="" type="checkbox"/>	Get Pulsation Input Module 1 S/N		
<input type="checkbox"/>	Get Pulsation Input Module 2 S/N		
<input type="checkbox"/>	Advanced Line Check Feedback		

Figure 25: Set/clear breakpoint

### 3.20.3 Debugging steps

In order to execute only a certain set of steps, one has the option to change debugging mode by activating 'Debug', check-marking the corresponding steps and pressing 'Start' or 'Step'. To continuously repeat the sequence of steps additionally activate 'Loop'.

**Notes:**

- The sequencer always starts at the currently selected row (not the first check-marked row).
- For proper initialization of system variables (e.g. 'UUT.Stop') one can set the start position to step 'UUT' before starting the debug sequence.

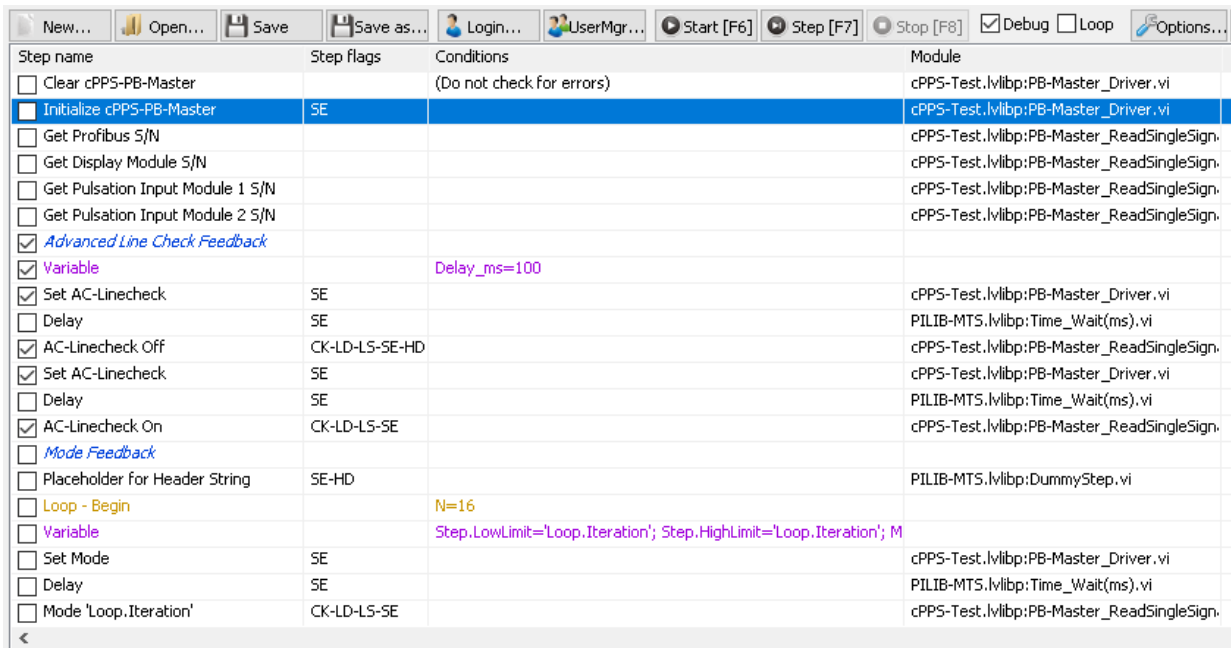


Figure 26: Select steps to run in debugging mode

### 3.20.4 Slow down execution speed

To slow down execution process for a closer investigation of a matter, there is the possibility by manually setting the system variable 'PostStepDelay' (see 3.15.2 and 3.15.4)

### 3.20.5 Force sequence to continue in any case

To avoid abortion of the sequence due to an always failing step or a module error one has the option to set flag 'Force sequence to continue in any case'

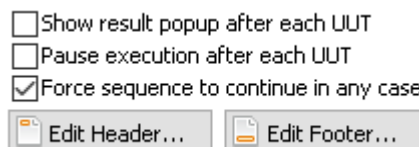


Figure 27: Flag to temporary ignore errors or failed status

## 3.21 Result Files

### 3.21.1 File storage

The result file of each UUT is stored in a subfolder in the 'Results' directory of the current project. The subfolder is named after the sequence file. The result file name is built by using the value of the 'UUT.S/N' variable (i.e. <UUT.S/N>.<ext>). If this variable is empty, the 9-digit value of the 'UUT.Counter' variable is taken for the file name which is increasing with each UUT (<UUT.Counter>.<ext>).

Generally, if a file already exists, an increasing 3-digit index is appended to the file name to avoid overwriting (<ExistingName>\_<iii>.<ext>). The file extension (<ext>) is depending on the result file type, which can be defined in application INI file (see 12)).

**Note:** A result file will be generated if there are any entries available in the Results Table (incl. column header or section rows). If the table remains empty, no file is written.

### 3.21.2 Format

The default format of the result file is based on a table-oriented ASCII file; its columns are separated by a TAB character. The settings can be changed in the program options (see 12). Lines are terminated by the OS-dependent end-of-line character(s) a CR character (carriage return). The file is structured into the 3 sections header, test and footer. Header and footer section are user definable and always have 2 columns, where the test section contains all the results of the test. Optionally, the results can be stored as an Excel sheet (see 3.22).

<i>Header</i>						
<i>Test</i>	<i>Conditions</i>	<i>Unit</i>	<i>Min. Value</i>	<i>Max. Value</i>	<i>Act. Value</i>	<i>Status</i>
<i>Footer</i>						

### 3.21.3 Edit header / footer

To include a customized header/footer in the report press button 'Edit Header...'/ 'Edit Footer...'. The header/footer section has a fixed 2-column format. It allows the user to include values of program variables or simple texts in the form key -> value. Use the corresponding button to add ('Insert' key), remove ('Delete' key) or edit ('Space' key) assignments. Put the text into quotation characters with optional format specifier ('<Variable> [, <format specifier>']') if you want to reference to a variable. It's also possible to leave both fields empty in order to add some space lines.

**It's strongly recommended to include the sequence revision variable ('TestSequence.Rev') in the result file header or footer (see example 4.6).**

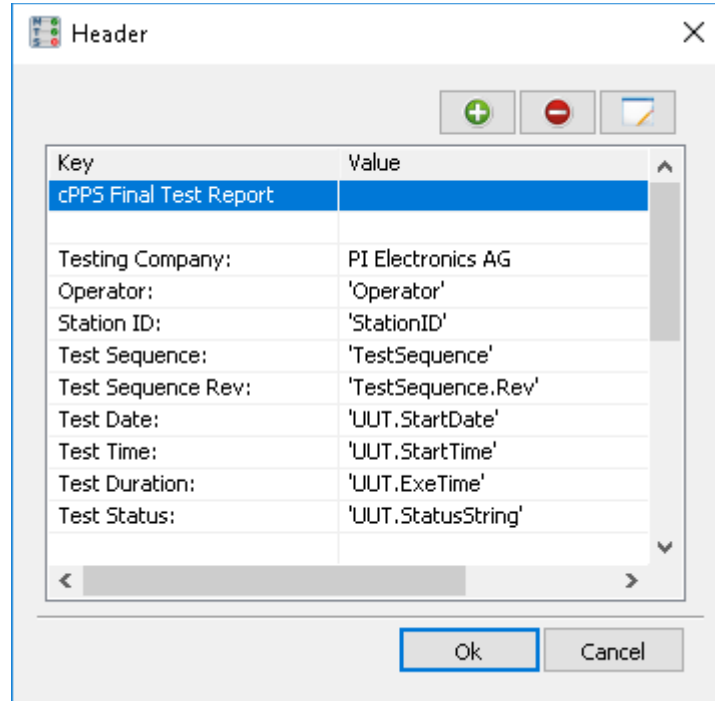


Figure 28: Header/Footer definition

### 3.22 Program options

To change the file format of the result file, open the options dialog by pushing 'Options' button. One has the choice between tab-limited (\*.txt) or comma separated values (\*.csv) text files, where the column separator can be defined, or an Excel file.

