

MESSINA Manual



Inhaltsverzeichnis

Inhaltsverzeichnis	2
Manual	5
Using MESSINA Help	6
What is MESSINA?	7
Overview	7
Features	7
Test Design	7
Test Configuration	7
Test Execution	8
Windows and Views	9
Workspaces	9
Project Explorer	11
Copying Signals, Parameters, Campaigns, and Testenvironments	12
Importing Projects	14
Importing an Existing MESSINA Project into the Current Workspace	14
Signalpool Manager	17
Add Signals to the Signalpool	17
Add Groups to the Signalpool	18
Signal Selection and Synchronization	20
Problems View	21
Properties View	23
Scheduling	24
Scheduling Configuration	24
Scheduling Parameter	26
Example	26
Designer Perspective	28
Parameter Manager	29
Add a Parameter	29
Test Case Editor	31
Add a Test Case	31
Programming the Test Case	33
Configurator Perspective	34
Configuration Manager	35
Add a Configuration to the Testenvironments	35
Managing the Ports	38
Library Explorer	42
Executor Perspective	43
Target Manager	44
Add a Target (Not available in SiL-only version)	44
Connect a Target	45
Result Manager	48
Console	51
Icons:	51
Log View and Log Levels	53
Log Levels	54
Preferences	56
ASCII Logger	56
Binary Logger	56
CAN ASC Logger	57
MDF Logger	57
Remote	57
Test Protocol Format	57

Test Settings	57
Visualization	58
Creating and Executing Tests	59
Programming Test Cases	60
Logging and Log Levels in Test Cases	62
Signal Capture	62
Using Parameters	64
Adding Parameters	64
Setting Parameters for a Campaign	65
Parameter Access within a Test Case	67
Setting Parameters for a Test Case	67
Test Campaigns	71
Add a Campaign	71
Nested Campaigns	73
Executing Tests	75
Signal Commands	76
waitTrigger()	77
waitValue()	78
setValue()	79
getValue()	80
getLocalValue()	81
register()	82
registerNotification()	83
log()	84
logValue()	86
Coregion()	88
sleep()	90
ASSERT()	91
ASSERT_EQUALS()	92
Visualization	93
Control Panels	94
Adding a Control Panel Visualization	94
Header Icons	94
Available Controls	95
Control Properties	96
Button	99
LED	100
Thermometer	101
Tank	102
Slider	104
Dial	106
Meter	108
Graph	109
Adding a y(t) Plot Visualization	109
Header Icons	110
Control Properties	110
Set the Scale of a Signal and Line Properties	111
Set Graph Properties	112
Panning and Zooming	112
Label	113
Check Box	114
Radio Button	115
Numeric Edit	117
Combo Box	118

Image	120
Graph Visualizations	121
Adding a y(t) Plot Visualization	121
Header Icons	122
Control Properties	122
Set the Scale of a Signal and Line Properties	123
Set Graph Properties	124
Panning and Zooming	124
Signal Capture	125
Table View Visualizations	127
Adding a Table View Visualization	127
Header Icons	127
Control Properties	128
Modifying a Table View Signal Value	129
Test Results	131
Test Reports	132
Test Reports	132
Signal Traces and Data Logging	136
Adding a Logger Visualization	136
Header Icons	136
Control Properties	137
Logging Signals	137
ASCII Signal Trace Files	138
MDF Signal Traces	139
Log View	139
FAQ	140

Manual

Using MESSINA Help

This document is designed to show users how to use the MESSINA test development environment quickly and effectively. Screen shots are used extensively throughout this document to illustrate the many features and options available.

A glossary of terms is provided to explain the meaning of acronyms and terms used throughout this document.

The MESSINA help uses the following conventions:

1. Terms or words used in the MESSINA development environment are shown ***Bold Italic*** (e.g. ***Configurator, Signalpool, Project Explorer*** etc.)
2. Menu locations within MESSINA or within this documentation are given using the notation: e.g. MenuItem → SubMenu → Item Selection (e.g. ***Window*** → ***Show View*** → ***Configuration Manager***). This would refer to the "Selection" item in the "SubMenu Item" submenu of the "MenuItem" main menu item.
3. Sections of source code used to describe Test Cases in JAVA script are shown in monospace

Sample source code section:

```
public class TestCase0 extends AllSignals {  
  
    public int run() throws IOException, TestFailedException, InterruptedException {  
  
        //Insert the Test Sequence here  
  
        return 0; //0 means PASSED  
  
    }  
  
}
```

What is MESSINA?

Overview

MESSINA is a model based software platform used for ECU testing. MESSINA can be used for all stages of testing from specification through to HiL. Hardware independent test sequences can be used seamlessly for software tests (SiL) and hardware tests (HiL). The test strategy can be extended across processes between OEMs and suppliers.

A highlight of MESSINA is its ability to integrate models from different modelling environments (like MATLAB/Simulink and many others) together with embedded systems in a full system test scenario. These models can be dynamically integrated at runtime and executed in parallel. The tests are described in JAVA notation. The modular Eclipse based design of MESSINA simplifies the integration of external editors for test case description. This feature allows MESSINA to be easily and completely integrated into different customer environments. MESSINA achieves a better test coverage for distributed ECU development and provides easy portability of test sequences between different test systems.

Features

- Creation of real-time capable test sequences in JAVA notation
- Simultaneous use of models from several modelling environments
- Automated execution of a series of Test Cases in a [Test Campaign](#)
- Hardware abstraction allows all Test Campaigns to run on different test platforms (e.g. HiL, SiL)

Test Design

The Test Cases in MESSINA are described using JAVA notation. All necessary control structures for the description of complex test sequences are available.

Variations of Test Cases can be created by taking a generalized test case and using parameters which can be set for specific test requirements. This allows the same test case to be used with different configurations which are defined by the parameters.

The symbolic signals of all models and I/Os used in the MESSINA project are available and shown in a global signalpool. The signalpool is the central point where all components have access to signals and data.

Test Configuration

Built-in Wizards allow quick and easy integration of the models into the Testenvironments and the configuration of all I/Os (e.g. analogue and digital I/O, CAN,etc). The Testenvironments (e.g. HiL or SiL)

configuration is realized by mapping the signals in the signalpool to either I/Os or models. Standard industry signal catalogues are available in the library (e.g. CANdb). These are automatically imported and available immediately for configuration of the test environment.

Test Campaigns consist of a series of Test Cases put together into a test sequence. The test cases within a campaign are executed in the sequence they are displayed in the [Project Explorer](#) under the Campaigns folder.

Test Execution

MESSINA Test Campaigns can be composed to execute automatically. The [Target Manager](#) is used to connect to a target system. Once the connection is made, the test environment configuration can be started.

Protocol files and measurement data are stored in the MESSINA project structure and are available for further evaluation after the test process is completed. The connection to existing version management systems secures all necessary information about the test process.

Many different Visualization options are integrated into MESSINA. These include graphic displays, table listings of individual signals, and signal loggers. The different visualization options can be used to view and monitor signals, states, and measurement data while the test process is running.

Windows and Views

This section will describe the MESSINA components, the different perspectives available, and the options available for each perspective. It is possible for you to customize the perspectives by detaching individual views and attaching them to other MESSINA perspectives. For purposes of explanation, we will use the default version of each perspective. The default perspective can be re-created at any time by calling **Main Menu** → **Window** → **Reset Perspective**.

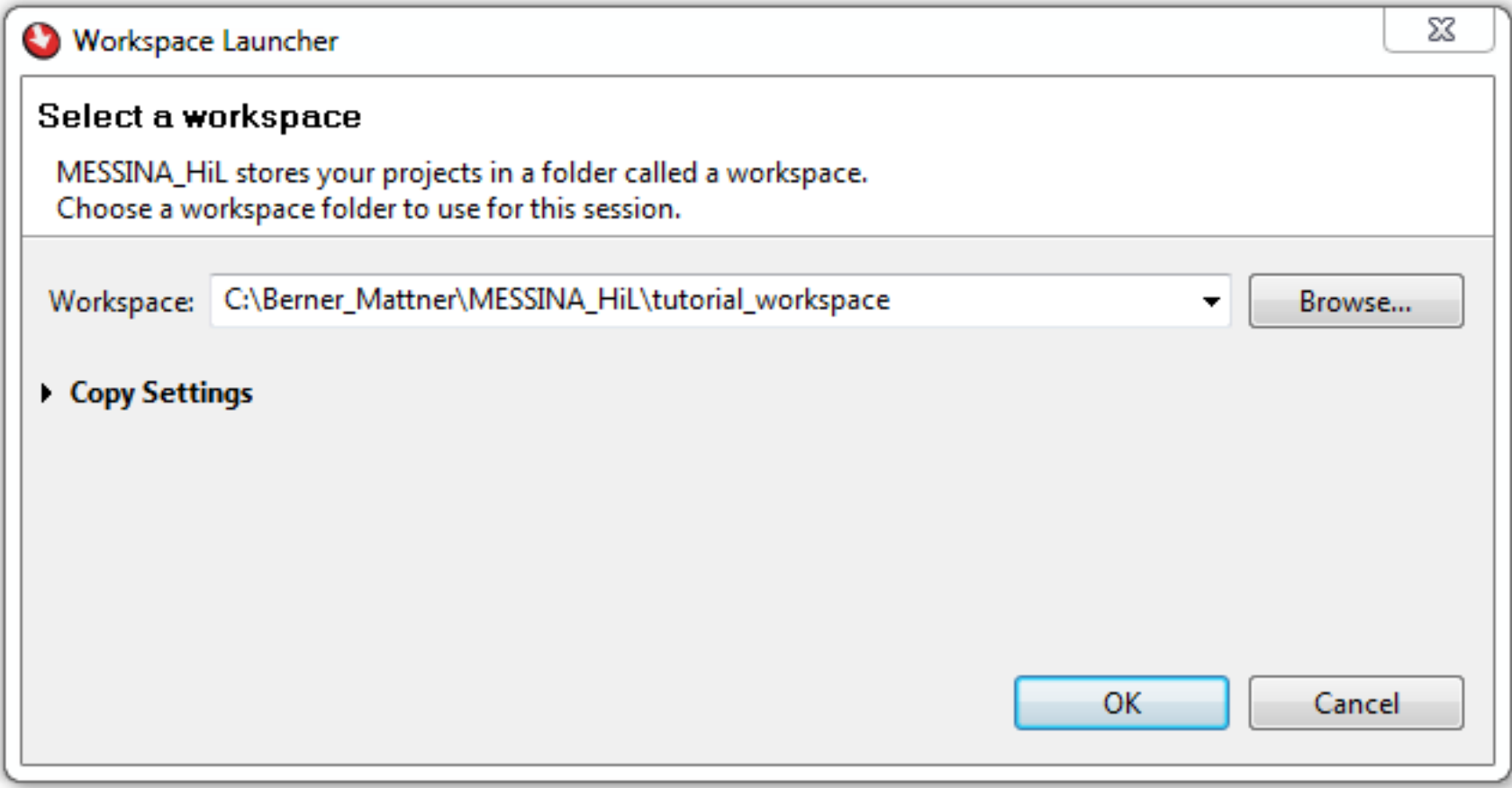
The [Project Explorer](#) view is shown in all perspectives. It is always on the left side of the main window and ensures that an overview of the project is always available. The MESSINA main menu is also displayed at the top of the main window in each perspective. The other views being displayed will depend on the perspective currently selected. Each view, regardless of which perspective it is associated with, can be called directly from **Main Menu** → **Window** → **Show View** and then selecting the desired view. Each view can be closed by clicking on the white "X" next to its name.

The MESSINA development environment consists of 3 main perspectives: the **Designer**, **Configurator**, and the **Executor**. You can switch between these perspectives by clicking on the appropriate tab at the top of the main MESSINA window or by calling **Main Menu** → **Window** → **Open Perspective** and selecting the desired perspective.

Workspaces

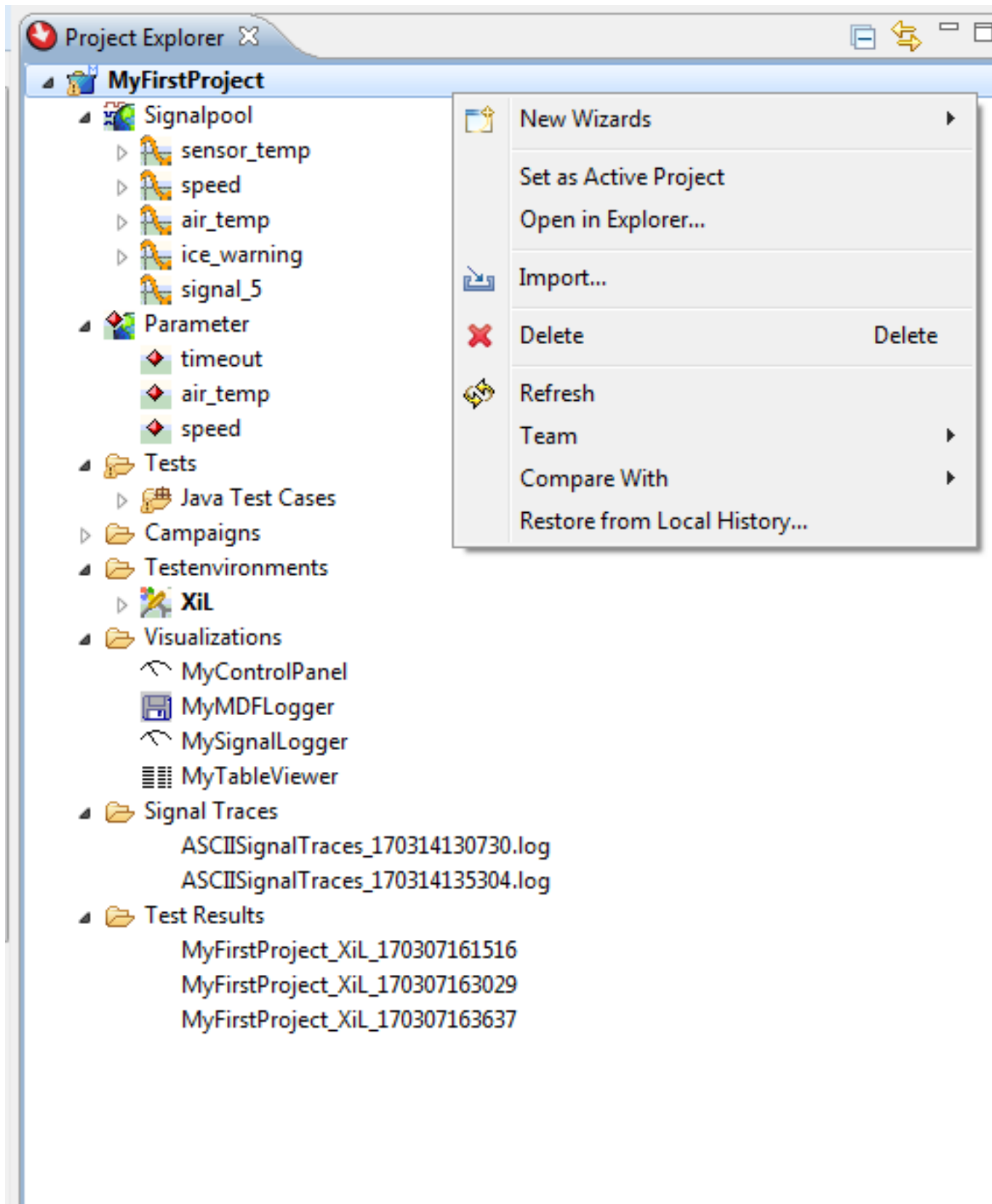
MESSINA uses **Workspaces** to organize its projects. This allows you to organize different MESSINA projects yourself. Examples would be for you to create a **Workspace** for a particular type or group of tests. A **Tutorial Workspace** is part of the MESSINA installation and can be used for sample projects. We will use the **Tutorial Workspace** to create and run a detailed example of a MESSINA project.

To switch between **Workspaces** use **Main Menu** → **File** → **Switch Workspace**. The last different workspace is listed, as well as an **Other** option. The following dialog gives an example of selecting a new **Workspace**, in this case the **tutorial_workspace** will be selected:



Project Explorer

The MESSINA **Project Explorer** view is available in all 3 perspectives (**Designer**, **Configurator**, and **Executor**). It lists all available projects and their associated elements in the form of a tree structure. An example is shown in the picture below which also includes the context menu available for each project.



The currently active project is bold (**MyFirstProject** in the example above). All items within the project structure can be viewed in the project explorer view. The context menu (as shown above for the selected project) offers the following options:

- **New Wizards:** This will call a sub-menu which lists wizards used for adding an element to the MESSINA project or to create a new project.

- **Set as Active Project:** This will make the currently selected project the active project. Its name will be bold.
- **Open in Explorer...:** This will open the windows explorer with the path where the selected component is located.
- **Import...:** This will open a dialog to import files, projects and models.
- **Delete:** When a project name is selected, this will delete the entire project with all its associated files and data. If another item (e.g. a visualization) is selected, the visualization will be deleted. All delete actions must be confirmed (dialog box) before they are performed.
- **Refresh:** The content of the **Project Explorer** view is updated

Note: the other items listed in the context menu apply to external tools (e.g. SubVersion) which are optional. These are not described in this documentation.

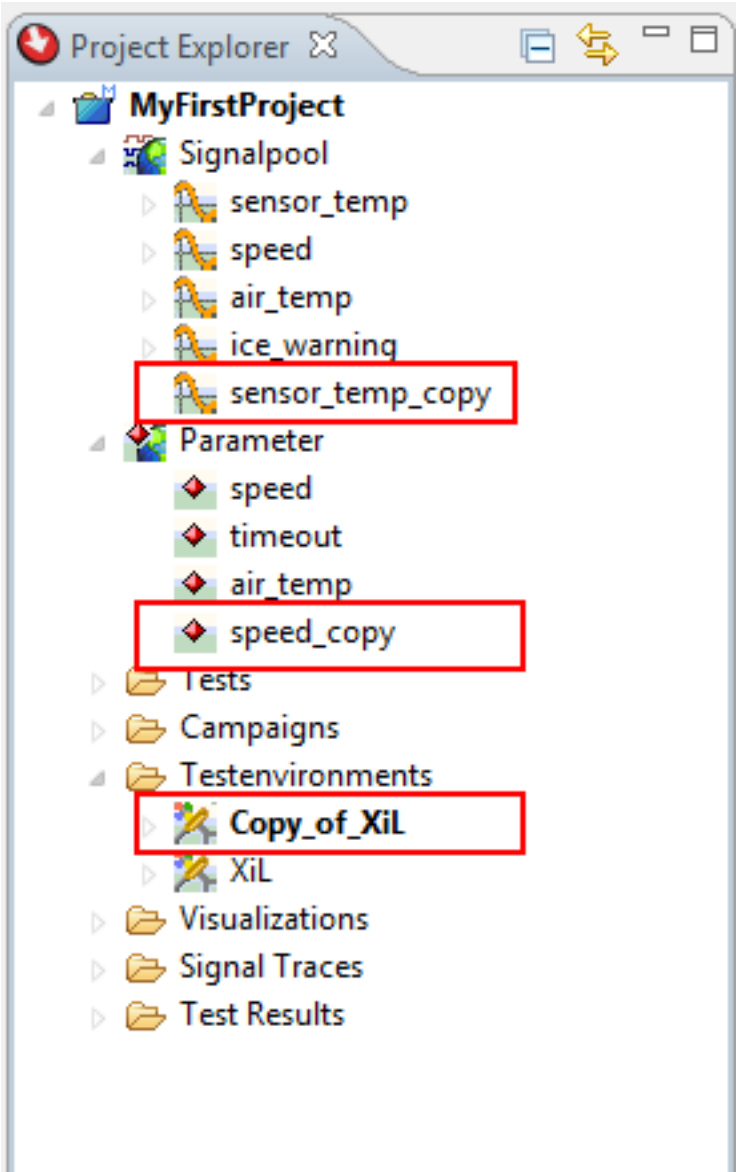
The context menu will vary depending on the type of element that is selected and which perspective is currently being displayed. The operations available for specific folders or items are described in the appropriate section. For example, although it may be possible to add a **signal group** to the **Signalpool** using the **Project Explorer**, this operation will be described in the [Signalpool Manager](#) section of this documentation.