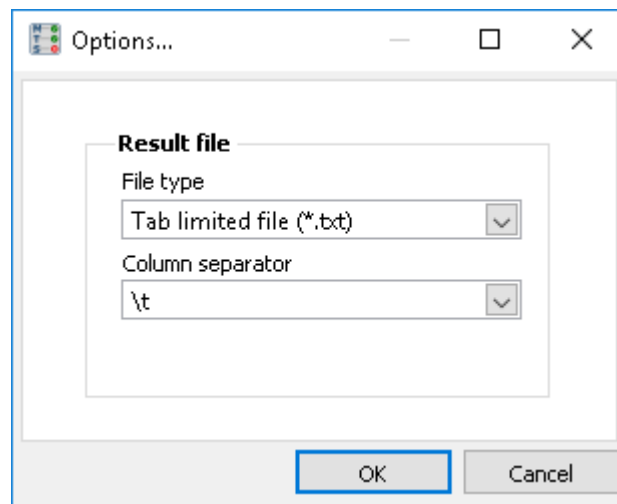


Figure 29: Options dialog

### 3.23 Program exit

Use the Windows close button(X) to quit the application.

### 3.24 Program shortcuts

For easier operation of the most important program functions some keyboard shortcuts are provided:

<i>Function</i>	<i>Shortcut</i>
New project/sequence	Ctrl+N
Open sequence	Ctrl+O
Save sequence	Ctrl+S
Login	Ctrl+L
User manager	Ctrl+U
Options dialog	Ctrl+M
Open Help File	F1
Start sequence	F6
Step sequence	F7
Stop sequence	F8
Add item	Insert
Remove item	Delete
Edit item	Space

## 4 Appendix

### 4.1 Guidelines for LabVIEW modules

Generally, no special conventions must be followed to integrate LabVIEW into the sequencer. However, the following guidelines will improve the compatibility with the MTS framework:

- Do NOT use arrays for front panel controls/indicators, so the module can interface with the MTS framework (see chapter 4.2).
- Controls/indicators with cluster structures are allowed. Atomic data elements inside structures are referenced as <cluster label name>.<...>.<atomic element label name> by the framework.
- To pass errors from the module to the framework (module error), the standard error output cluster must be named to 'error out' (label name).
- Do not use 'Open Front Panel when called' in the window properties, since it has no effect for dynamically called VI's. Use property nodes instead, if you wish a front panel to open.

#### 4.1.1 Using the variable Step.Attachment

Since the result evaluation of the MTS framework is based on a fixed data type (Step.Result), it's sometimes necessary to process the evaluation inside the module step (e.g. if the measured value is an array). In this case it's possible to format the result table as TAB separated text inside the module and pass it to the 'Step.Attachment' variable in a post action variable assignment. In this way the module's implicit result values are added to the result file. It's advantageous to stick to the MTS result format for the internally created result table in order to get a proper report (7 columns).

### 4.2 Supported interface data types for LabVIEW modules

In order to pass data from MTS framework variables to the LabVIEW module's atomic control elements or vice-versa numerous data types are allowed which are compatible with the four variable data types (chapter 3.15.1):

#### 4.2.1 Boolean (BOOL)

Name	Shortcut	LabVIEW type code (decimal)
Boolean	BOOL	33

#### 4.2.2 Signed Integer 64 Bit (I64)

Name	Shortcut	LabVIEW type code (decimal)
Signed Integer 8Bit	I8	1
Signed Integer 16Bit	I16	2
Signed Integer 32Bit	I32	3
Signed Integer 64Bit	I64	4
Unsigned Integer 8Bit	U8	5
Unsigned Integer 16Bit	U16	6
Unsigned Integer 32Bit	U32	7
Enumerated Integer 8Bit	Enum8	21
Enumerated Integer 16Bit	Enum16	22
Enumerated Integer 32Bit	Enum32	23

#### 4.2.3 Double Precision Floating-Point Number (DBL)

Name	Shortcut	LabVIEW type code (decimal)
Single Precision Floating-Point Number	SGL	9
Double Precision Floating-Point Number	DBL	10

#### 4.2.4 String (STR)

<i>Name</i>	<i>Shortcut</i>	<i>LabVIEW type code (decimal)</i>
String	STR	48
Path	PATH	50
DAQmx String	DAQmx	55

### 4.3 LabVIEW Format Specifiers

For conversion of any variable into reported text you can use LabVIEW format specifiers in the form '<Variable>, <format specifier>'. For functions that produce a string as an output, a format specifier uses the following syntax elements. Double brackets ( `[]` ) enclose optional elements.

`%[$][-][+][#][^][0][<Width>][.<Precision> | _<SignificantDigits>]Conversion Code`

where <Width> must be a number greater than zero and .Precision and <SignificantDigits> must be a number greater than or equal to zero.

#### 4.3.1 Format Specifiers Syntax Elements

The following table displays the syntax elements for format specifiers. Refer to the examples in chapter 4.3.2 for more information.

Syntax element	Description
%	Begins the format specifier.
\$ (optional)	When you use a formatting function, this modifier specifies the order in which to display variables. Include the digit that represents the order of the variable immediately before this modifier.
- (optional)	When you use a formatting function, this modifier left justifies the parameter rather than right justifies it within its width.
+ (optional)	When you use a formatting function, this modifier includes sign even when the number is positive.
^ (optional)	When you use a formatting function and the e or g conversion codes, this element formats the number in engineering notation, where the exponent is always a multiple of three.
# (optional)	When you use a formatting function, this modifier removes trailing zeros. If the number has no fractional part, this modifier also removes the description part.
0 (optional)	When you use a formatting function, use this modifier without the - modifier to pad any excess space to the left of a numeric parameter with zeros rather than with spaces to reach minimum width.
Width (optional)	When you use a formatting function, the Width element specifies the minimum character field width of the output. This width is not a maximum width. LabVIEW uses as many characters as necessary to format the parameter without truncating it. LabVIEW pads the field to the left or right of the parameter with spaces, depending on justification. If Width is missing or 0, the output is only as long as necessary to contain the converted input parameter.
.Precision or _Significant Digits (optional)	<p>When you use a formatting function, . or _ controls the number of digits displayed. If you use ., LabVIEW uses the number that follows as a precision specifier for digits to the right of the decimal point. If you use _, LabVIEW uses the number that follows as the specified number of significant digits to use in the display format.</p> <p>.Precision—When you use it with floating-point notation, this element specifies the number of digits to the right of the decimal point. If . is not present, LabVIEW uses a precision of six digits. If . is 0, LabVIEW does not insert a precision. When you use it with string parameters, .Precision specifies the maximum width of the scanned field. LabVIEW truncates strings longer than this length.</p> <p>_Significant Digits—Displays the data by rounding to the number of digits you specify. LabVIEW rounds the data only for display purposes, which does not affect the original data. .Precision affects only the digits to the right of the decimal point, and _Significant Digits includes all non-spacing digits. For example,</p> <ul style="list-style-type: none"> <li>– 3.457 has 4 significant digits</li> <li>– 0.0012 has 2 significant digits</li> <li>– 123000 has 3 significant digits</li> </ul> <p><b>Note:</b> You cannot use precision and significant digits together in a single format specifier.</p> <p>For single-precision, floating-point numbers, National Instruments recommends that you use values from 1 to 6 for _Significant Digits. For double-precision and extended-precision, floating-point numbers, National Instruments recommends that you use values from 1 through 13 for _Significant Digits.</p>
<Embedded Format Information>	Contains a time-specific format string. Refer to the chapter 0 for valid format strings. Only %W, %D, %H, %M, %S and %u apply to relative time.
Conversion Codes	<p>Characters that specify how to scan or format a parameter.</p> <p>Use the following conversion codes for integers and fixed-point numbers:</p> <ul style="list-style-type: none"> <li>– x—Hexadecimal integer (for example, B8).</li> <li>– o—Octal integer (for example, 701).</li> <li>– b—Binary integer (for example, 1011).</li> <li>– d—Signed decimal integer.</li> <li>– u—Unsigned decimal integer.</li> </ul> <p>Use the following conversion codes for floating-point and fixed-point numbers:</p>

	<ul style="list-style-type: none"> <li>– f—Floating-point number with fractional format (for example, 12.345).</li> <li>– e—Floating-point number in scientific notation (for example, 1.234E1).</li> <li>– g—LabVIEW uses f or e depending on the exponent of the number. LabVIEW uses f if the exponent is greater than –4 or less than the precision specified. LabVIEW uses e if the exponent is less than –4 or greater than the precision specified.</li> <li>– p—Floating-point number in SI notation.</li> </ul>
	<p>Use the following conversion codes for a string:</p> <ul style="list-style-type: none"> <li>– s—String (for example, abc). When scanning, s matches only up to the next white-space character. A space matches one or more consecutive white-space characters. To scan a string that may contain white space, use the characters in set conversion code. Specify all the characters which the string may contain within the brackets, including space or other white-space characters.</li> <li>– [ ]—Characters in set. [ ] matches a string that contains only the characters specified between the brackets. Character matches are case sensitive. The [ ] conversion code is useful only when you are scanning strings. To match the caret symbol (^) in a set, make sure it is not the first character after the bracket. The following examples demonstrate the use of the characters in set conversion code:             <ul style="list-style-type: none"> <li>• %[aeiou]—Matches any string that contains only lowercase vowels.</li> <li>• %[0-9a-zA-Z ]—Matches a string that contains numbers, letters, or spaces. You can use a hyphen to specify ranges of characters in the set.</li> <li>• %[^\s;]—Matches any string of characters up to but not including the first comma or semicolon.</li> </ul> <p style="margin-left: 20px;">To match a hyphen, specify it as the first or last character in the set.</p> </li> </ul>
	<p>Use the following conversion codes for time:</p> <ul style="list-style-type: none"> <li>– T—Absolute time. Refer to chapter 4.3.2 for examples of using absolute time.</li> <li>– t—Relative time. Refer to chapter 4.3.2 for examples of using relative time.</li> </ul>

#### 4.3.1.1 Format Codes for the Time Format String

The Format Date/Time String function calculates the date and time and formats the output string according to the following format codes.

Format Code	Value
<%<>T>	absolute time container
<%^<>T>	universal time container
<%a>	abbreviated weekday name (for example Wed)
<%A>	full weekday name (for example Wednesday)
<%b>	abbreviated month name (for example Jun)
<%B>	full month name (for example June)
<%c>	locale-specific default date and time
<%d>	day of month (01–31)
<%H>	hour (24-hour clock) (00–23)
<%I>	hour (12-hour clock) (01–12)
<%j>	day number of the year (001–366)
<%m>	month number (01–12)
<%M>	minute (00–59)
<%p>	AM or PM flag
<%S>	second (00–59)
<%<digit>u>	fractional seconds with <digit> precision
<%U>	week number of the year (00–53), with the first Sunday as the first day of week one; 00 represents the first week
<%w>	weekday as a decimal number (0–6), with 0 representing Sunday
<%W>	week number of the year (00–53), with the first Monday as the first day of week one; 00 represents the first week
<%x>	locale-specific date
<%1x>	long date format
<%2x>	locale-specific date
<%X>	abbreviated long date format
<%y>	year within century (00–99)
<%Y>	year, including the century (for example, 1997)
<%z>	difference between locale time and universal time (HH:MM:SS)
<%Z>	time zone name or abbreviation, depending on the operating system locale settings

#### Notes:

- LabVIEW returns abbreviated weekday and month names as numeric values for systems that do not support abbreviated names, such as Chinese and Korean.
- You cannot use <%H> (24-hour clock) and <%p> (AM/PM flag) together in the same time format string. If you use <%I> (12-hour clock) instead of <%H>, <%p> operates properly.

### 4.3.2 Format Specifier Examples

Type	Argument(s)	Format String	Resulting String	Comments
Automatic Formatting (%g)	12.00	%#g	12	If you specify #, LabVIEW removes trailing zeros. If you specify g, LabVIEW chooses scientific notation or floating-point notation based on the number to format.
	12000000	%#g	1.2E+6	
Decimal (%d)	12.67	score= %.0f%%	score= 13%	When you specify %f, LabVIEW rounds the argument. Use %% to format a single %. Use .0 to remove decimal.
Floating-Point (%f)	12.67	Temp: %5.1f	Temp: 12.7	The 5 in the <b>Format String</b> section specifies a width of 5, and the 1 specifies the number of digits to the right of the decimal, or precision.
Scientific Notation (%e)	12.67	%.3e	1.267E+1	Add ^ to change to engineering notation where the exponent is always a multiple of three.
	12.67	%^.3e	12.670E+0	
SI Notation (%p)	12000000	%.2p	12.00M	A value of .2 specifies that you want a precision of 2 digits in the resulting string. A value of _2 specifies that you want two significant digits in the resulting string. %p adds the SI prefixes.
	12000000	%.2p	12M	
Hexadecimal (%x)	12	%02x	0C	A - indicates that you want the resulting string left justified. A 0 indicates that you want the resulting string padded with zeros. The number specifies the width of the resulting string. In other words, a value of -n pads the resulting string with spaces to the n width. b is base 2, o is base 8, and x is base 16.
Octal (%o)	12	%06o	000014	
Binary (%b)	12	%b	1100	
Relative Time (%t)	91.80	%.2t	01:31.80	LabVIEW formats elapsed time in terms of complete weeks (%W), days (%D), hours (%H), minutes (%M), seconds (%S), and fractions of seconds (%u). Refer to the chapter 0 for more time format codes.
	91.8	%<Hours:%H Minutes:%M Seconds:%S>t	Hours:00 Minutes:01 Seconds:31	
Absolute Time (%T)	00:00:00.000 AM 1/1/2001 (Universal Time)	%<%3X %x>T	12:00:00.000 AM 01/01/2001	%T specifies absolute time. Any information you include with the < and > brackets indicates how you want to format the absolute time. This format, including the punctuation, changes based on the regional settings of the computer. The time changes based on the configured time zone for the computer. Refer to the chapter 0 for more time format codes.
	00:00:00.000 AM 1/1/2001 (Universal Time)	%<%Y.%m.%d>T	2001.01.01	
	00:00:00.000 AM 1/1/2001 (Universal Time)	%^<%3X %x>T	06:00:00.000 AM 12/31/2000	
String (%s)	Smith John	Name: %s, %s.	Name: Smith, John.	In the second example, the format string specifies to use at most six characters from the string Hello, World then pad with spaces so the total string length is equal to 10.
	Hello, World	String: %10.6s	String: Hello,	

## 4.4 How to create a Source Distribution or a Packed Library?

To use LabVIEW functions in the test sequencer it's necessary to create either a source distribution of the top-level VI's or a Packed Library (\*.lvlibp), so all needed Sub-VI's from the vi.lib, instr.lib or user.lib are included in the module set.