The **Project Explorer** view can be used to delete items from the project. Adding items is performed using the appropriate other view (e.g. use the **Signalpool Manager** to add a new signal to the project). The **Project Explorer** view can also be used to arrange the order of some elements (e.g. the order of the signals in the signalpool can be changed by drag-and-drop operations).

Copying Signals, Parameters, Campaigns, and Testenvironments

Copying signals, parameters, full campaigns, and test environments can be done in the project explorer view. Mark the item to be copied, press and hold the **CTRL** key, press and hold the left mouse button, then move the mouse to the location where the item to be copied into. A "+" sign will indicate that it is possible to perform the copy operation when moving the mouse. Release the mouse button to perform the copy operation.

The picture below shows a signal, a parameter and a test environment after each copy operation.

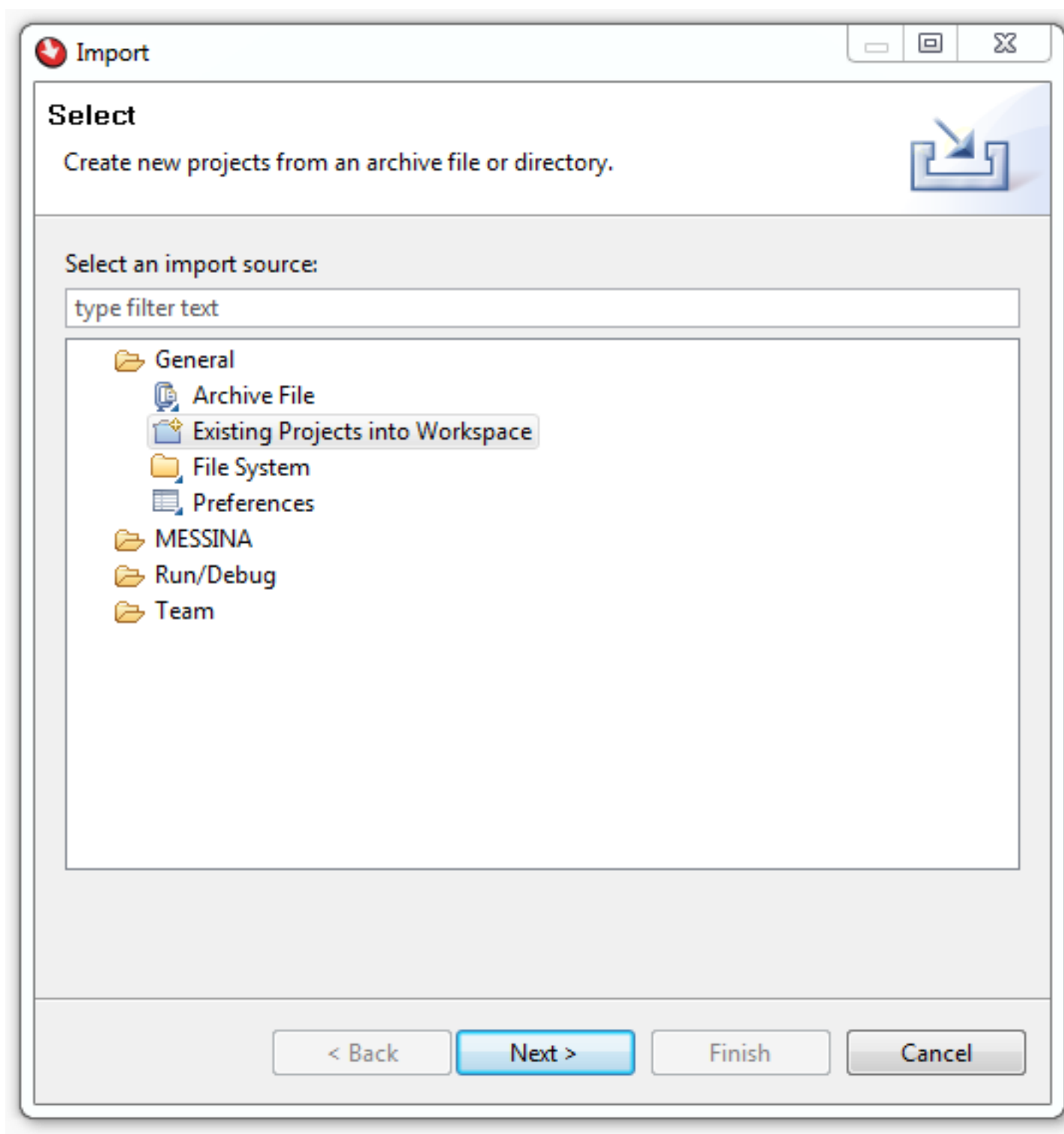


Importing Projects

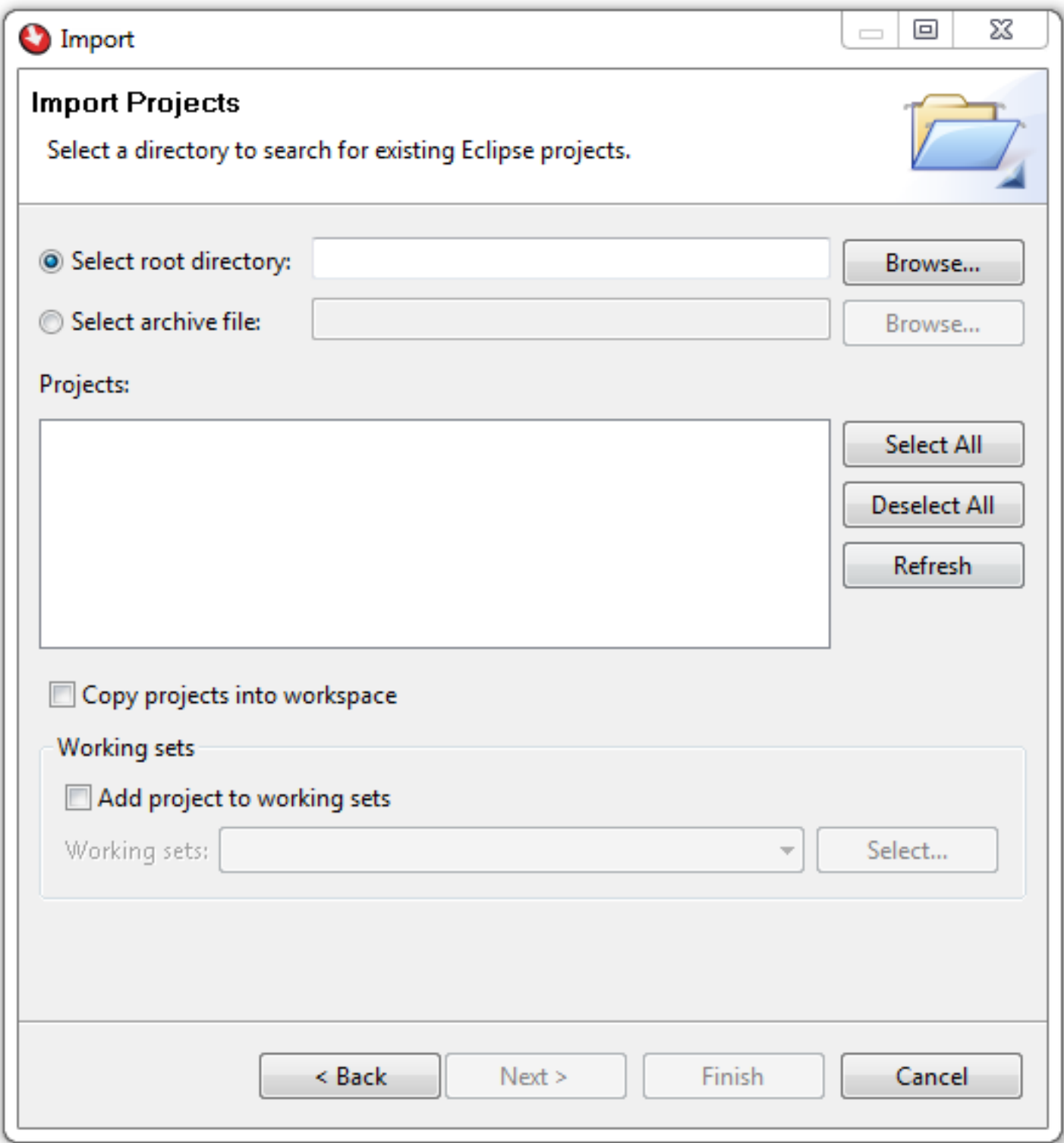
This section describes how to import an existing MESSINA project into the current workspace.

Importing an Existing MESSINA Project into the Current Workspace

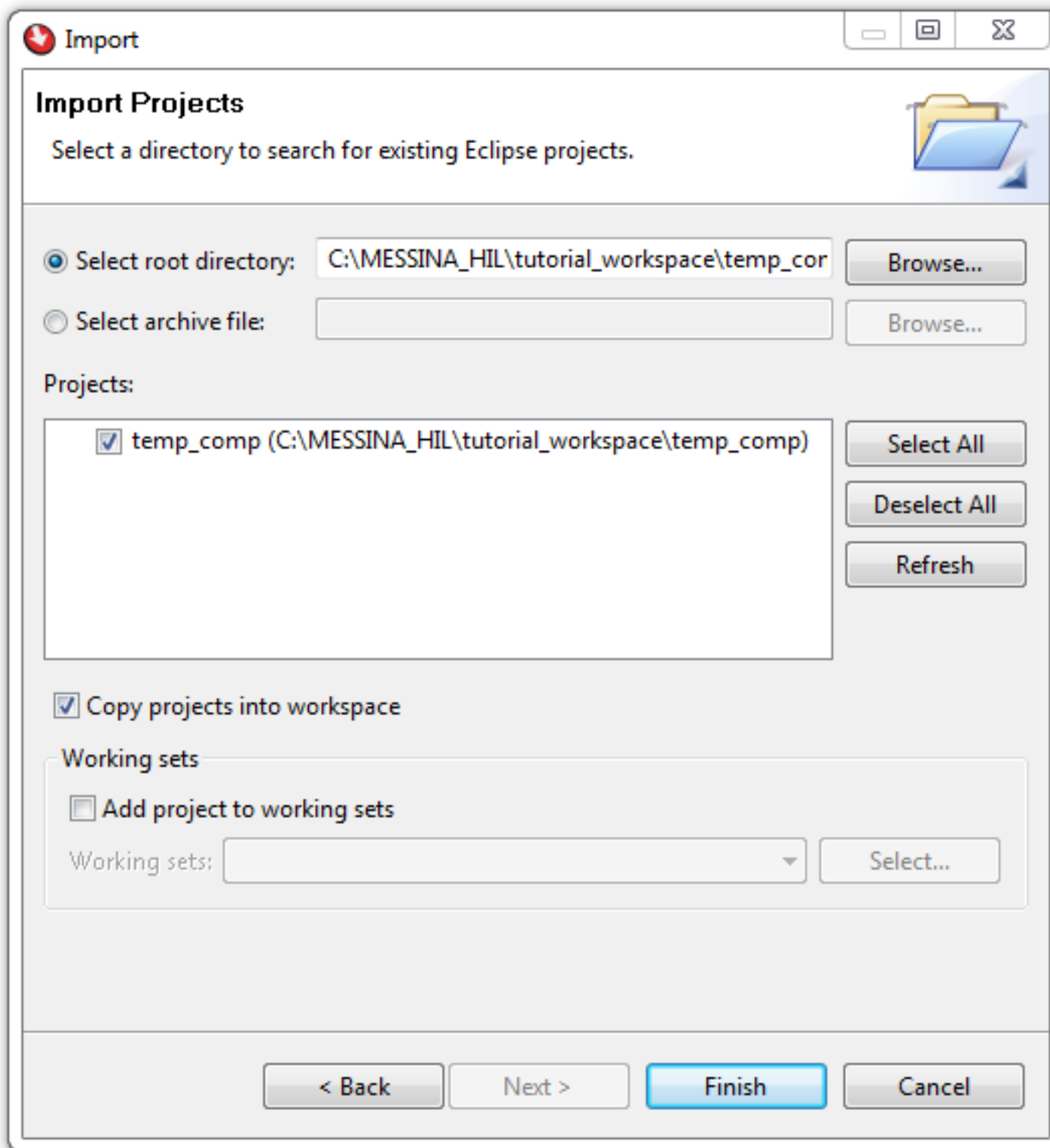
From within the current workspace, select the **Main Menu** → **File** → **Import option** and select **Existing Projects into Workspace** from the **General** folder.



Press the **Next** button. The following dialog is called:



Use the **Browse** button to select the Workspace root directory as shown in the picture below. Alternatively, the project directory can be selected.



Select the desired project from the list, or press the **Browse** button to navigate to the MESSINA workspace where the desired project is located. Activate the check box next to the desired project name in the list.

Important Note: Be sure that the ***Copy projects into workspace*** checkbox is active. This ensures that the project files are copied into the new workspace.

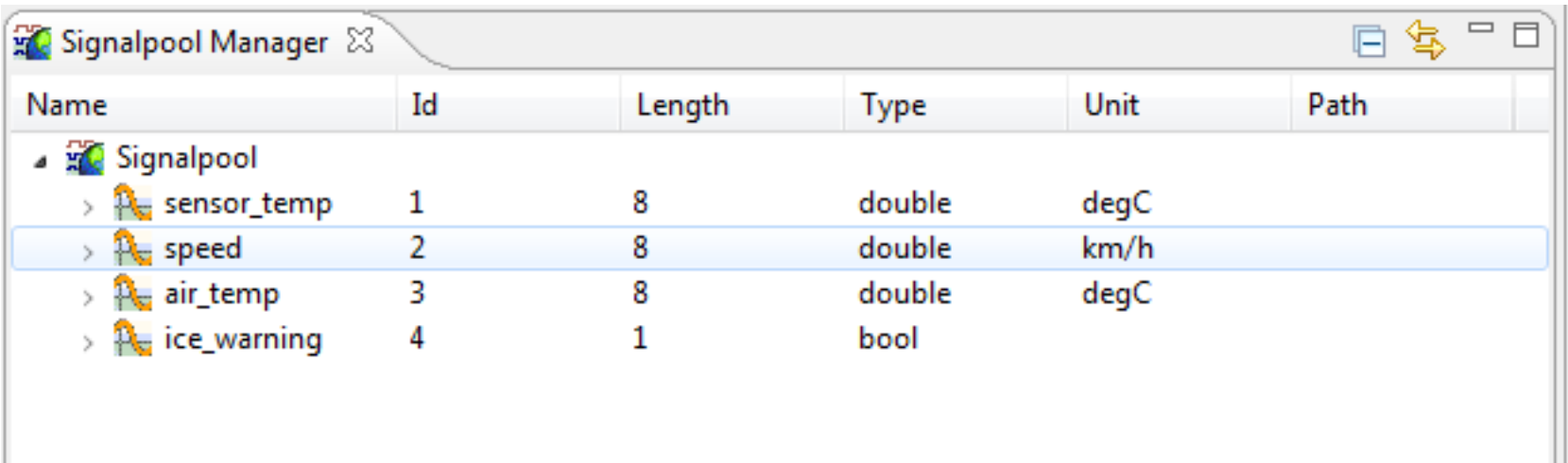
Press the ***Finish*** button.

The project is now visible in the new MESSINA workspace. The original project in the original workspace remains unchanged.

Signalpool Manager

The MESSINA **Signalpool Manager** view is available in the **Configurator** perspective.

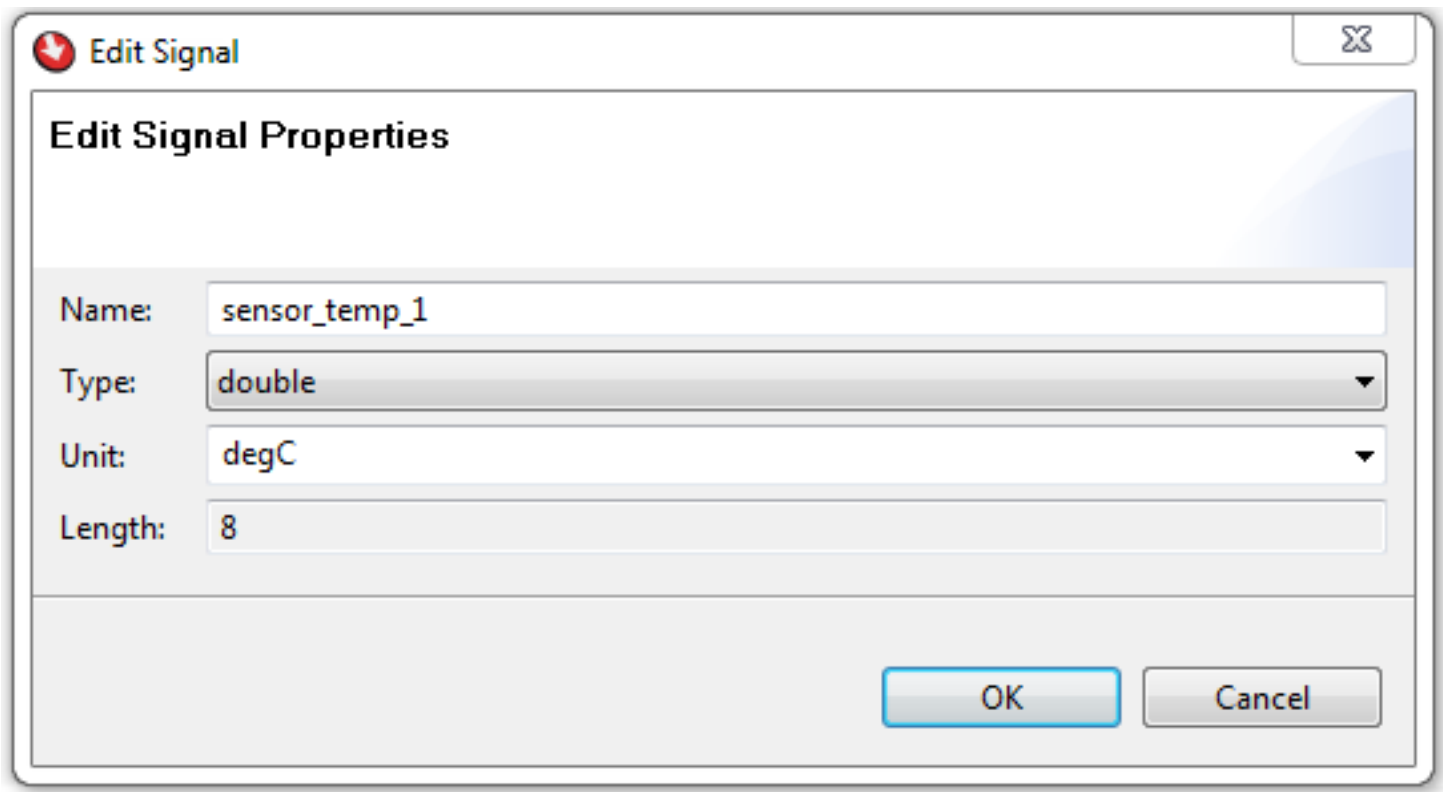
The **Signalpool** is the central point where all MESSINA data is collected and distributed. It is an active process running on the target operating system when a MESSINA test process is executed. After creating a new MESSINA project, the **Signalpool** is empty. Signals that are required for visualization, monitoring, and evaluation must be added to the signalpool. Only signals that interest us need to be mapped to the **Signalpool**.