### 4.4.1 Source Distribution

- 1) In the Project Explorer navigate to 'Build Specification' and open the context menu 'New' -> 'Source Distribution':

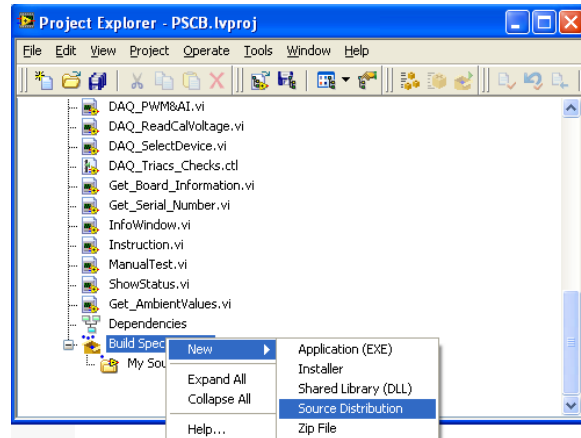


Figure 30: *New Source Distribution*

- 2) Distribution Settings: Enter a name for the distribution definition, choose 'Single Destination' as 'Packaging Option' and define the directory where the distribution should be stored:

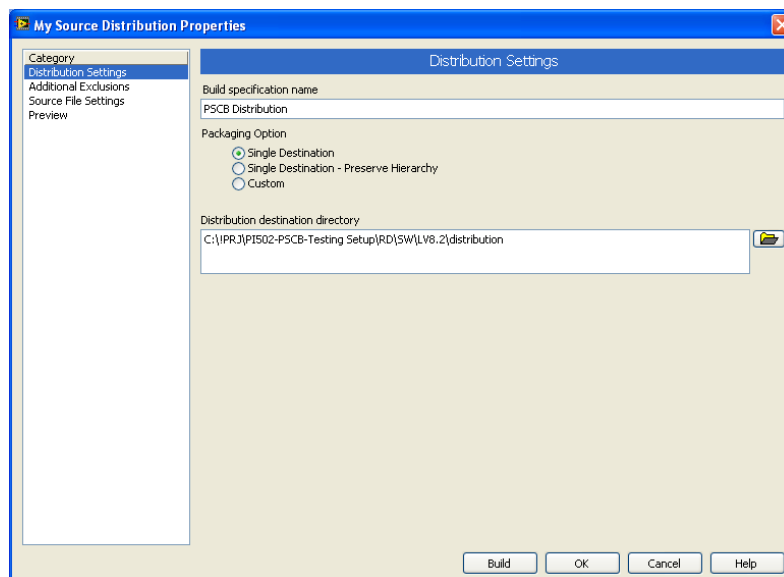


Figure 31: *Distribution Settings*

- 3) Additional Exclusions: Select option 'Remove as much as possible' and uncheck all exclusion boxes, so all library VI's are included in the distribution.

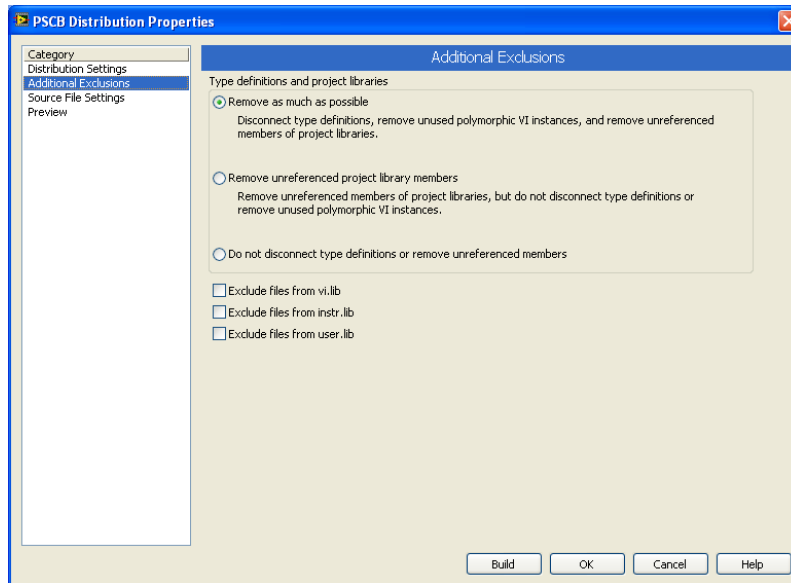


Figure 32: *Additional Exclusions*

- 4) Push 'Build' to create the distribution and 'OK' to save the build definition in the project.

### 4.4.2 Packed Library

LabVIEW packed project libraries are project libraries that package multiple files into a single file with a .lvlib file extension. The top-level file of a packed library is a project library. By default, the packed library has the same name as the top-level project library.

- 1) Precondition: All your needed test modules have to be inside a project library, which is part of the project:

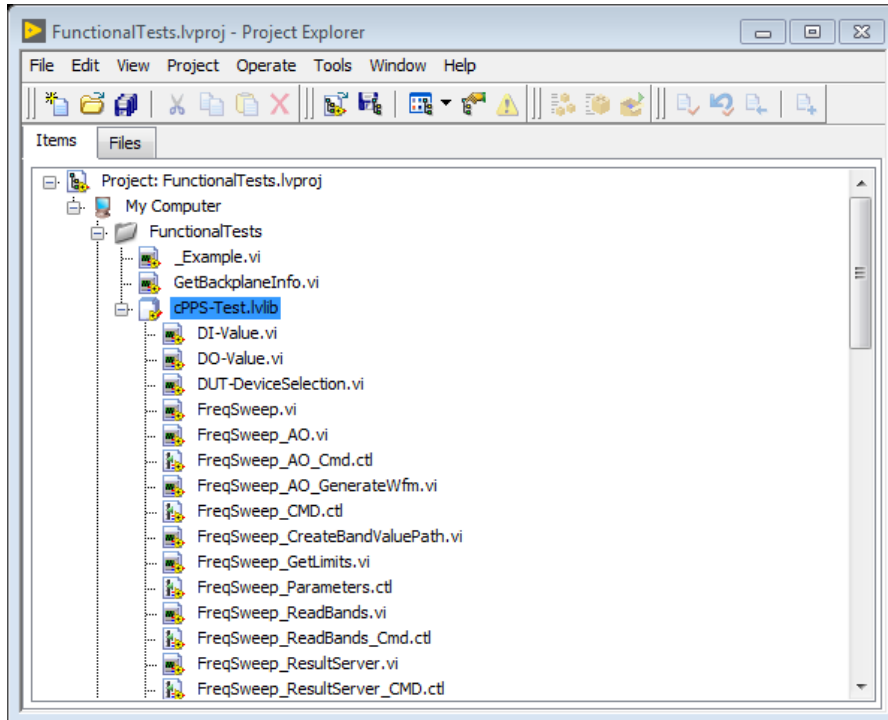


Figure 33: Test modules arranged as a project library

- 2) From the Project Explorer window, right-click 'Build Specifications' and select New->Packed Library from the shortcut menu to display the Packed Library Properties dialog box and configure settings to build a packed library:

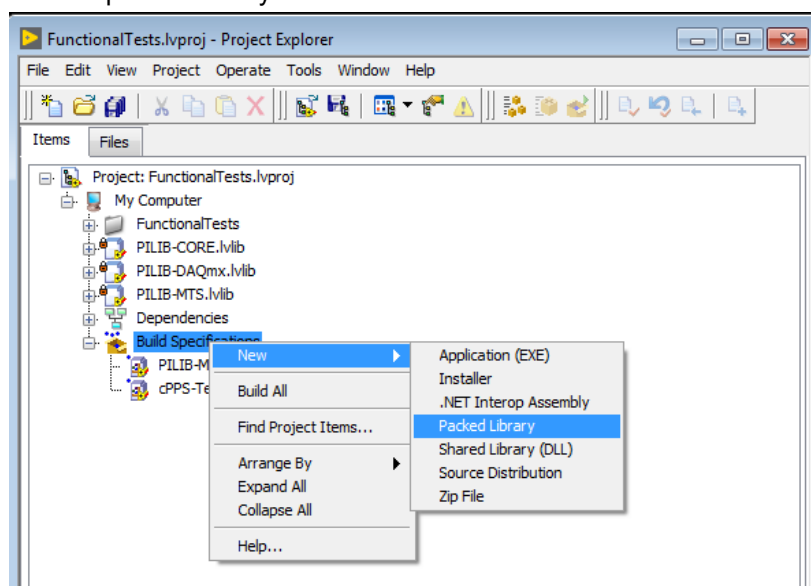


Figure 34: Create a new Packed Library

3) Information: Enter build parameters such as target folder and file name.