Name	Id	Length	Type	Unit	Path
Signalpool					
> sensor_temp	1	8	double	degC	
> speed	2	8	double	km/h	
> air_temp	3	8	double	degC	
> ice_warning	4	1	bool		

Add Signals to the Signalpool

A signal can be added to the signalpool by marking the desired port in the expanded view of the **Configuration Manager**. The **Context Menu** → **Map To** → **New Signal** will call the dialog box shown below:



Edit Signal

Edit Signal Properties

Name:

Type:

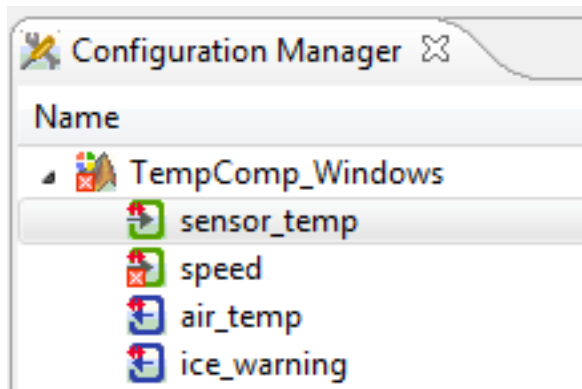
Unit:

Length:

The pull-down lists for **Type** and **Unit** can be used to set the required properties for the signal. The

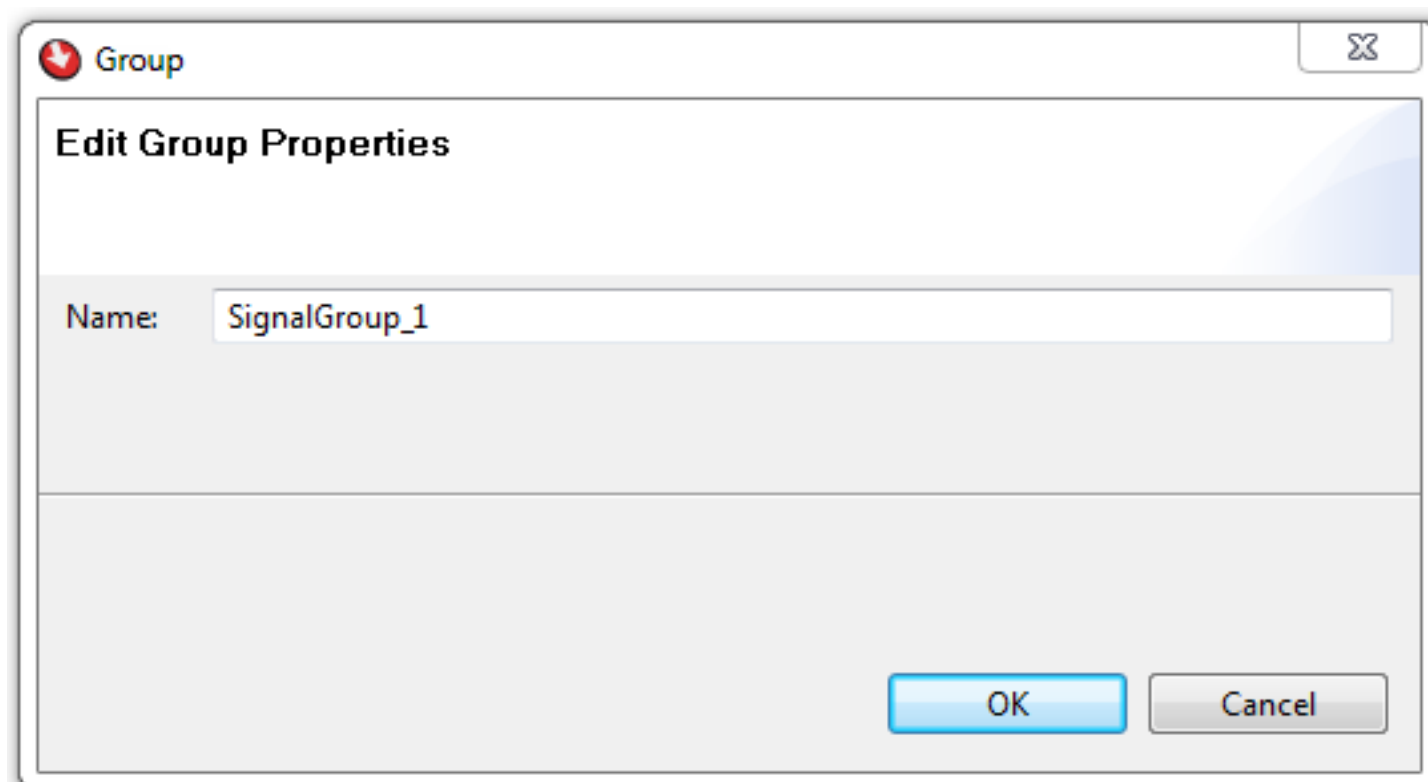
signal name can also be edited if required. It is not required that the signal name assigned here match the name assigned to it in the library. The name assigned here is how the signal will be displayed in the **Signalpool Manager**. Pressing **OK** will create the signal and add it to the **Signalpool**.

Signals can be deleted from the signalpool using the context menu. Signals that have been mapped to a port should not be deleted. Any signals that are mapped (listed in the **Configuration Manager** view) that do not longer exist (because they have been deleted from the signalpool) are marked with an error icon. In the picture below the **speed** signal was deleted from the signalpool.



Add Groups to the Signalpool

For clarity and organization, it is possible to organize signals into **Groups**. Creating a signal **Group** is done using the **Context Menu** → **New Group** option. This can be done either in the **Signalpool Manager** or the [Project Explorer](#) view. The following dialog is called:




The new **Group** is added as a folder to the **Signalpool** tree structure. Individual signals can be added to the **Group** by drag-and-drop.

Groups can be deleted from the **Signalpool** using the context menu **Delete** option.

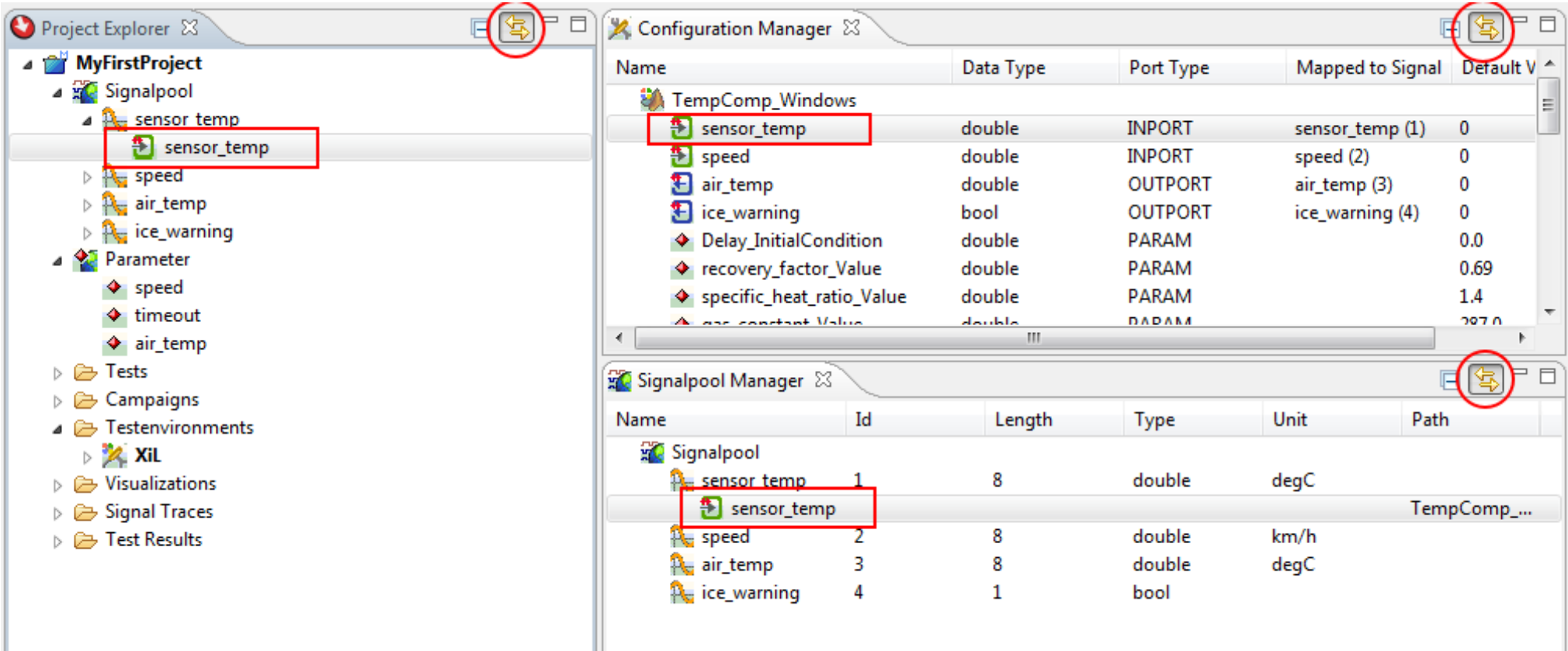
Warning: Deleting a signal **Group** will result in all signals within the group being deleted.

Signal Selection and Synchronization

The different MESSINA perspectives and views offer access to signals and allow signal selection. For example, it is possible to access signals in the **Signalpool** using the **Project Explorer** view, the [Signalpool Manager](#) view in the **Configurator** perspective or the **Signalpool Manager** view in the **Executor** perspective. MESSINA offers a convenient way of synchronizing the selection of signals in different views, and even in different perspectives.

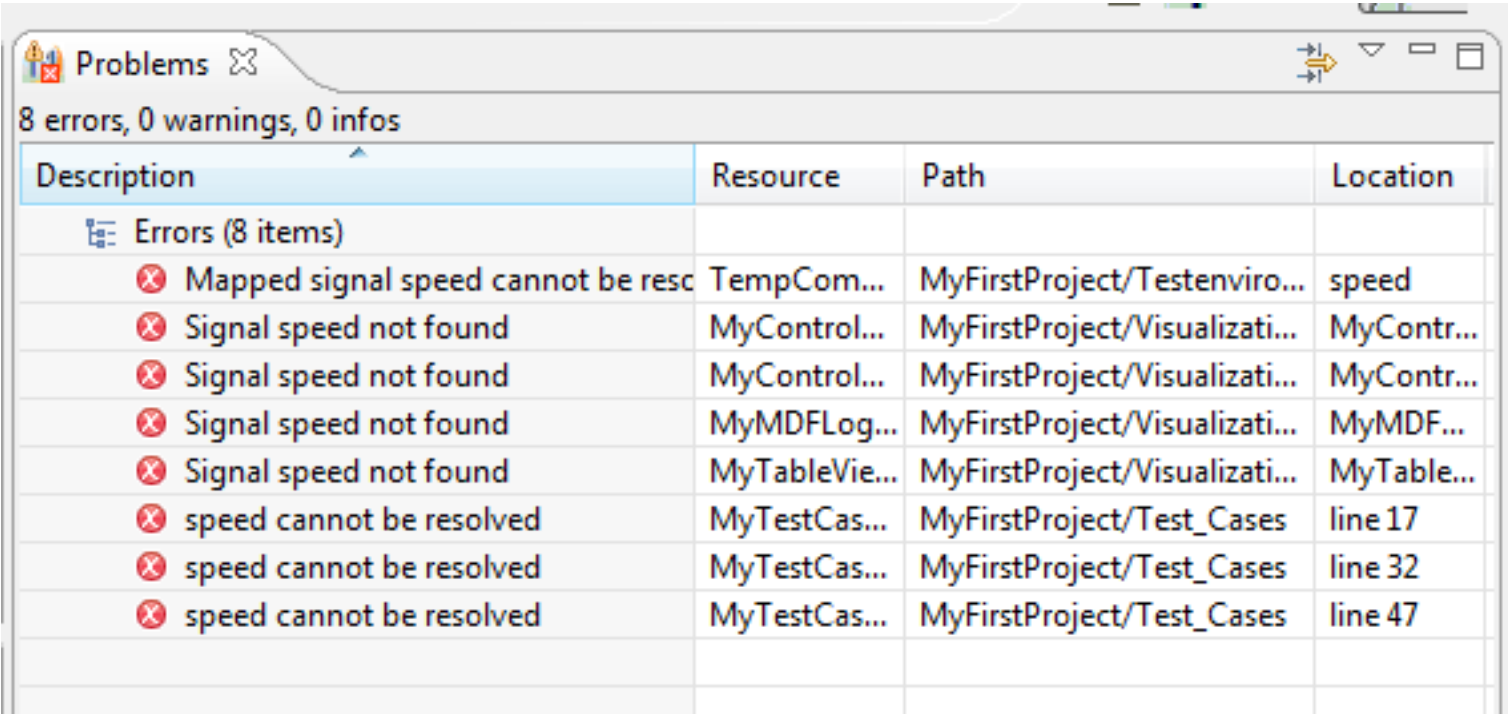
Each view capable of synchronization must first be selected by activating the  icon available at the top right. A signal can then be selected by mouse click, for example, in the **Project Explorer** view. The same signal is also selected in all other views whose synchronization is active, including views in other perspectives.

The picture below shows an example of how the MESSINA **Configurator** perspective is used to select the same signal in the [Project Explorer](#), [Configuration Manager](#), and [Signalpool Manager](#) views.



Problems View

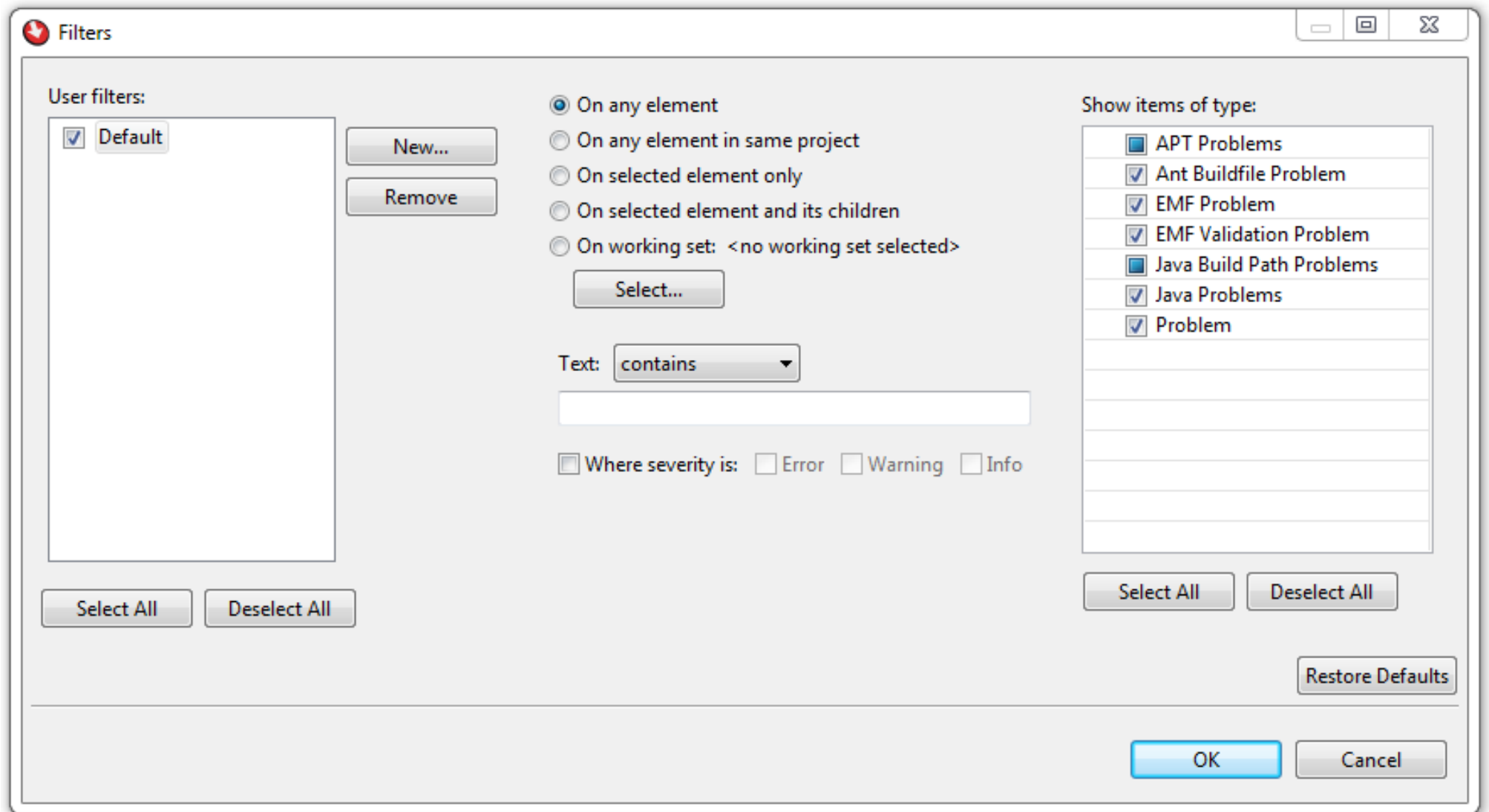
The **Problems** view offers a convenient way of locating and identifying errors and warnings within a MESSINA project. It can be called from **Main Menu** → **Window** → **Show View** → **Problems**. The picture below shows the **Problems** view with several errors and a warning.



Problems 8 errors, 0 warnings, 0 infos			
Description	Resource	Path	Location
Errors (8 items)			
✖ Mapped signal speed cannot be resolved	TempCom...	MyFirstProject/Testenviro...	speed
✖ Signal speed not found	MyControl...	MyFirstProject/Visualizati...	MyContr...
✖ Signal speed not found	MyControl...	MyFirstProject/Visualizati...	MyContr...
✖ Signal speed not found	MyMDFLog...	MyFirstProject/Visualizati...	MyMDF...
✖ Signal speed not found	MyTableVie...	MyFirstProject/Visualizati...	MyTable...
✖ speed cannot be resolved	MyTestCas...	MyFirstProject/Test_Cases	line 17
✖ speed cannot be resolved	MyTestCas...	MyFirstProject/Test_Cases	line 32
✖ speed cannot be resolved	MyTestCas...	MyFirstProject/Test_Cases	line 47

The above picture was generated by deleting the **speed** variable from the **Signalpool** in the temp_comp example project. The errors caused by this action all indicate that **speed** cannot be resolved. Double clicking on the line of an error or warning will result in a jump to the view where the error occurs.

Pressing the  icon (at the top right of the **Problems** view) calls the following dialog:



These settings and options allow an extensive filter capability to be applied to the Problems view. The above picture shows a filter name **Default** that is active and which uses the settings currently being displayed on the dialog. It is possible to configure several filters with different settings if required.

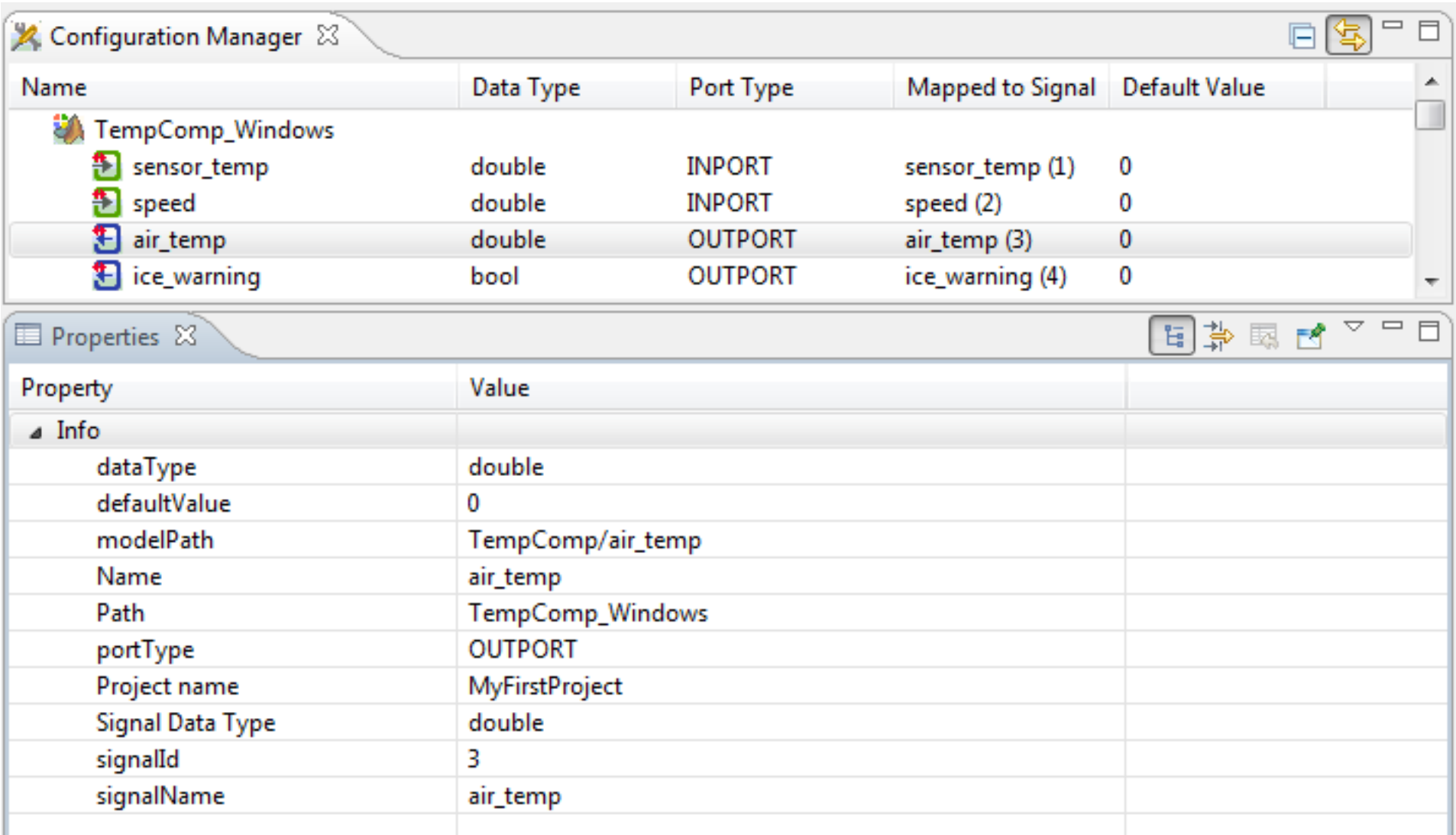
Properties View

The **Properties** view offers a convenient way viewing additional information about MESSINA components such as signals, parameters, test cases etc. It can be called from

Main Menu → **Window** → **Show View** → **Properties**. The **Properties** view appears in the currently active perspective as a separate view.

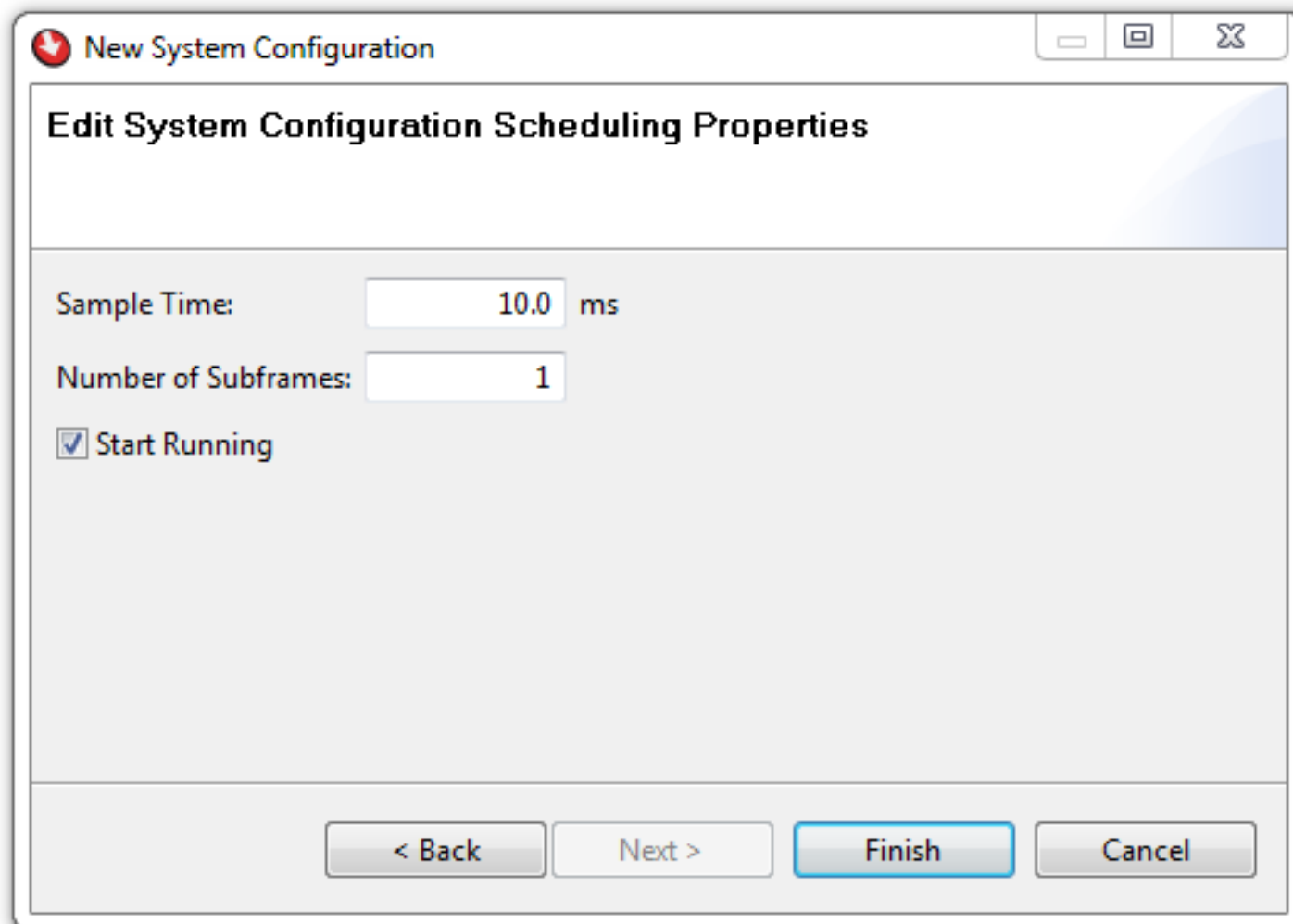
In order to view the properties of a MESSINA component, simply activate the **Properties** view, and select the desired component using the mouse. The available information is displayed when the component is selected.

The picture below shows the **Properties** view. An output port of the **TempComp** model is selected.



Scheduling

In [Add a Configuration to the Testenvironments](#) we learnt that a system configuration must exist in the project before any model and hardware configuration can be added. While adding a configuration we set basic scheduling properties.

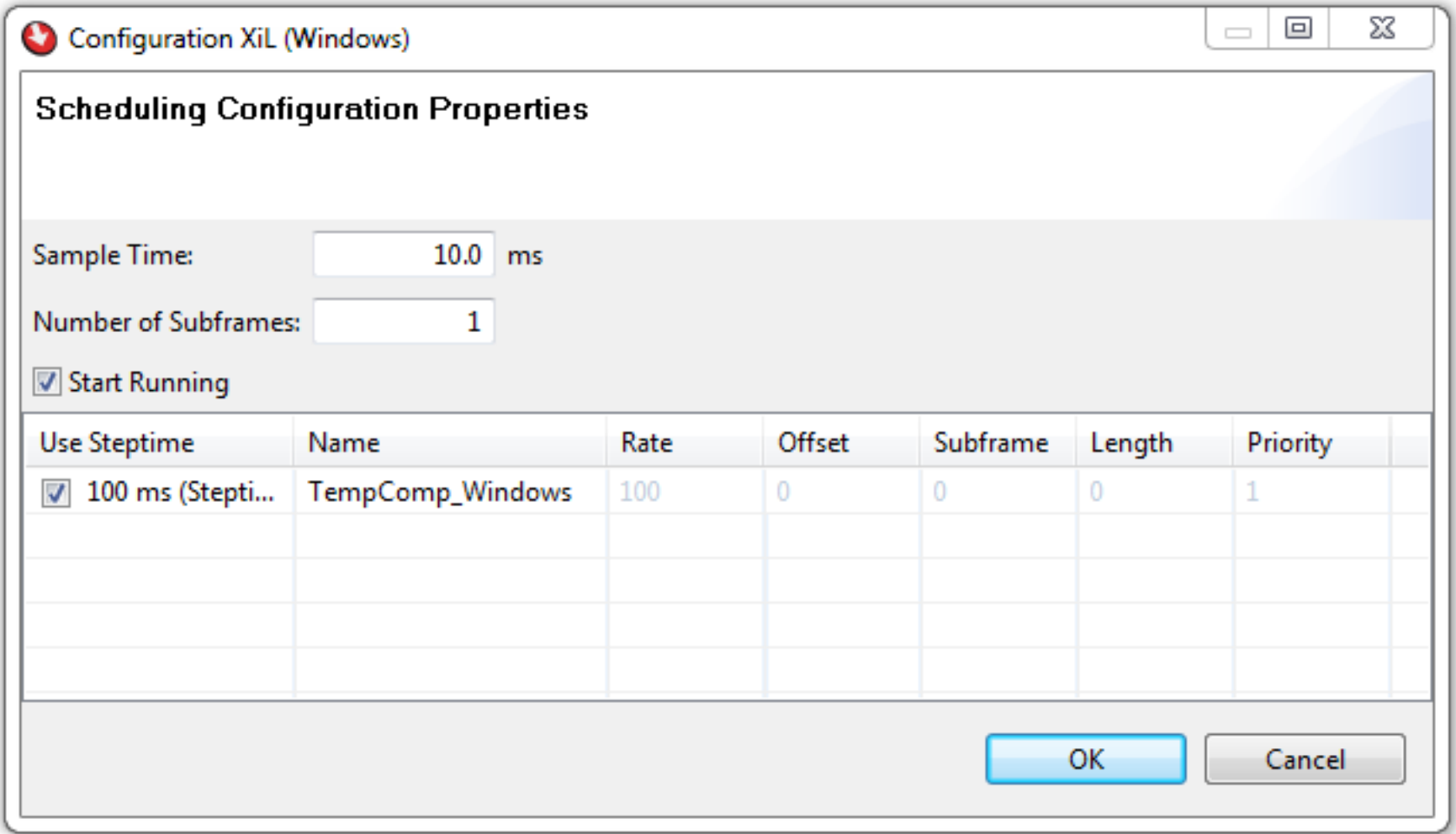


By default the **Sample Time** is 10.0 msec for Windows targets and 1.0 msec for VxWorks targets. The **Number of Subframes** can be set to a number between 1 and 15. **Start Running** is an upcoming feature and checked by default. For now it has to remain activated.

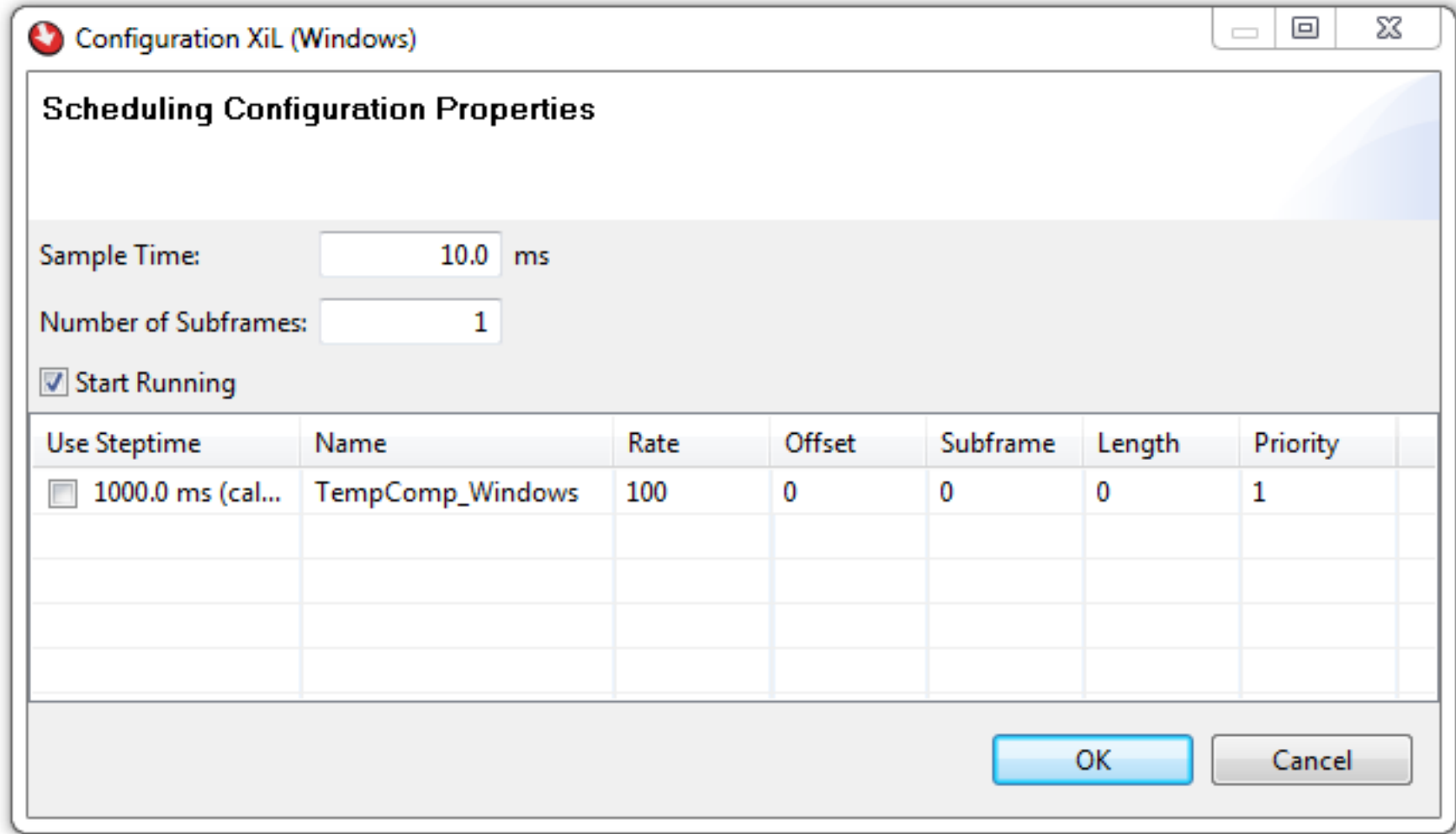
- **Sample Time:** Specifies the duration of a major frame and the temporal resolution in milliseconds. Depending on the scheduling properties of the model or hardware configuration several frames are summarized to a cycle time.
- **Number of Subframes:** A major frame is always divided into sub-frames. They contain the task which shall be processed. Sub-frames are limited to a maximum of 15. At the end of a sub-frame the output will be saved and can be used for the next task.

Scheduling Configuration

To edit your scheduling properties and use additional settings select **Scheduling Configuration...** either in the context menu, in the **Configuration Manager** or in the **Project Explorer** after selecting the desired configuration. Otherwise you can select **Edit** → **Scheduling Configuration...** for the active configuration.



The dialog consist of two parts. The upper part contains the basic properties you have set before. The lower part contains settings for the models or hardware within the configuration. By default the **Steptime** of the model or hardware configuration is selected. **Steptime** has to be set in the Import Wizard. In this mode, you cannot change any settings. To change setting disable the **Steptime**. The dialog will change.



Now you are able to change the settings. It can be useful if you:

1. Need a **Sample Time** less than 1ms for VxWorks. (It is not possible to have a **Sample Time** less than 10 ms in Windows.)
2. Wish to control the exact order of the model and hardware configuration within the **Sample Time**.
3. Want to optimize the cycle time. By dividing the major frame into sub-frames it is possible to use a data exchange at the end of a sub-frame to have the next step using the output instead of waiting for the end of a major frame to get the output.