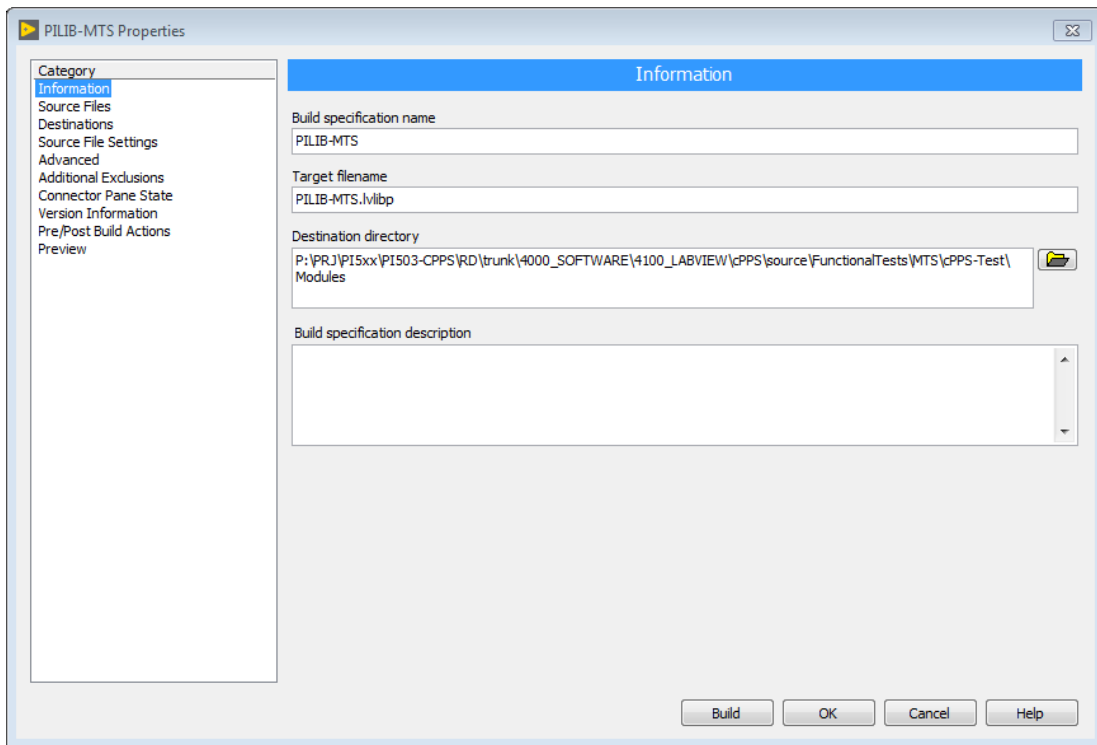


Figure 35: Packed Library naming and destination paths

4) Source Files: Select source project library (\*.lvlib) as Top-Level Library:

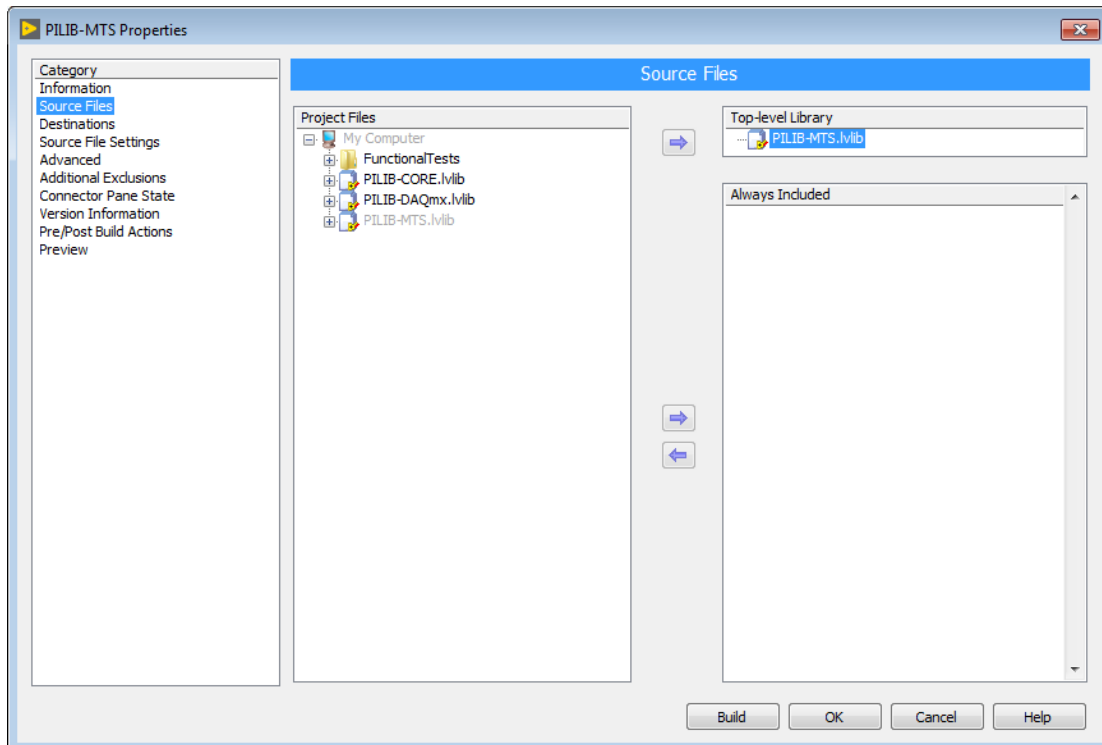


Figure 36: Packed Library source files selection

- Destinations: Define target directories and if needed any subdirectory for support files. A packed library contains only LabVIEW files. By default, LabVIEW saves non-LabVIEW files to the same destination directory as the packed library. Select Support Directory in the Destinations list and change the path in the Destination path text box to change where LabVIEW saves non-LabVIEW files.