By adding more model and hardware configurations to the testenvironment, you will have a separate scheduling configuration for each model and hardware configuration.

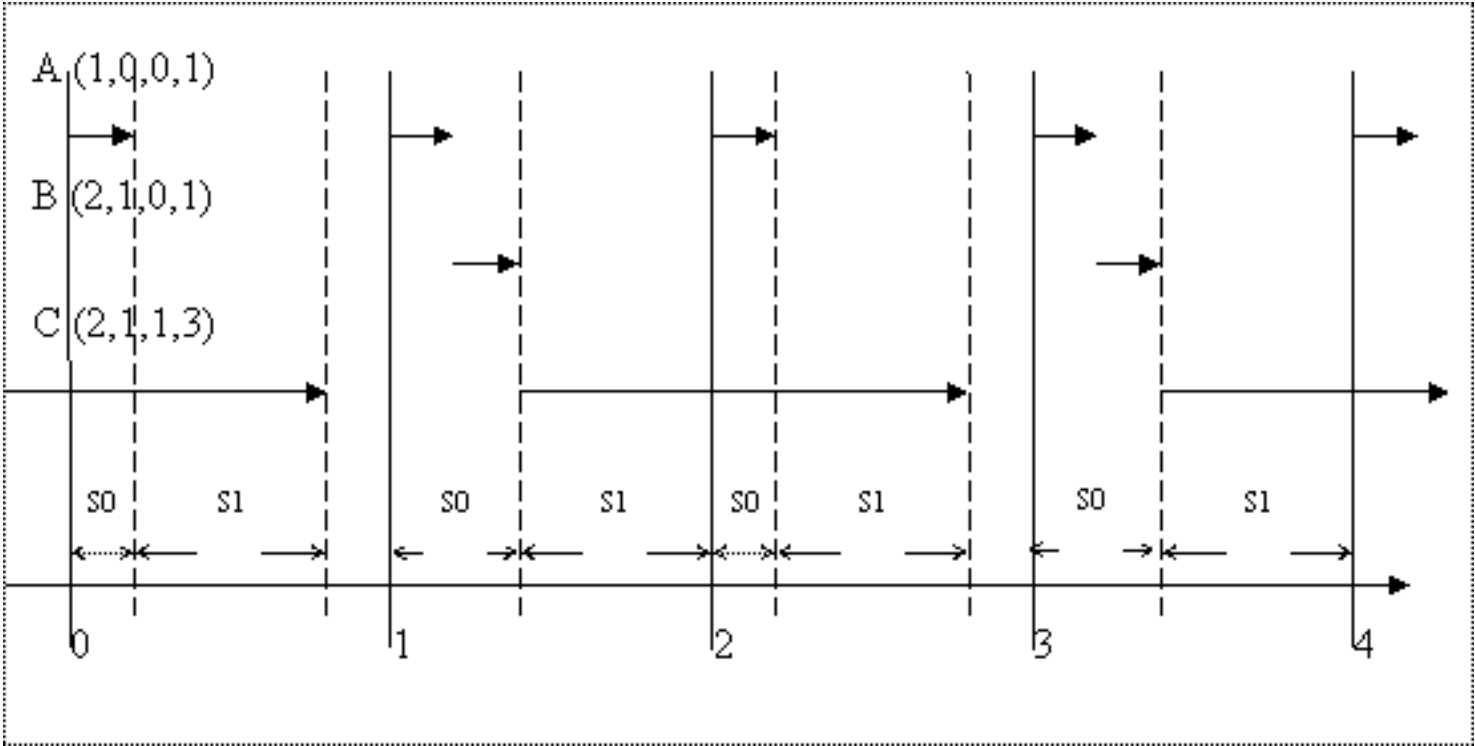
After disabling the default steptime, the calculated steptime equates to the **Sample Time**. If you change **Rate** or **Sample Time** this value will change, too because it is a product of **Rate** and **Sample Time**. (Calculated Steptime = Rate * Sample Time)

Scheduling Parameter

To optimize the cycle time you are able to change the scheduling parameter. Depending on you requirements you can customize the parameters.

- **Rate**: Repetition rate of a frame, E.g. If **Rate** is set to 1, it will be repeated every frame (0,1,2,3,4). If **Rate** is set to 2 it will always skip one frame (0,2,4,6,8). If **Rate** is set to 3 it will always skip 2 frames (0,3,6,9,12).
- **Offset**: This option will move the starting point of a model and hardware configuration to a major frame. If Offset is set to 0 the configuration will start in the first major frame, if it set to 1 or more the start major frame will move to the next major frame within the cycle time. The **Offset** value must be less than the **Rate** value. E.g. Rate 3 Offset 2: 2,5,8,11.
- **Subframe**: It is possible to have an offset for sub-frames, too. It works like the offset for major frames. By this you can arrange the starting point of a sub-frame and move it to a sub-frame. The duration of a sub-frame can not be specified. To give a limitation use **Length**.
- **Length**: This can be used to limit the calculating time of sub-frames. To allow the maximum calculating time set it to 0. Another number you type in set the end of the cycle time to a specific sub-frame. E.g. 10 will let the cycle time end after ten sub-frames. The maximum length can be calculated and is a product of **Rate** * **Number of Subframes**. If a sub-frame need more than the specified length it will cause a real time violation.

Example



Configuration	Rate	Offset	Subframes	Length
A	1	0	0	1
B	2	1	0	1
C	2	1	1	3

This picture will give you an overview about how the scheduling configuration works. The x axis shows the frames of sample time. S0 and S1 are sub-frames and the dashed line shows their starting and ending point. The arrows below the letters describe the duration of the task.

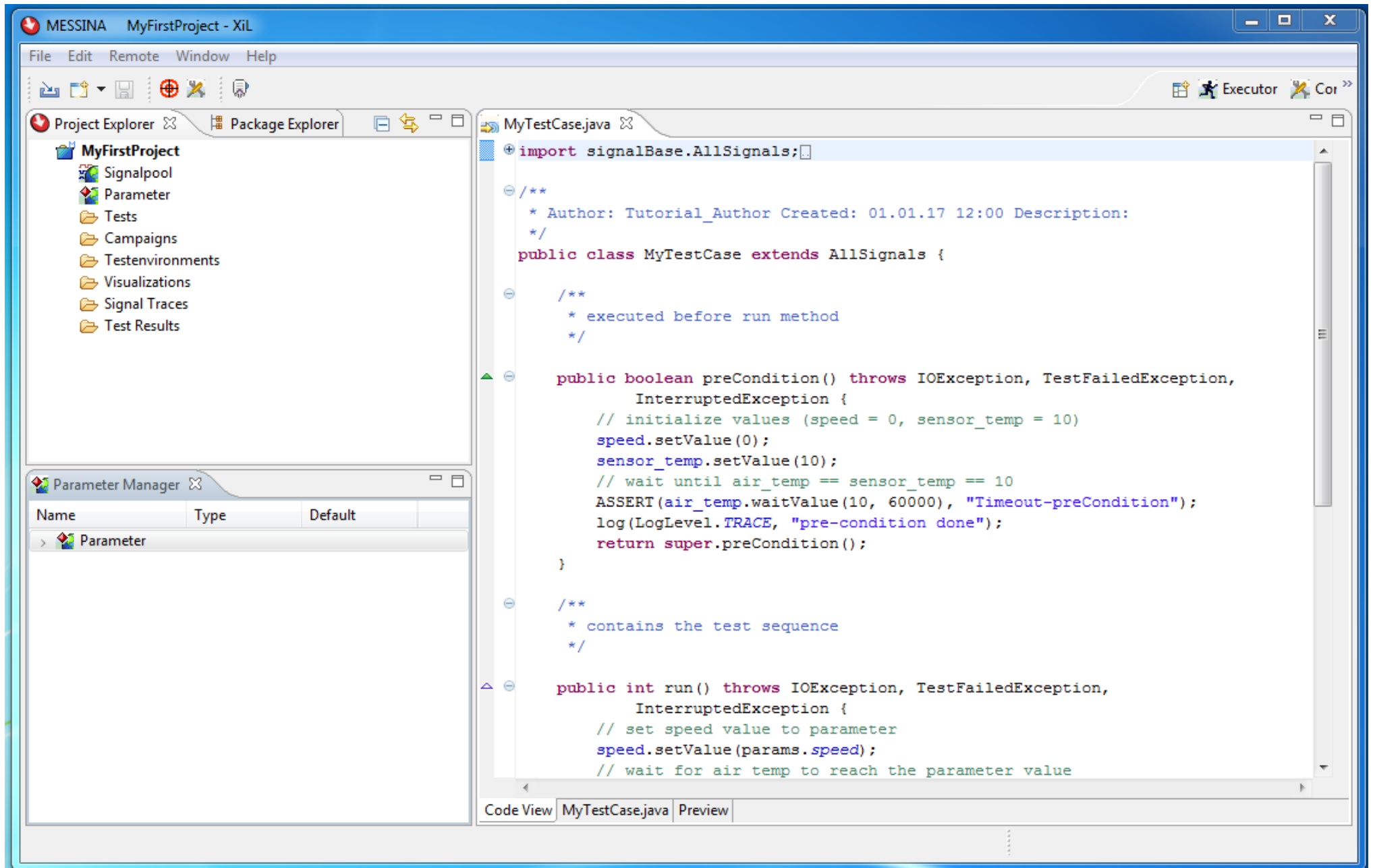
As you can see A is repeated in every major frame and will start in the first one. There is no offset for the sub-frame, so it starts on line 0. The calculation is done within the first sub-frame as it is limited by **Length**.

B will always skip one major frame and start in the second as **Rate** and **Offset** allow. There is no offset for sub-frames, so it will start when A is finished. But in this case B cannot use output of A because there is no end of sub-frame or major frame. We know the output is stored only at the end of major frames or sub-frames.

C will skip a major frame and start in the second major frame as B did. It starts at the beginning of the second sub-frame and can use the output of A and B to calculate. With a length of 3 it has to finish the task within three sub-frames.

Designer Perspective

The **Designer** perspective's main purpose is to create the project, create the **Test Cases/Campaigns** and add **Parameters**. It consists of 3 main views: The [Project Explorer](#), the [Parameter Manager](#) and the [Test Case Editor](#). The individual views are described in detail in the following sections. The following picture shows the default **Designer** perspective.



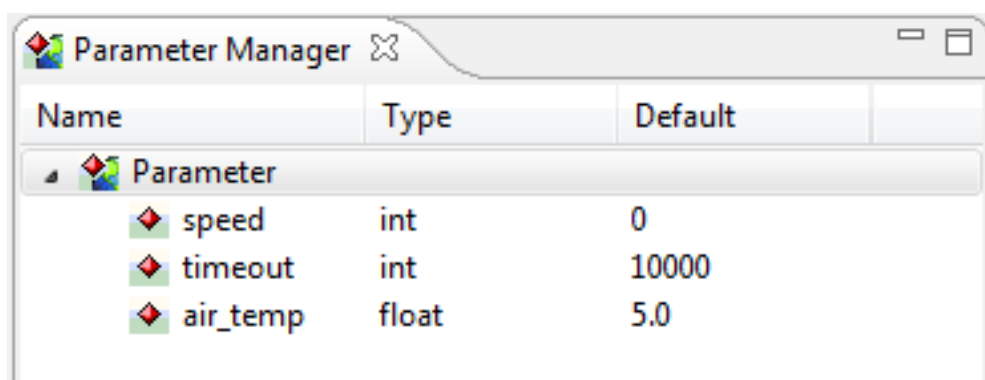
Parameter Manager

The MESSINA *Parameter Manager* view is available in the *Designer* perspective.

The ***Parameter Manager*** view lists all parameters that are available in the project. These are like global variables that can be used directly when programming tests using the ***Test Case Editor*** in Java script.

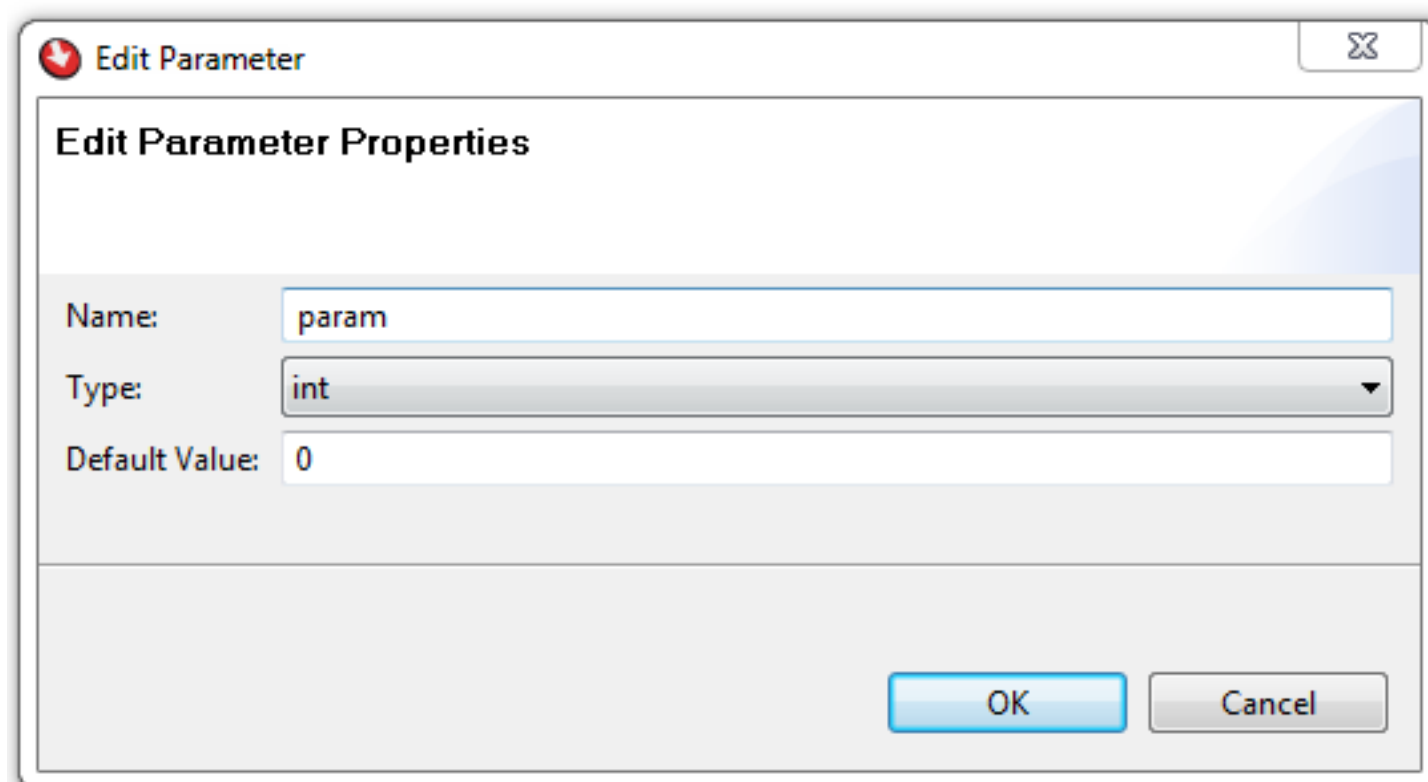
Parameters are a powerful feature of MESSINA because they allow the same **Test Case** to be used over and over again but with different settings. They can be used, for example, to test limits within a test system. The same **Test Case** would be added to a **Campaign** several times but would use different parameters, for example, to test an upper and lower limit.

A picture of the **Parameter Manager** view is shown below.



Add a Parameter

A new parameter can be added to the **Parameter Manager** using the context menu and selecting the **New Parameter** option. The following dialog is called.



A **Name**, **Type**, and **Default Value** are required. The parameter is automatically added to the **Parameter Manager** view.

A detailed description of how to create and use parameters can be found in the [Using Parameters](#) section of this documentation.

Test Case Editor

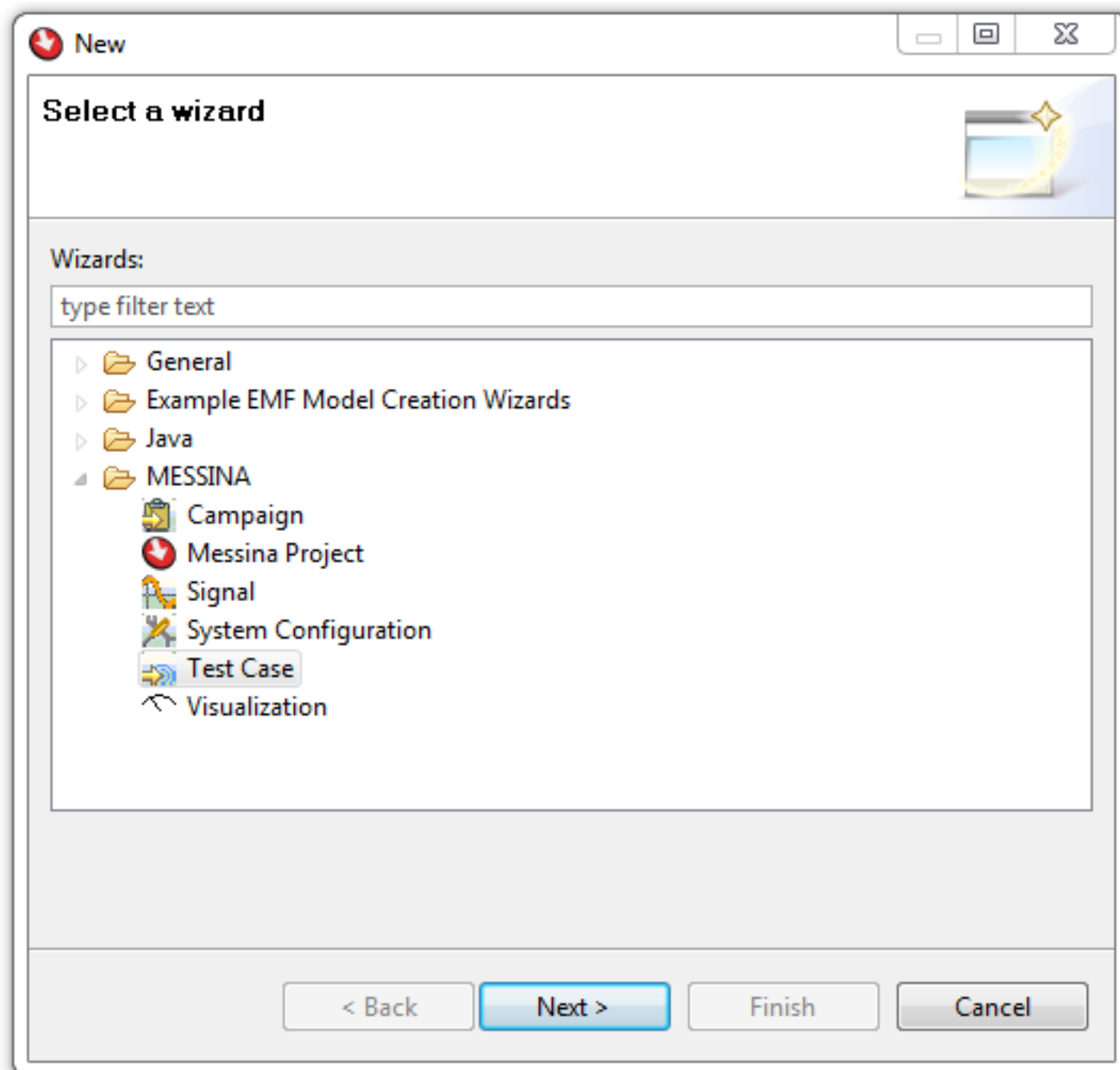
The MESSINA **Test Case Editor** view is available in the **Designer** perspective.