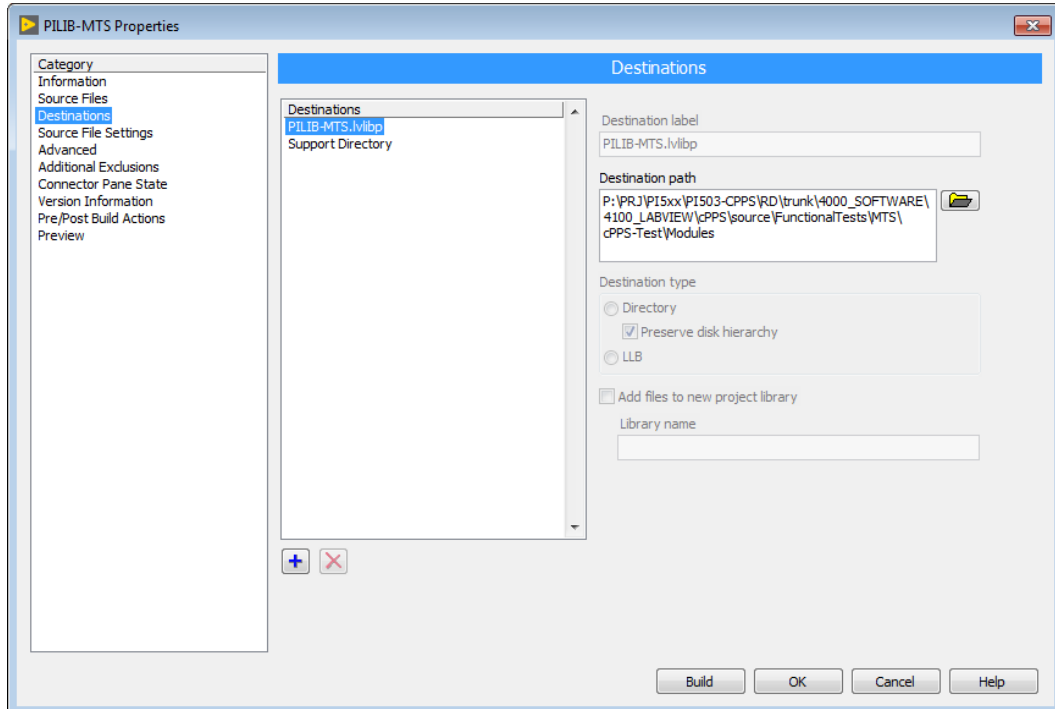


Figure 37: Packed Library build destination path

- Source File Settings: Customize any VI properties if needed, otherwise use the default settings:

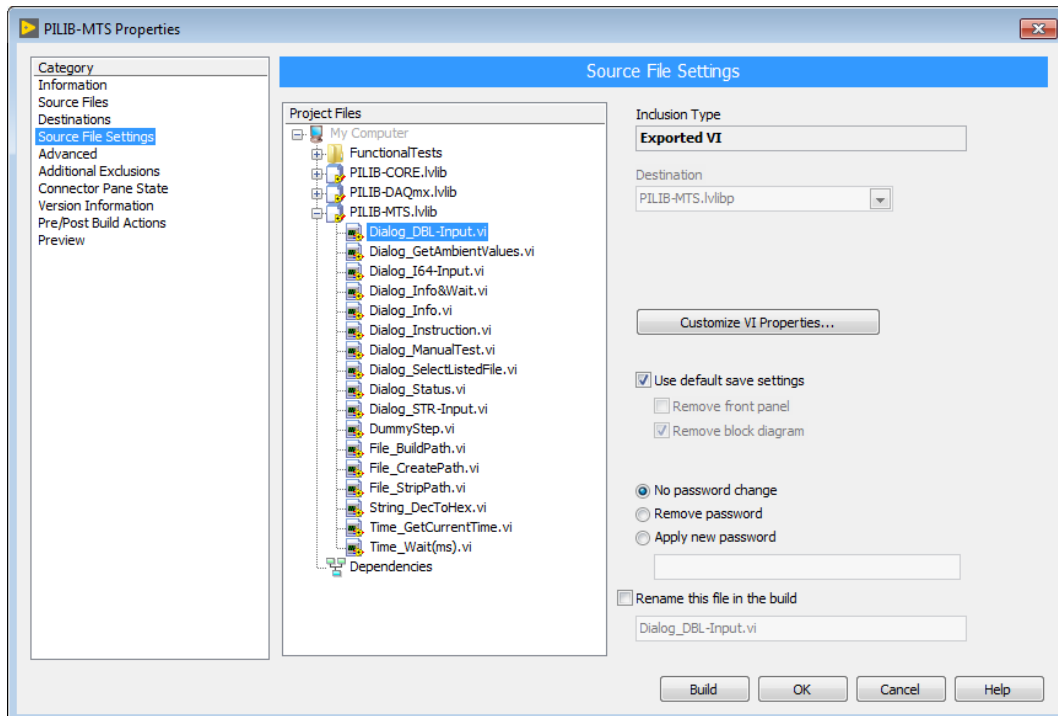


Figure 38: Packed Library source file settings

- 7) Advanced: Checkmark 'Allow future versions of LabVIEW to load the packed library' in order to allow future versions of LabVIEW Runtime Engines to load the library without recompilation:

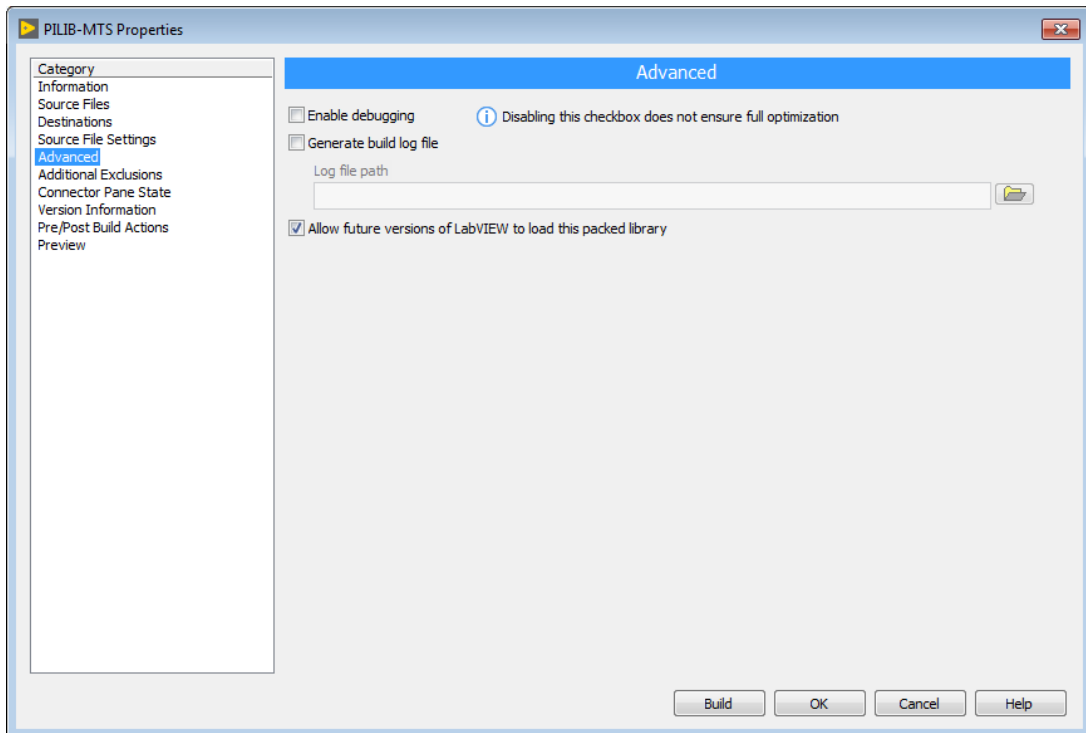


Figure 39: *Packed Library advanced build options*

- 8) Additional Exclusions: Define any files which should be excluded from the build, otherwise use default settings:

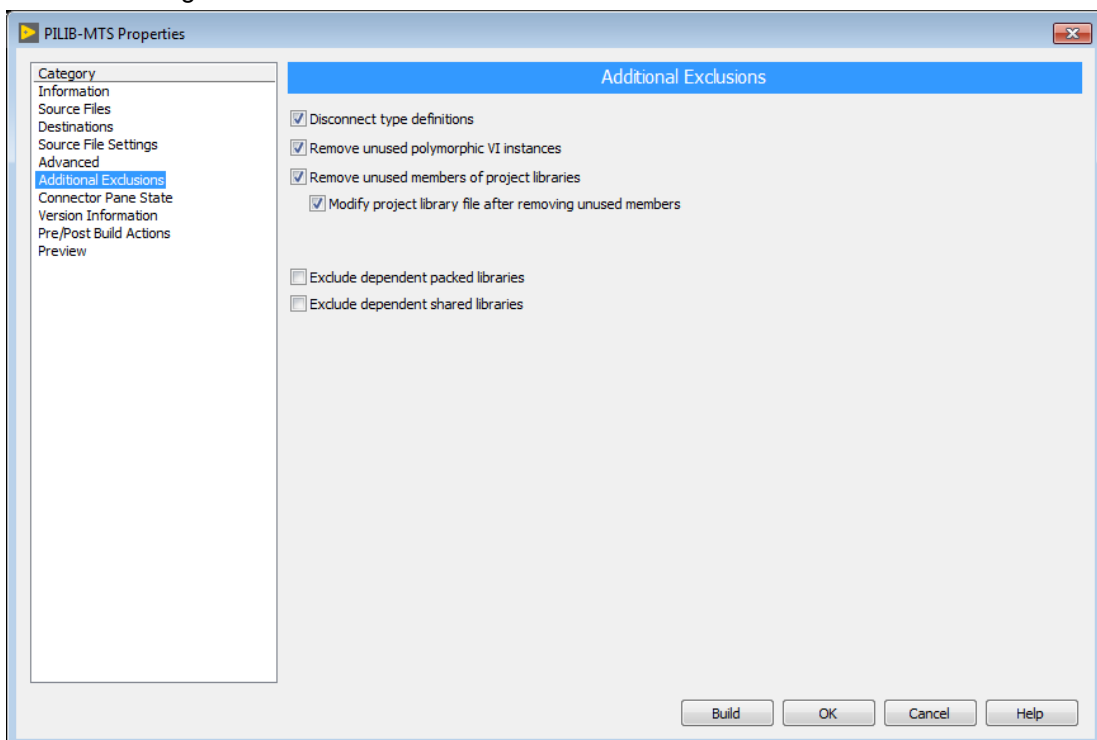


Figure 40: *Packed Library file exclusions*

- 9) Connector Pane Settings: Enable 'Callers adapt at run time to Exported VI connector pane state' in order to avoid recompilation of any caller VI's in case the connector pane of a library member changes.

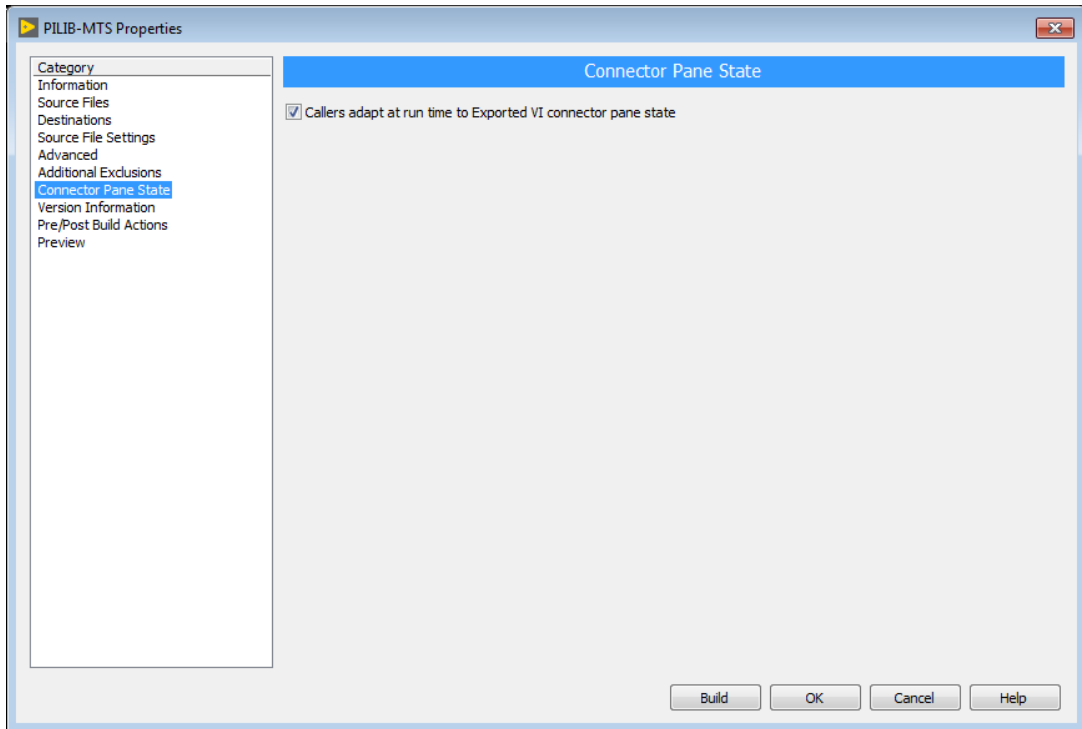


Figure 41: *Packed Library pane compatibility*

- 10) Version Information: Use consistent version numbers when building a library. The library name and version number will be included in the system variable 'TestModules.Libraries' (see 3.15.2) and can be used for test documentation (e.g. for report header):

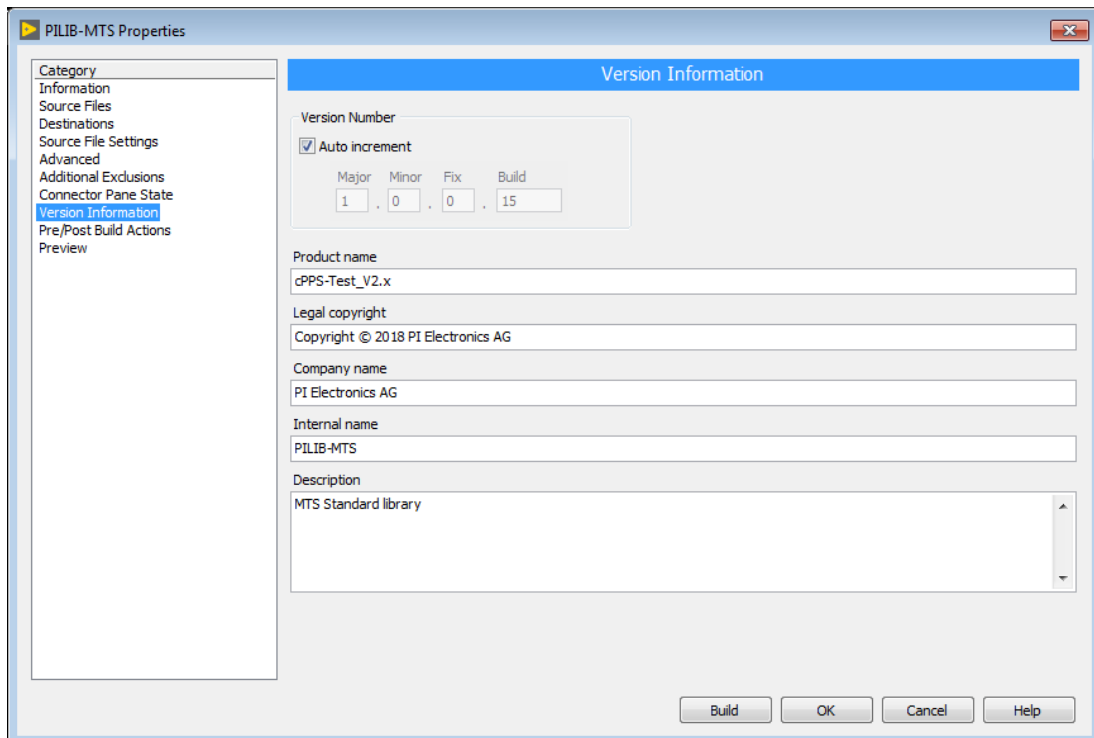


Figure 42: *Packed Library version definition*

- 11) Pre/Post Build Actions: Specify any VIs to execute before or/and after build process, otherwise leave unchecked:

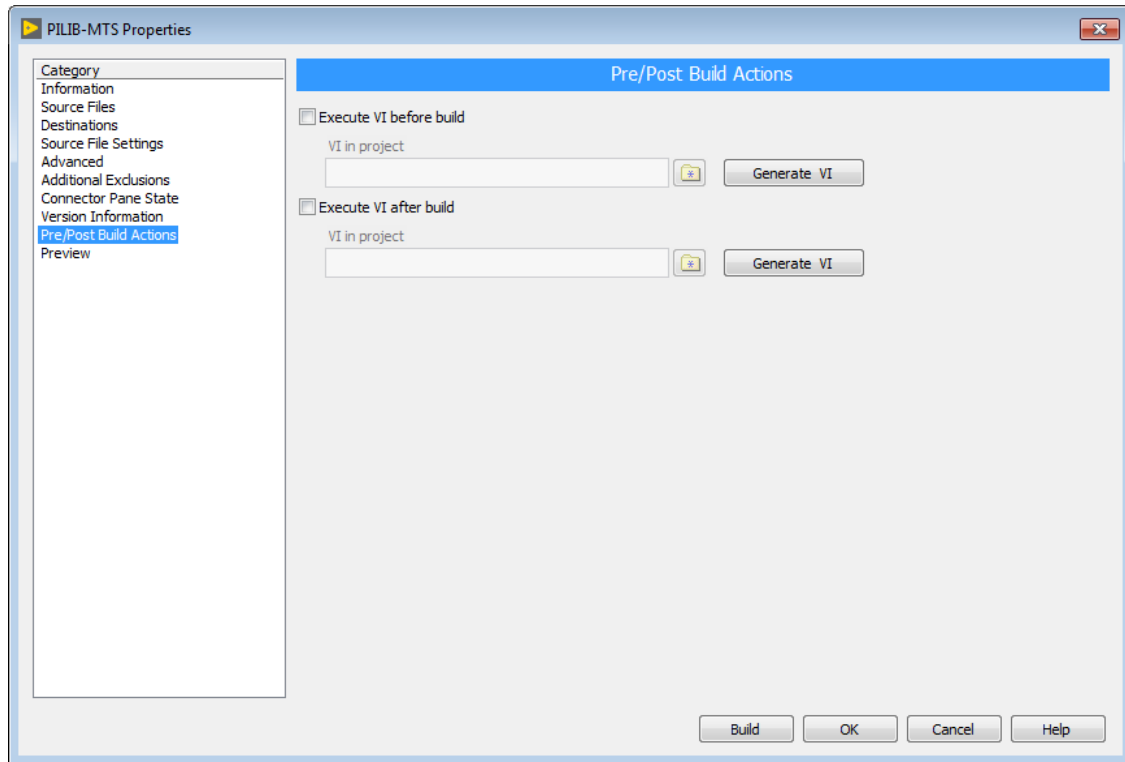


Figure 43: Packed Library Pre/Post build actions

- 12) Push 'Build' to create the packed project library and 'OK' to save the build definition in the project.

## 4.5 MTS application files

Following a list of files necessary to run the MTS application. Some of them need write access by the MTS application, some read-only access.

File name	Folder location	Access	Description
MTS.exe	<ProgramFilesFolder>\PI Electronics AG\MTS	R	Application executable
MTS.ini	<ProgramFilesFolder>\PI Electronics AG\MTS	R	Application base settings
MTS_Releases.txt	<ProgramFilesFolder>\PI Electronics AG\MTS	R	Application release history
MTS_OperatingManual.pdf	<ProgramFilesFolder>\PI Electronics AG\MTS	R	Application help document
lvanlys.dll	<ProgramFilesFolder>\PI Electronics AG\MTS	R	Library for mathematical functions
MTS_Settings.ini	<ProgramDataFolder>\PI Electronics AG\MTS	R/W	Application configuration settings
<HW-ID>_License.ini	<ProgramDataFolder>\PI Electronics AG\MTS	R/W	License key file
groups.def	<ProgramDataFolder>\PI Electronics AG\MTS\Users	R/W	User group definition
*.psw	<ProgramDataFolder>\PI Electronics AG\MTS\Users	R/W	User password files

## 4.6 Report Example

Test Report							
Part name:	PSCB						
Part number:	3BUS208796-001						
Revision:	REV. A						
PO:	D106018						
Manufacturer:	PI Electronics AG						
Serial number:	CH07390015						
Test date:	17.12.2008						
Test time:	19:30:09						
Test duration:	237.3 s						
Ambient temperature:	22 °C						
Relative humidity:	40 %						
Test station:	PI-T500-1						
Test file:	C:\Projects\PSCB\PSCB-Test.mtsq						
Test file revision:	38						
Test status:	failed						
Test	Conditions	Unit	Min. Value	Max. Value	Act. Value	Status	
Supply Voltages						failed	
+15V: Voltage	Nominal	V	14.80	15.20	14.95	passed	
+15V: Current	Nominal	A	0.010	0.040	0.093	failed	
-15V: Voltage	Nominal	V	-15.20	-14.80	-14.98	passed	
-15V: Current	Nominal	A	-0.030	-0.010	-0.023	passed	
+12V: Voltage	Nominal	V	11.50	12.50	12.02	passed	
-12V: Voltage	Nominal	V	-12.50	-11.50	-12.05	passed	
+15V: Voltage	-10%	V	13.20	13.80	13.47	passed	
+15V: Current	-10%	A	0.010	0.040	0.089	failed	
-15V: Voltage	-10%	V	-13.80	-13.20	-13.44	passed	
-15V: Current	-10%	A	-0.040	-0.010	-0.023	passed	
Digital Outputs Open Drain						passed	
DH-CH1: Current		A	0.230	0.242	0.239	passed	
DH-CH1: LED Check	Visual Check					passed	
DH-CH2: Current		A	0.230	0.242	0.236	passed	
DH-CH2: LED Check	Visual Check					passed	
DH-CH3: Current		A	0.230	0.242	0.236	passed	
DH-CH3: LED Check	Visual Check					passed	
DH-CH4: Current		A	0.230	0.242	0.238	passed	
DH-CH4: LED Check	Visual Check					passed	
Digital Outputs PWM						passed	
PWM-CH1: Current	@ 100% PWM	A	1.22	1.32	1.28	passed	
PWM-CH1: Current	@ 50% PWM	A	0.65	0.71	0.71	passed	
PWM-CH1: LED Check	Visual Check					passed	
PWM-CH2: Current	@ 100% PWM	A	1.22	1.32	1.29	passed	
PWM-CH2: Current	@ 50% PWM	A	0.65	0.71	0.71	passed	
PWM-CH2: LED Check	Visual Check					passed	
AC-Trip Switch						passed	
AC-Trip: Line Voltage		Vrms	110.00	130.00	124.81	passed	
AC-Trip: Ref. Current		Arms	1.265	1.317	1.290	passed	
Operator:	J. Sample						

**Notes:**



Segelhofstrasse 1  
CH-5405 Baden  
Switzerland

Phone: +41 56 222 20 01  
E-Mail: [info@pie.ch](mailto:info@pie.ch)  
Web: [www.pie.ch](http://www.pie.ch)