The **Test Case Editor** is a source code editor window which allows the specific functions to be programmed for each test case in a MESSINA project. The test case scripts are written in the Java language.

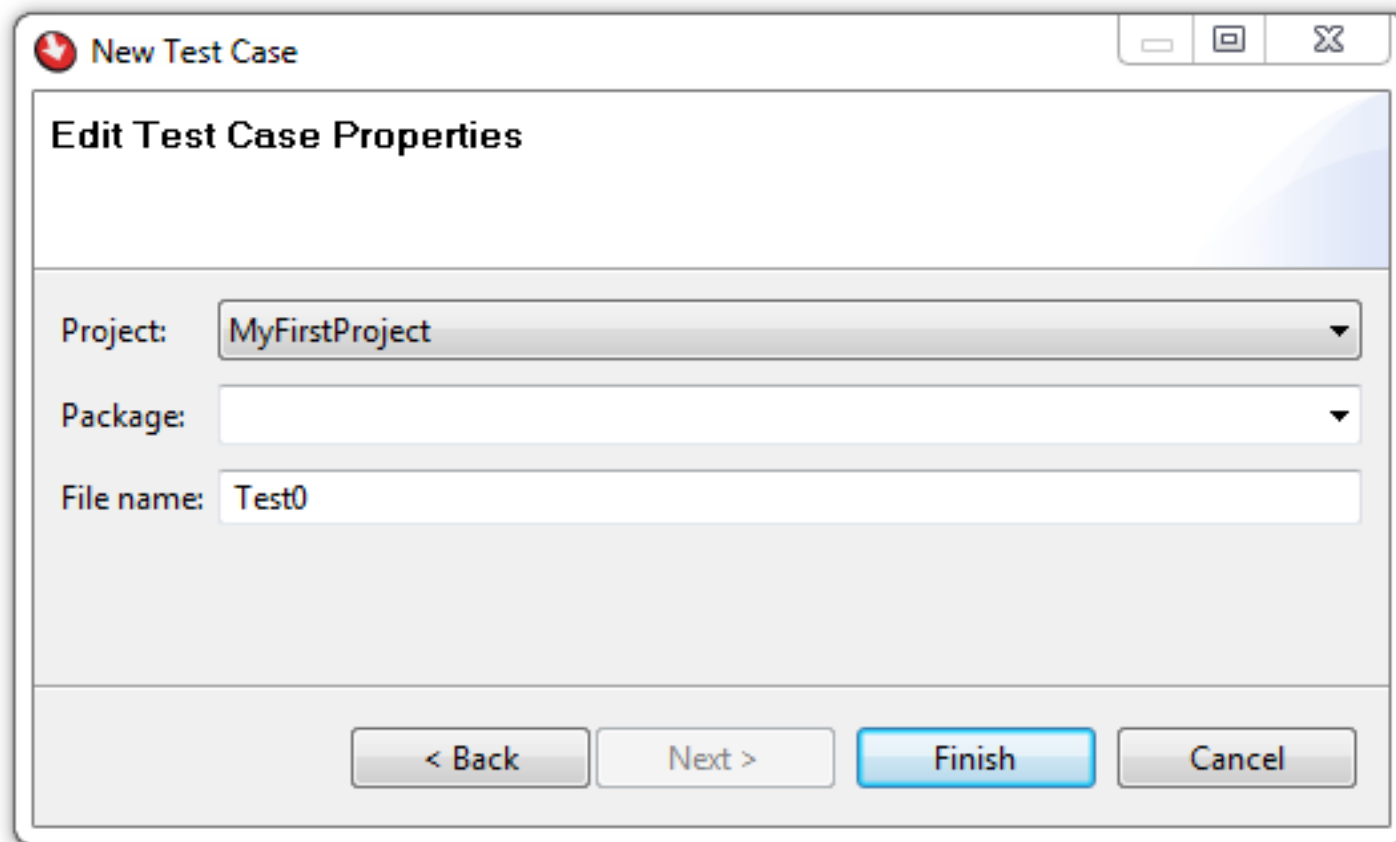
Add a Test Case

The **Main Menu** → **File** → **New** → **Test Case** dialog is used to add a **Test Case**. It is also possible to open the **File** → **New** → **Other** dialog and select the **Test Case** wizard in the MESSINA folder.

Alternatively, the **Test Case Wizard** can be called using the **Context Menu** → **New Wizards** → **Test Case**.



The following dialog box appears when the **Test Case Wizard** is called:



The **Project** pull-down list can be used to select the project where the **Test Case** is to be created. All projects listed in the [Project Explorer](#) view will be available in the pull-down list. The currently active project is automatically set as the selected item. Should **Packages** (groups of Test Cases) exist in the project, these would be listed in the **Package** pull-down list.

The **Package** text box can be used to create and name a **Package** (who would have thought?). A **Package** is used to create a group of **Test Cases**. **Packages** are listed as separate items under the **Test Cases** folder in the **Project Explorer** view.

The **File Name** text box is used to enter the **Test Case** name. This will be the name of the source code file which will be saved in the MESSINA project structure. A sample name is always generated automatically, but this can be changed to any **Test Case** name that does not already exist in the project.

Note: Blank spaces are not allowed in the name.

When a new **Test Case** is created, the source file is created using the name entered above. The **Test Case** is automatically added to the **Test Case** folder in the **Project Explorer** view. The file is automatically assigned a file type ".java" to indicate that it is a java source file. A source code template consisting of an empty class is automatically generated as shown below.


```
*Test0.java
import signalBase.AllSignals;

/**
 * Author: Tutorial_Author
 * Created: 01.01.17 12:00
 * Description:
 */
public class Test0 extends AllSignals {

    /**
     * executed before run method
     */
    public boolean preCondition() throws IOException, TestFailedException,
        InterruptedException {

        // insert the test precondition here

        return super.preCondition();
    }

    /**
     * contains the test sequence
     */
    public int run() throws IOException, TestFailedException,
        InterruptedException {

        // insert the test sequence here

        return 0; //0 means PASSED
    }

    /**
     * executed after run method (also in case of an error / exception)
     */
    public boolean postCondition() throws IOException, TestFailedException,
        InterruptedException {

        // insert the test postcondition here

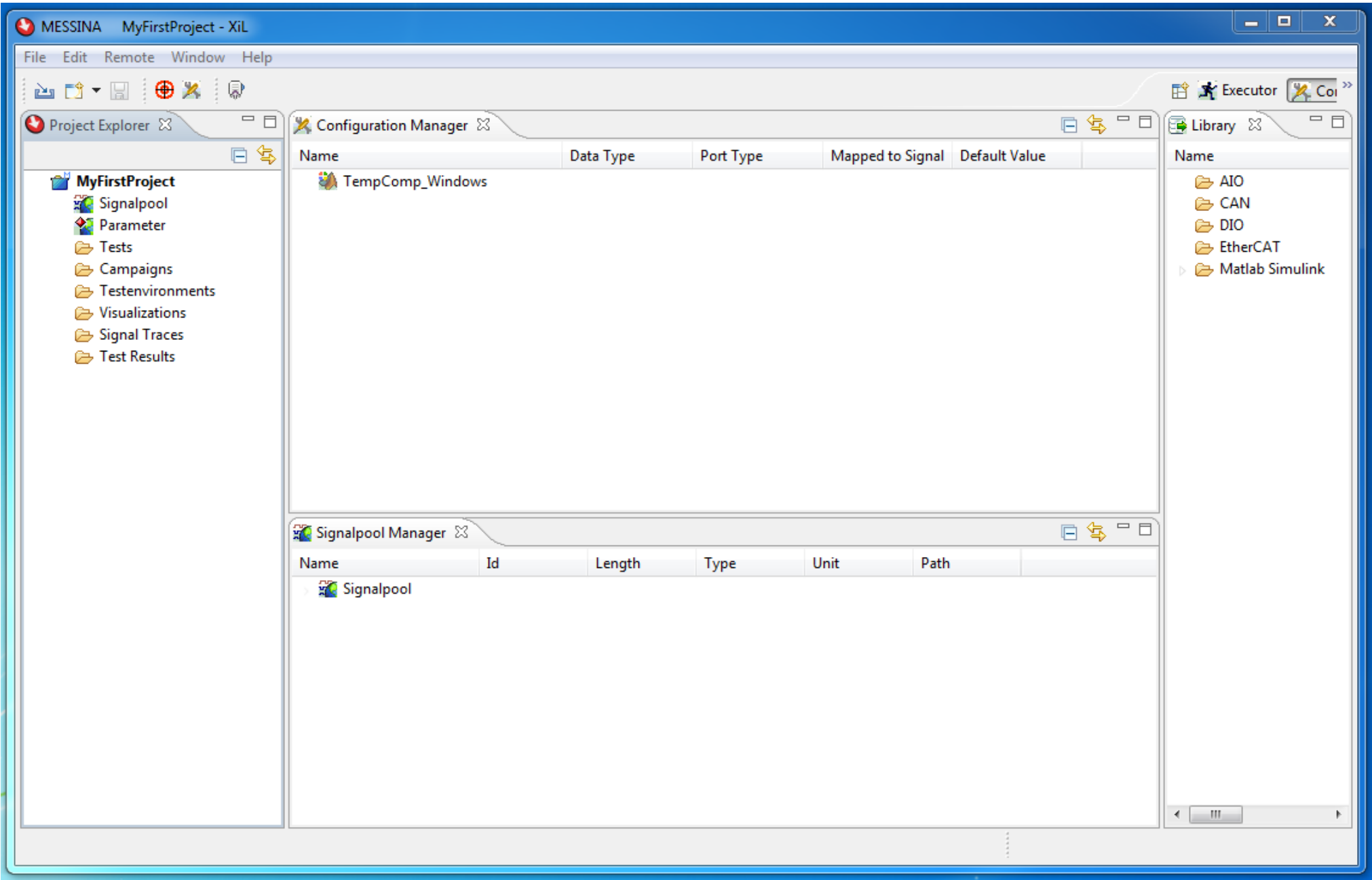
        return super.postCondition();
    }
}
```

Programming the Test Case

A detailed description of **Test Case** programming can be found in the [Programming Test Cases](#) section of this documentation.

Configurator Perspective

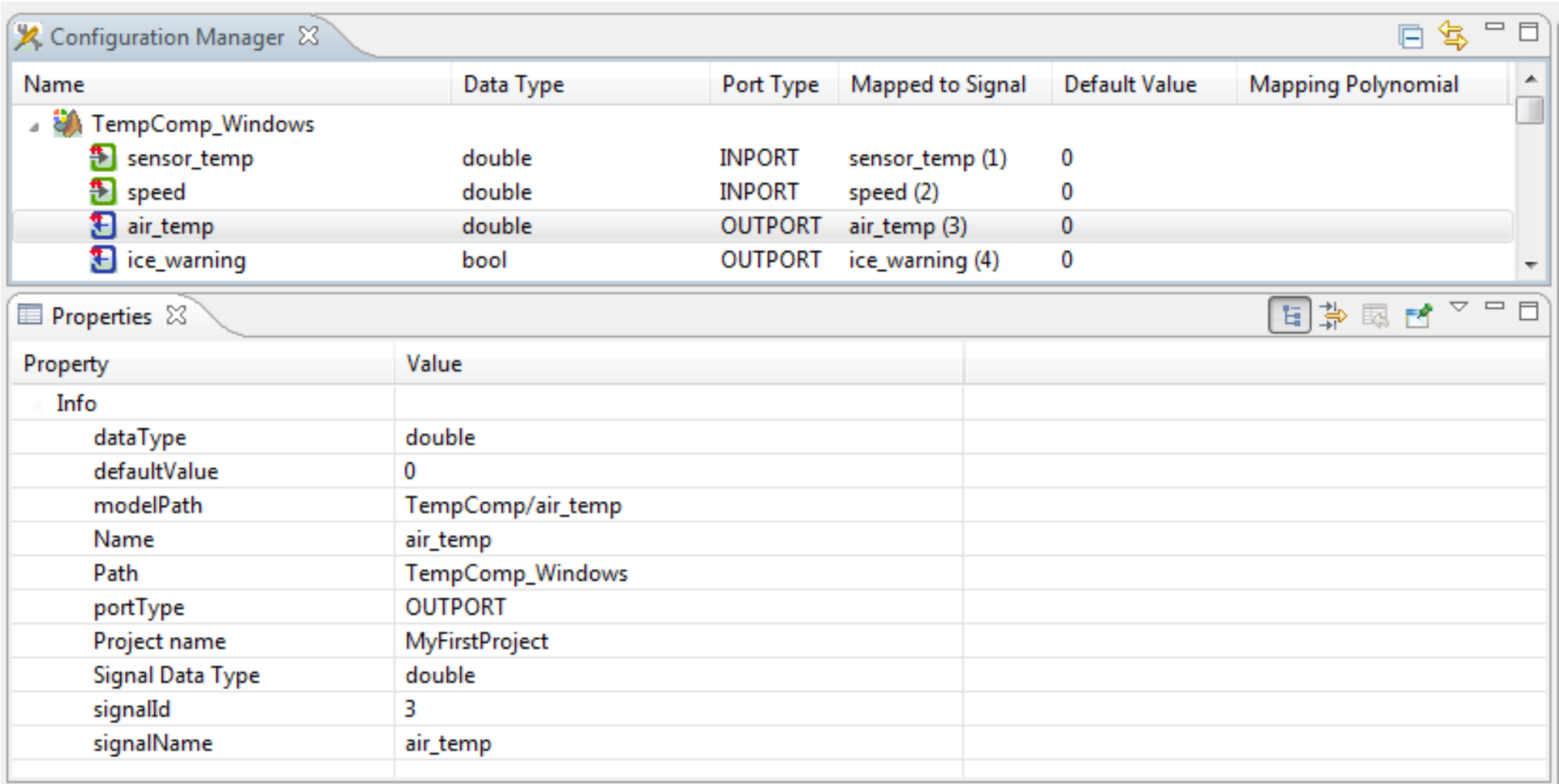
The **Configurator** perspective's main purpose is to configure and setup the individual items within the MESSINA project and to create a test environment. It consists of 5 main views: The [Project Explorer](#) , [Configuration Manager](#), [Library Explorer](#), **Port Manager** and [Signalpool Manager](#). The individual views are described in detail in the following sections. The following picture shows the default **Configurator** perspective.



Configuration Manager

The MESSINA *Configuration Manager* view is available in the *Configurator* perspective.

The *Configuration Manager* lists all models and hardware configurations that are available in the current project. MESSINA projects can contain many different configurations. The properties of the selected configuration is available in the *Properties View*.

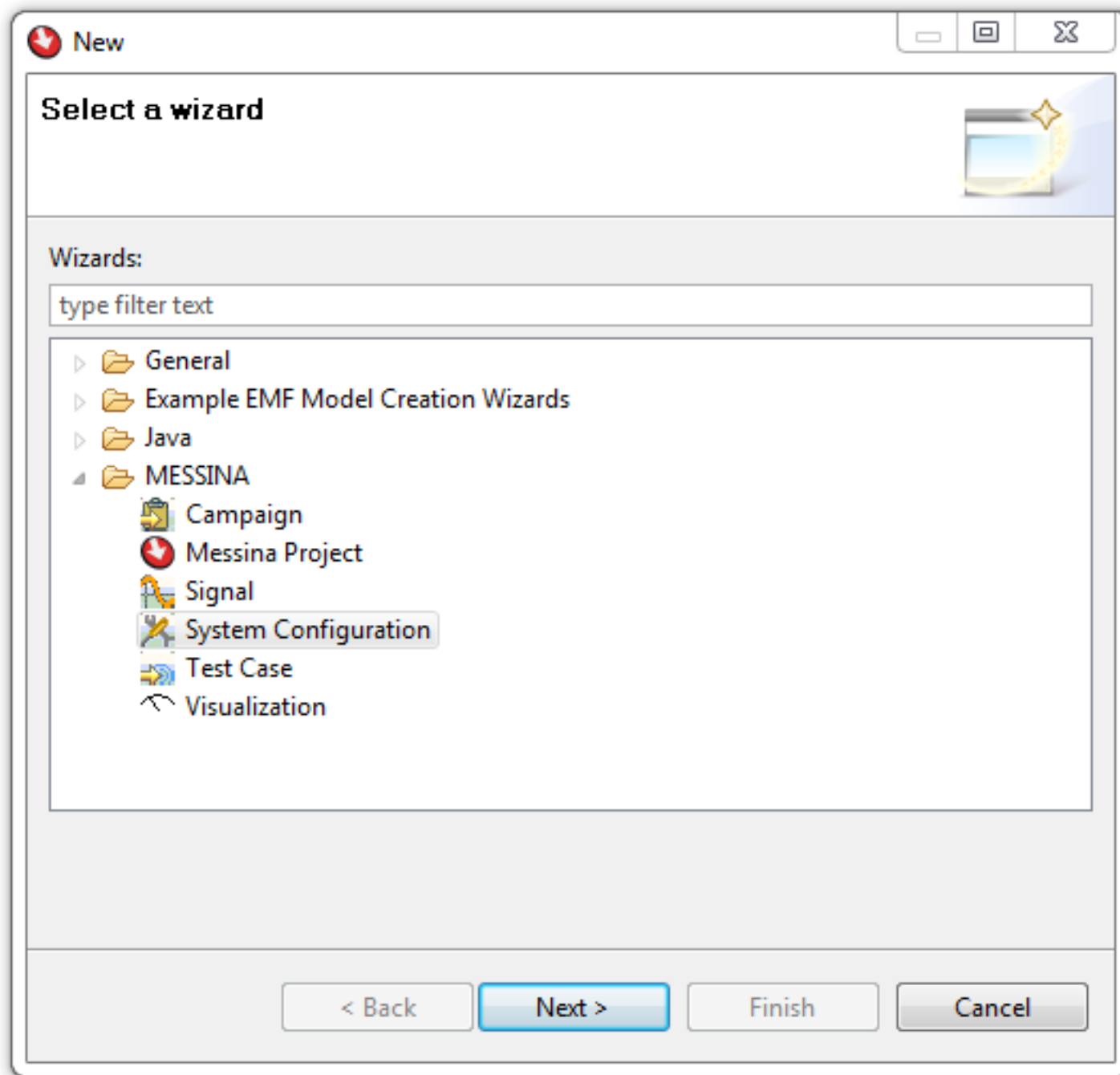


In order to add models or hardware configurations to the *Configuration Manager*, a system configuration must first be added to the *Testenvironments* of the project.

Add a Configuration to the Testenvironments

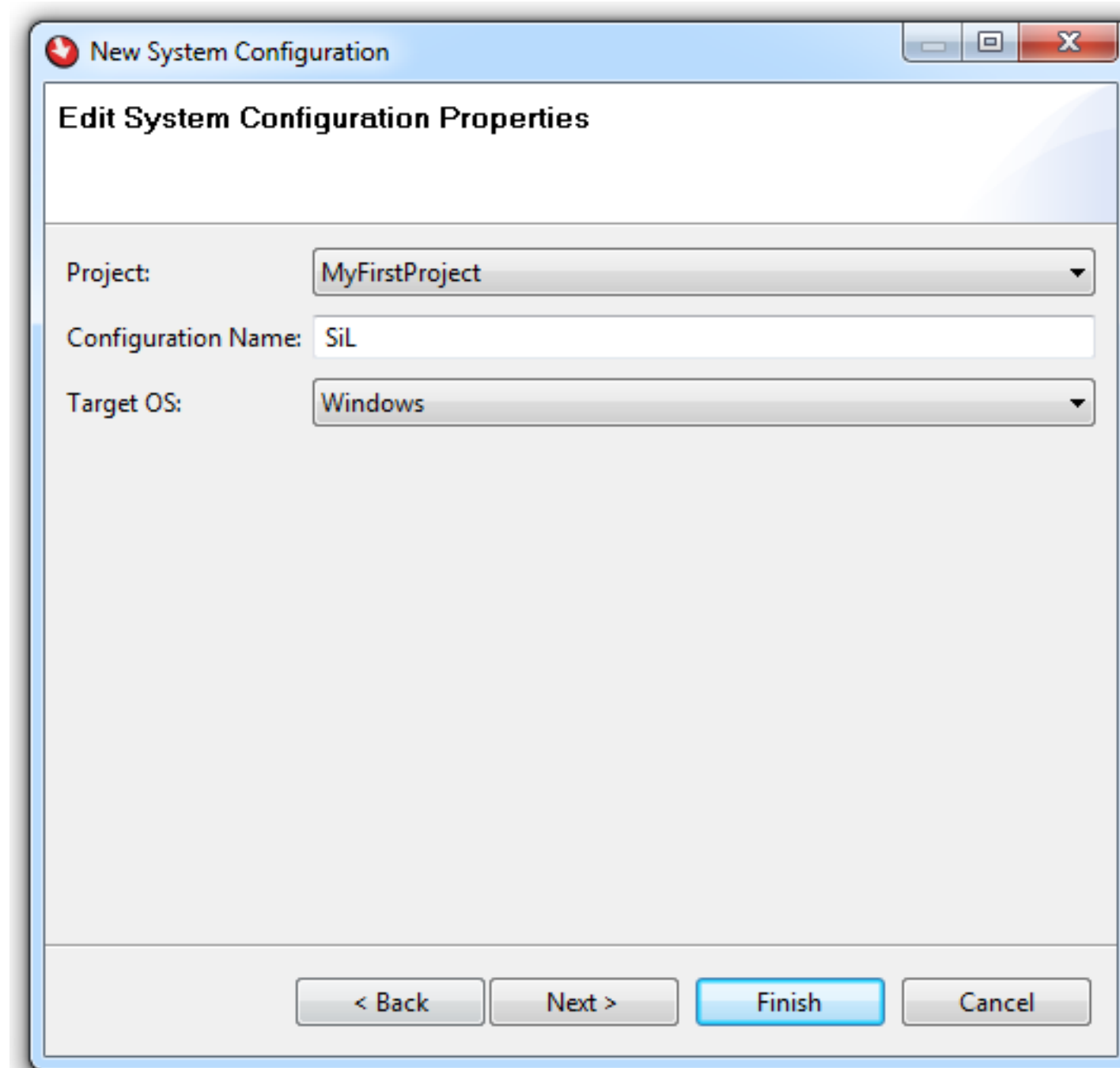
A system configuration must exist in the project before any models, hardware, or signals can be added. Several system configurations can be added to handle different test scenarios for e.g. SiL or HiL.

The Main Menu → File → New → System Configuration dialog is used to add a configuration. It is also possible to open the File → New → Other dialog and select the *System Configuration Wizard* in the MESSINA folder.



Alternatively, the **System Configuration Wizard** can be called using the **Context Menu** → **New Wizards** → **System Configuration**.

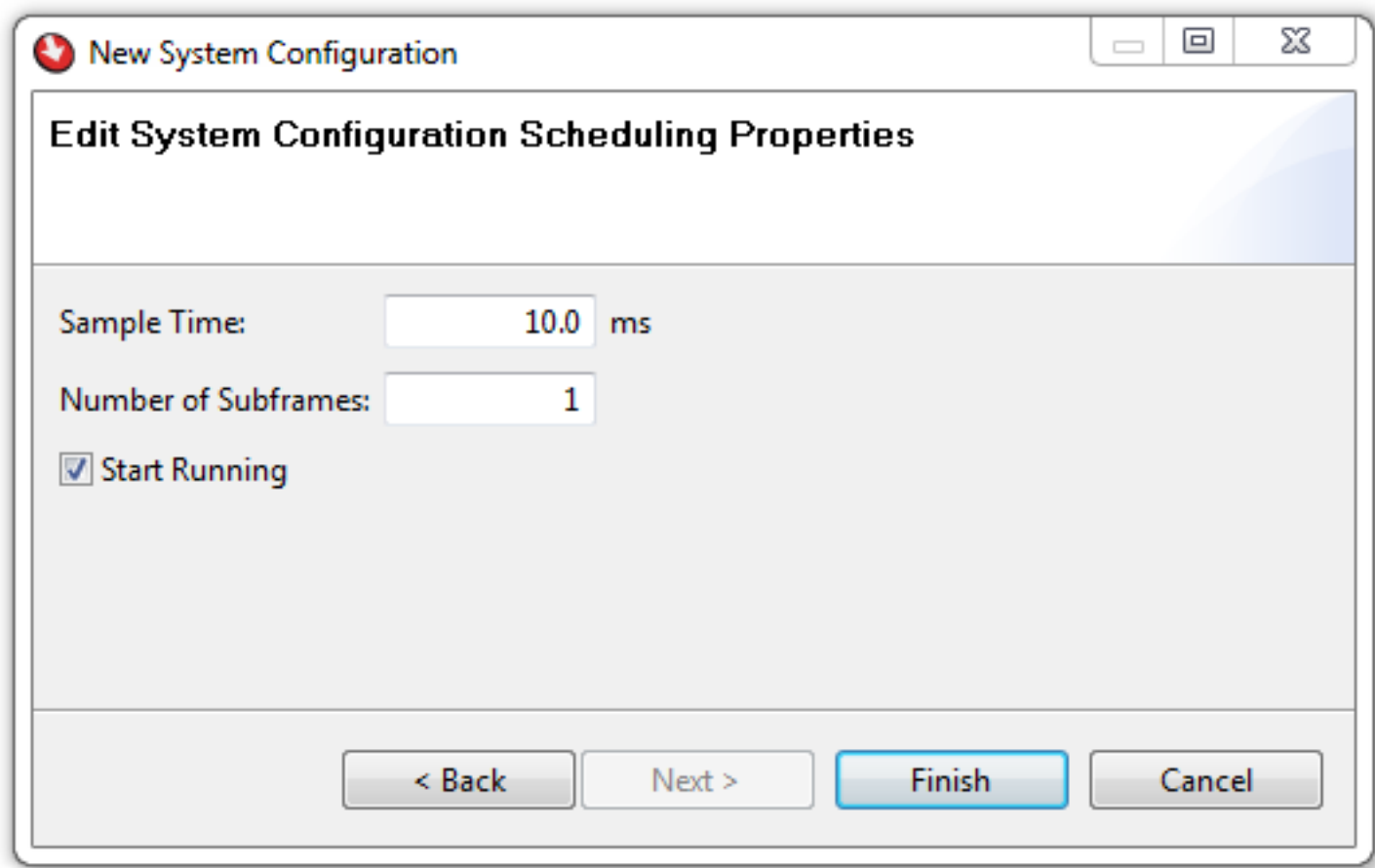
The configuration name is then entered in the following dialog (e.g. SiL):



The **Target OS** can be selected from the pull-down list. Currently supported target systems are VxWorks targets or Windows targets.

Note: The components in the configuration must match the selected target system, in other words only Windows components can be used in a Windows target configuration and only VxWorks components can be used in a VxWorks target configuration.

Click on **Next** to edit the System Configuration Scheduling Properties.



By default the Sample Time is 10.0 ms for Windows target and 1.0 ms for VxWorks target. The **Number of Subframes** can be set to a number between 1 and 15. **Start Running** is an upcoming feature and checked by default. To ensure a faultless function it has to remain activated.

Once completed, the new system configuration is shown in the [Project Explorer](#) in the folder **Testenvironments**.

Once a configuration is available, the required components (e.g. MATLAB/Simulink models or hardware configurations) can be added to the **Configuration Manager** by drag-and-drop from the [Library Explorer](#) view.

You can find expert scheduling configuration in [Scheduling](#).

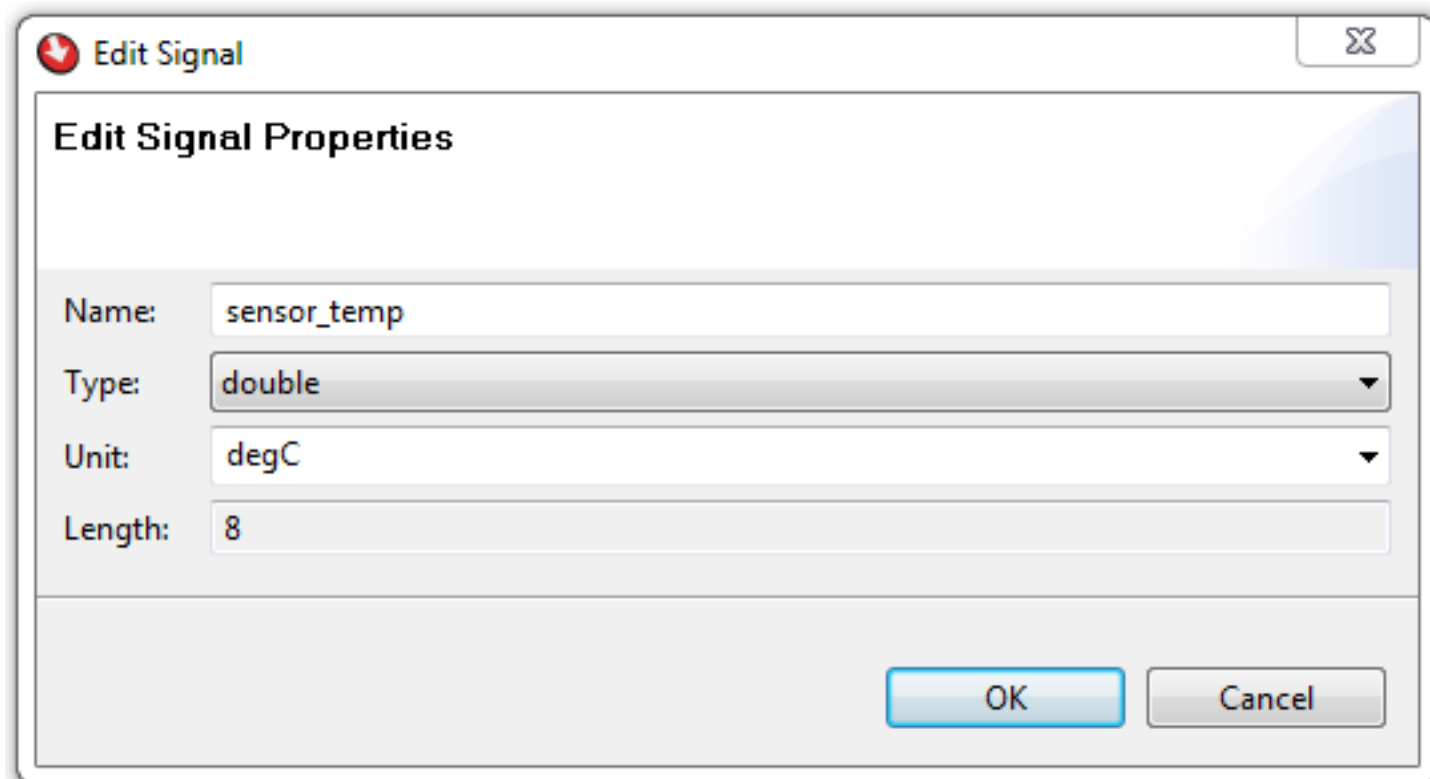
Managing the Ports

Name	Data Type	Port Type	Mapped to Signal	Default Value	Mapping Polynomial
TempComp_Windows					
sensor_temp	double	INPORT	sensor_temp (1)	0	
speed	double	INPORT	speed (2)	0	
air_temp	double	OUTPORT	air_temp (3)	0	
ice_warning	bool	OUTPORT	ice_warning (4)	0	

The **Data Type**, **Port Type**, signal mapping, **Default Values** and **Mapping Polynomials** are also displayed. The **Mapped to Signal** column indicates which signals are mapped to the **Signalpool** and the corresponding **Signalpool** name and ID number. This makes is easy to cross reference any signal

from the **Configuration Manager** to the **Signalpool**.





There are two different ways to map a port from a model or hardware configuration to the **Signalpool**. The first is to use the **Context Menu** → **Map To...** option after the port in the **Configuration Manager** has been selected. The following dialog is displayed:





Pressing the **OK** button will create the signal in the **Signalpool** and map the port to that signal.

The second method is to use drag-and-drop to map the signal from the **Configuration Manager** to the [Signalpool Manager](#) view. In both cases, the port is mapped to the **Signalpool**. The **Configuration Manager** shows the result of the signal mapping in the **Mapped to Signal** column.

An overlay icon (beside the signal name) indicates the current state of the port. The icons and their meaning is as follows:

-  The port is not mapped
-  The port is mapped to a signal in the **Signalpool**
-  The port is mapped, but the mapped signal was not found (deleted)
-  The port is mapped, but the mapped signal has a different data type (warning)

The port type describes the direction of the signal values and is marked using an icon:

-  INPORT: from the **Signalpool** to the component port (Read)
-  OUTPORT: from the component port to the signal (Write)

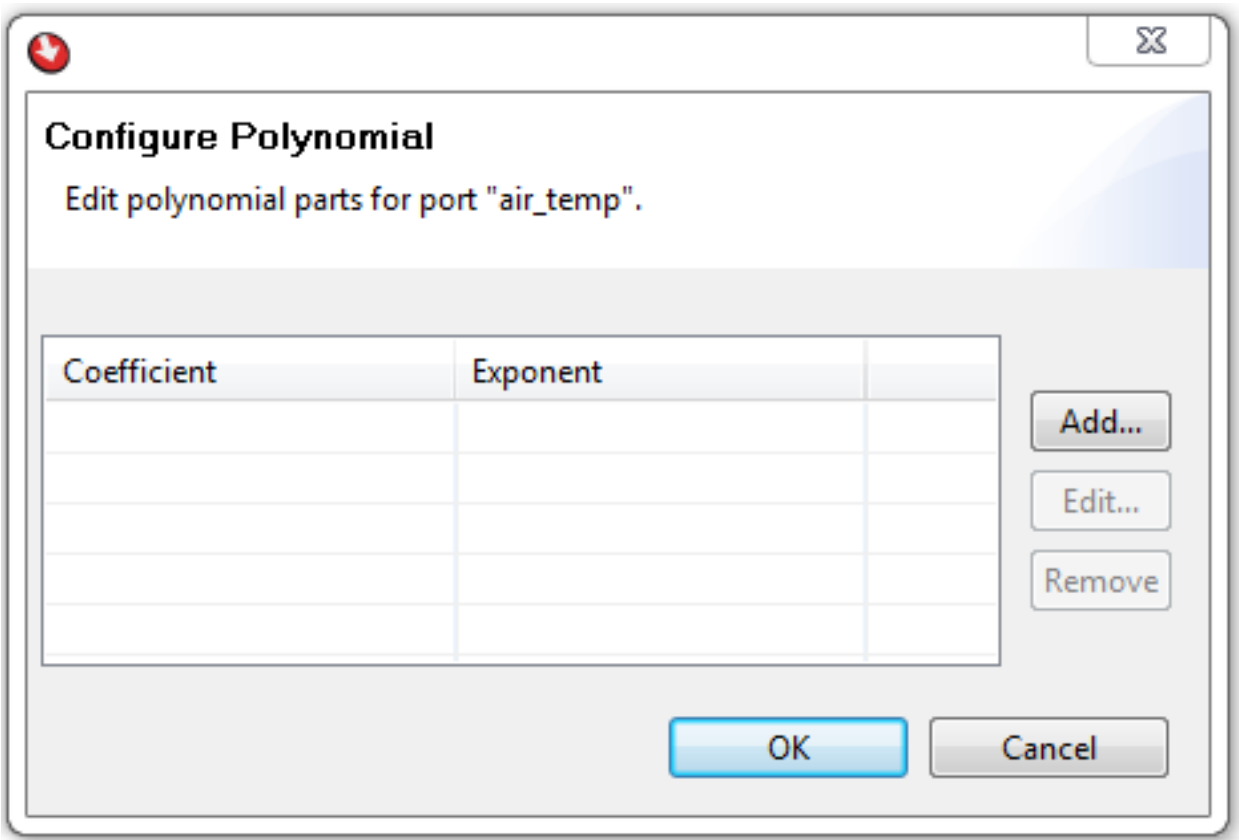
Parameters listed in the **Configuration Manager** view are marked using the  icon.

Warning: Usually ports and signals of the same type are connected. MESSINA supports the type cast from one type to another (except byte arrays). It is possible that individual values will loose precision.

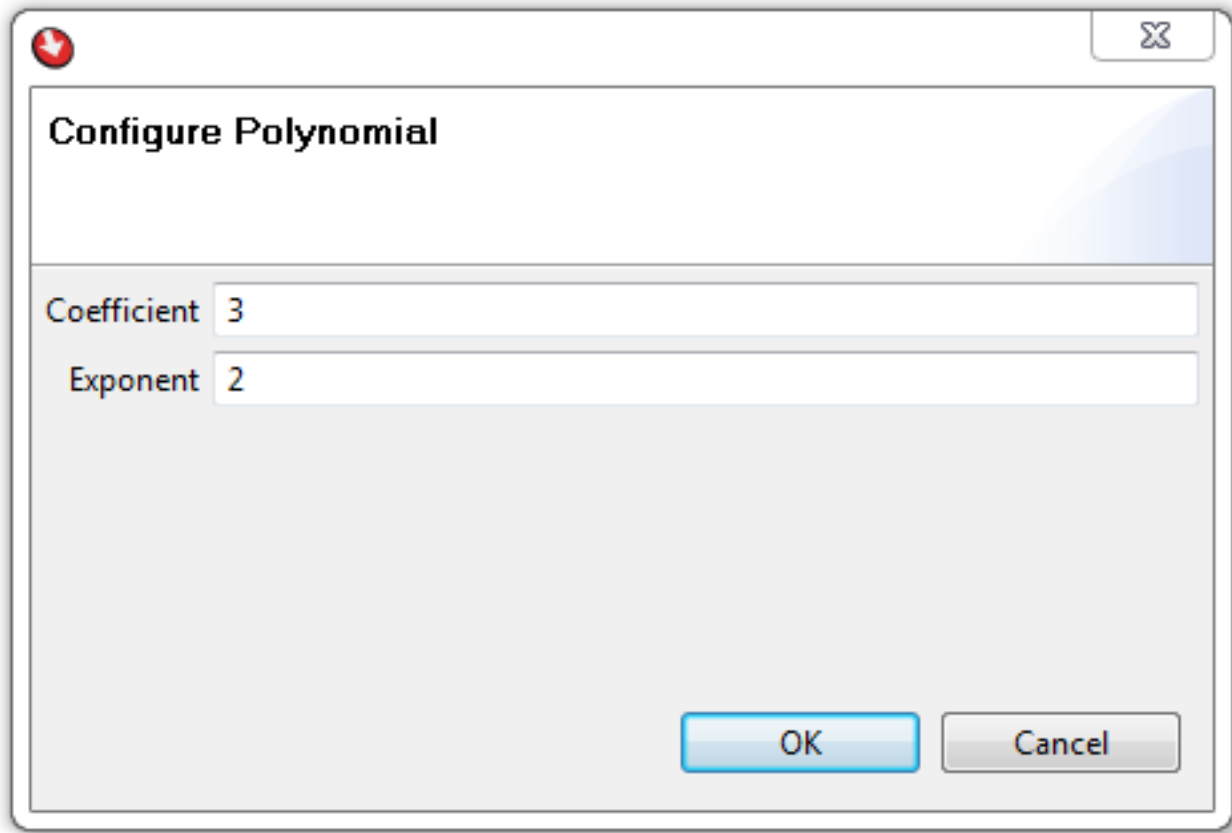
The **Configuration Manager** can also be used to directly select the corresponding signal in the [Signalpool Manager](#) view. This is done by selecting desired signal and selecting the **Context Menu** → **Select Signal in Signalpool Manager** menu item.

A Signal can be modified by a polynomial $y(x)$. At OUTPORTS with value x , the mapped Signal will get the value $y(x)$. INPORTS will get the value $y(x)$ if the mapped Signal gets Value x .

To add a **Mapping Polynomial** select **Context Menu** → **Configure Polynomial...** after selecting a port in the **Configuration Manager**.

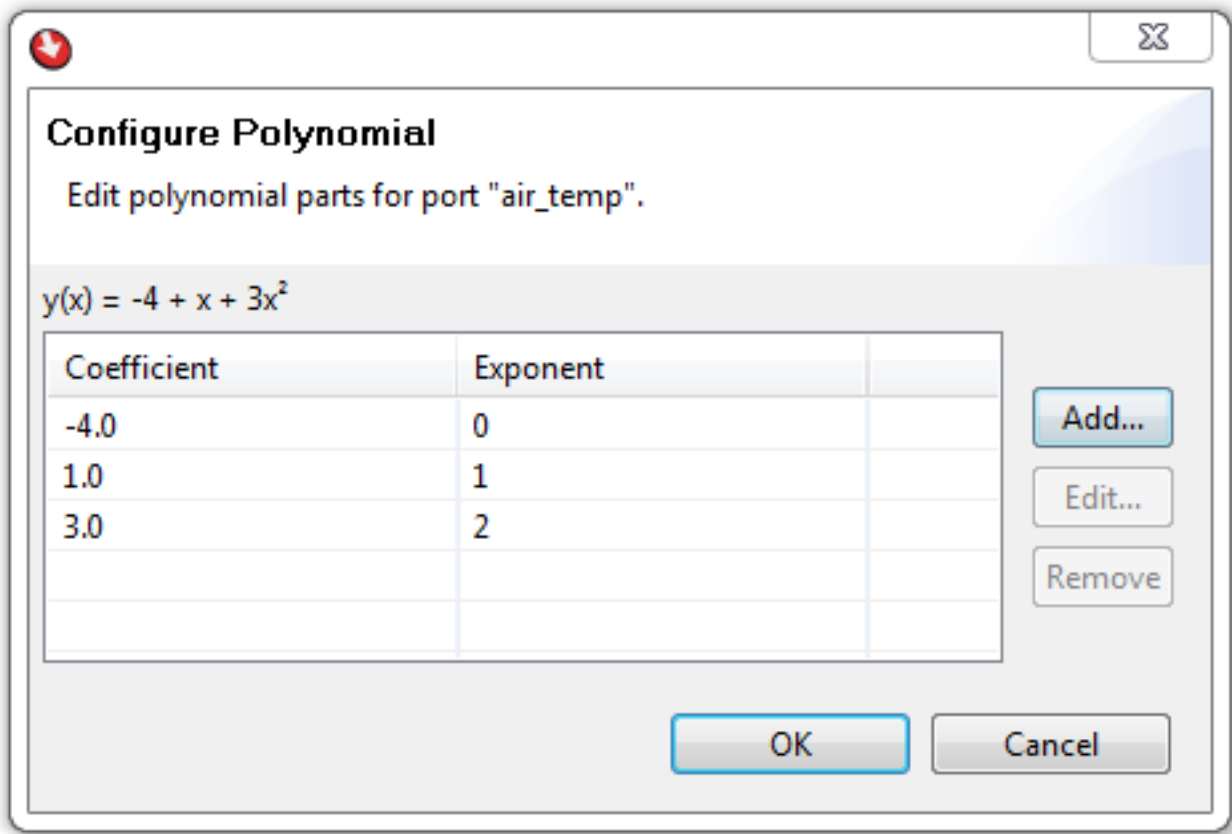


Click **Add...** to modify the polynomial.



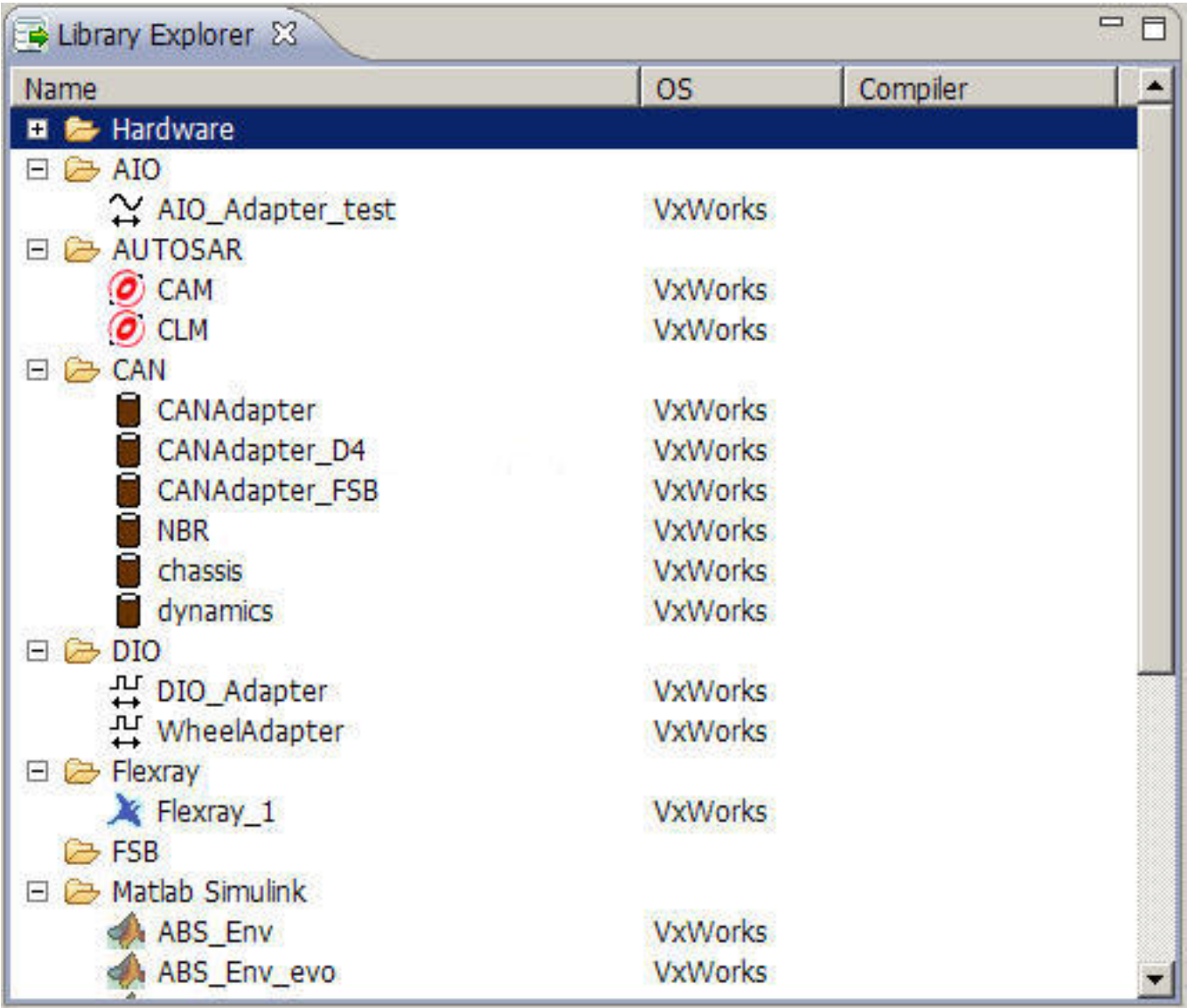
Enter the **Coefficient** and the **Exponent** of the desired polynomial and press **OK**.

Several polynomials with different Exponents can be added. The picture below shows an example for the polynomial $y(x) = -4 + x + 3 \cdot x^2$. For example, if the OUTPORT **air_temp** gets the value 2, the signal **air_temp** will get the value $-4 + 2 + 3 \cdot 2^2 = 12$.



Library Explorer

The MESSINA *Library Explorer* view is available in the *Configurator* perspective. It lists all available models and hardware configurations. Items listed in the *Library Explorer* can be added to the [Configuration Manager](#) through drag-and-drop. A picture of the *Library Explorer* view is shown below:

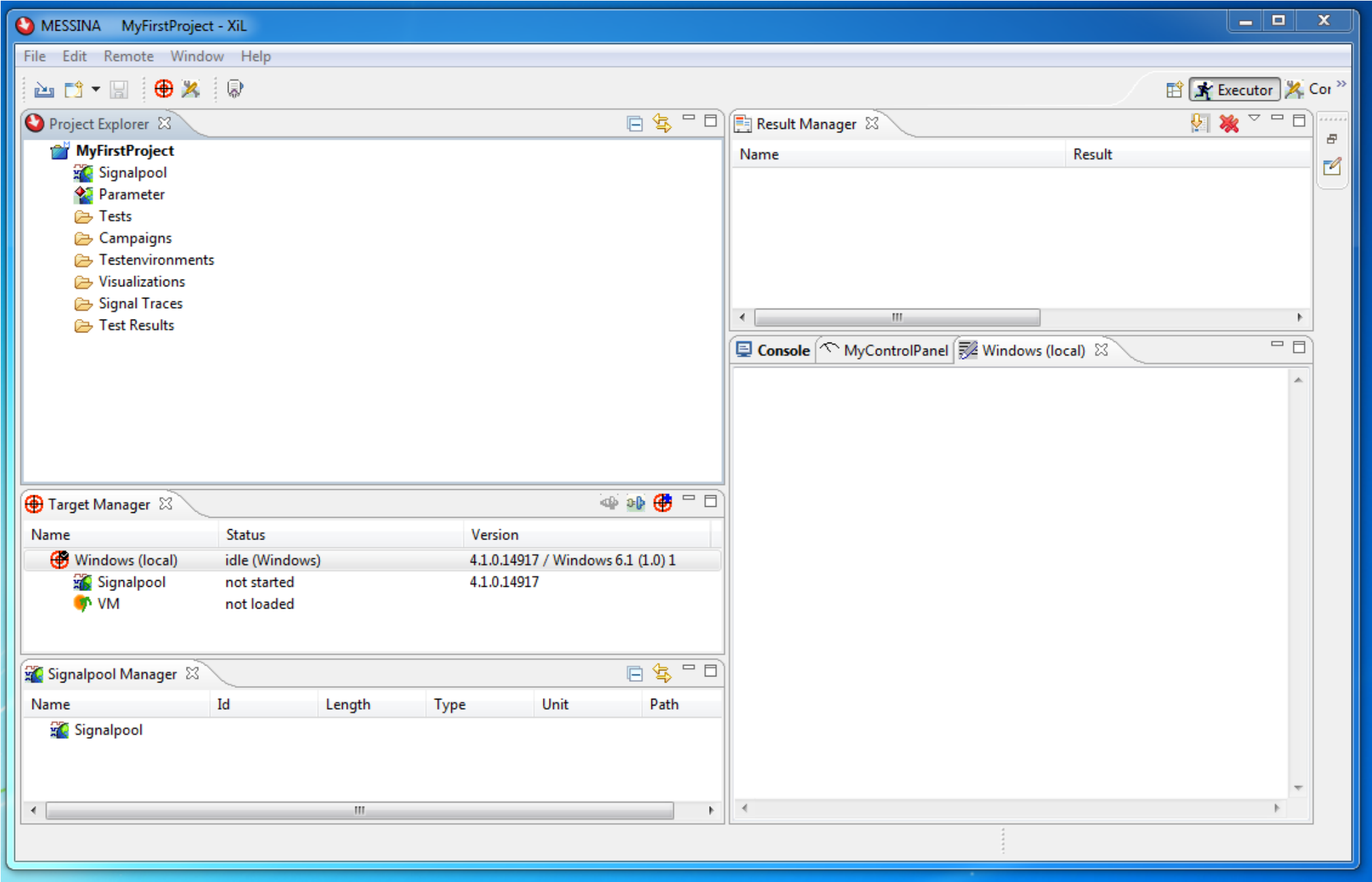


After a new MESSINA installation the library is empty.

It is possible to import models from external tools (e.g. MATLAB/Simulink) into the MESSINA *Library Manager* and make them available for running tests. The device specific documentation (e.g. CAN or MATLAB/Simulink) provides a detailed description of the process of importing a library into the MESSINA development environment.

Executor Perspective

The **Executor** perspective's main purpose is to execute tests and view results. It consists of 5 main views: The [Project Explorer](#) , [Target Manager](#), [Result Manager](#), **Visualizations** and [Signalpool Manager](#). The individual views are described in detail in the following sections. The following picture shows the default **Executor** perspective.



Target Manager

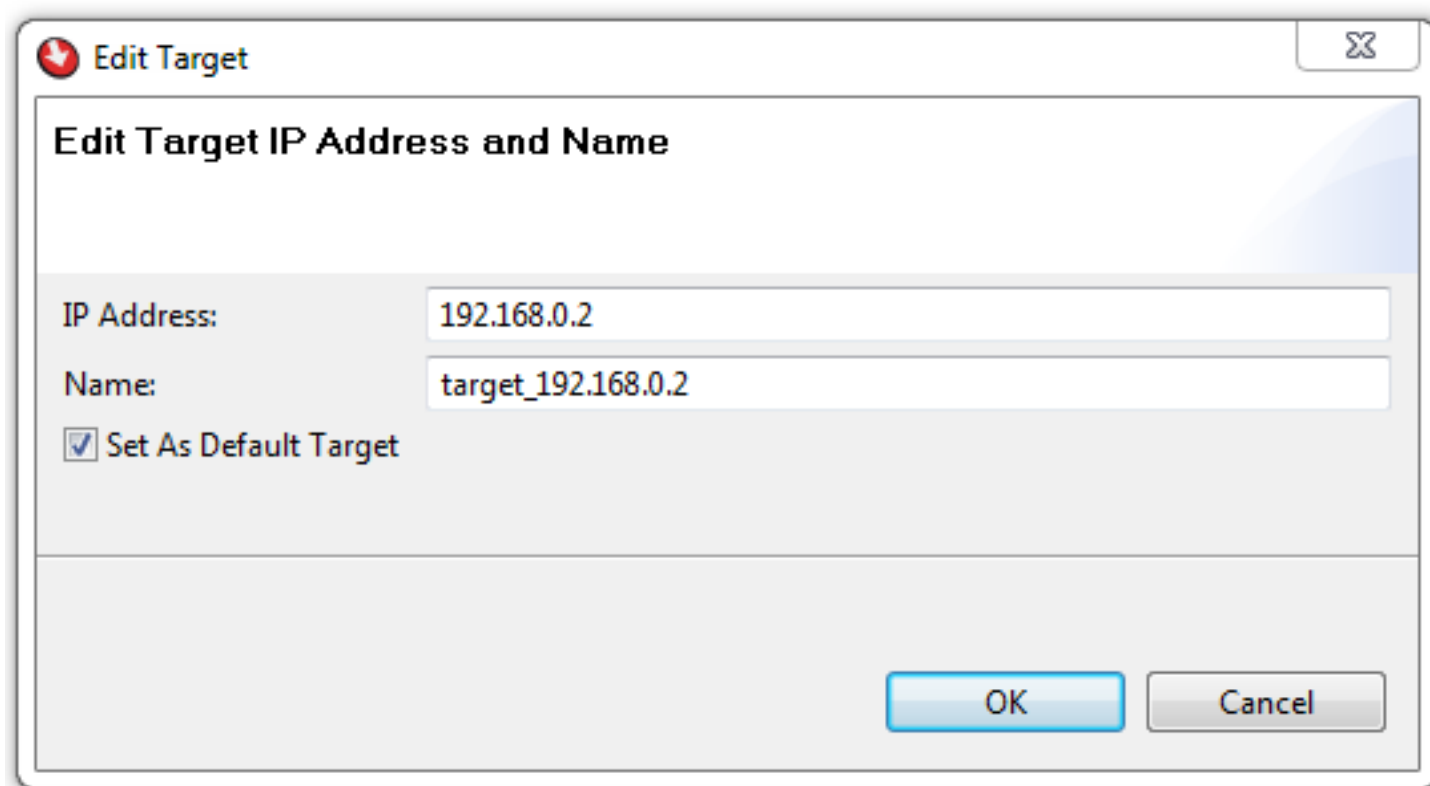
The MESSINA **Target Manager** view is available in the **Executor** perspective.

The **Target Manager** view lists all currently configured targets. This does not mean that all targets listed are immediately available (able to be connected), rather it lists all targets that could be connected if all required external settings are made.

A Windows target is always available (installed automatically).

Add a Target (Not available in SiL-only version)

A new target can be added to the **Target Manager** by clicking the icon . The following dialog is called:

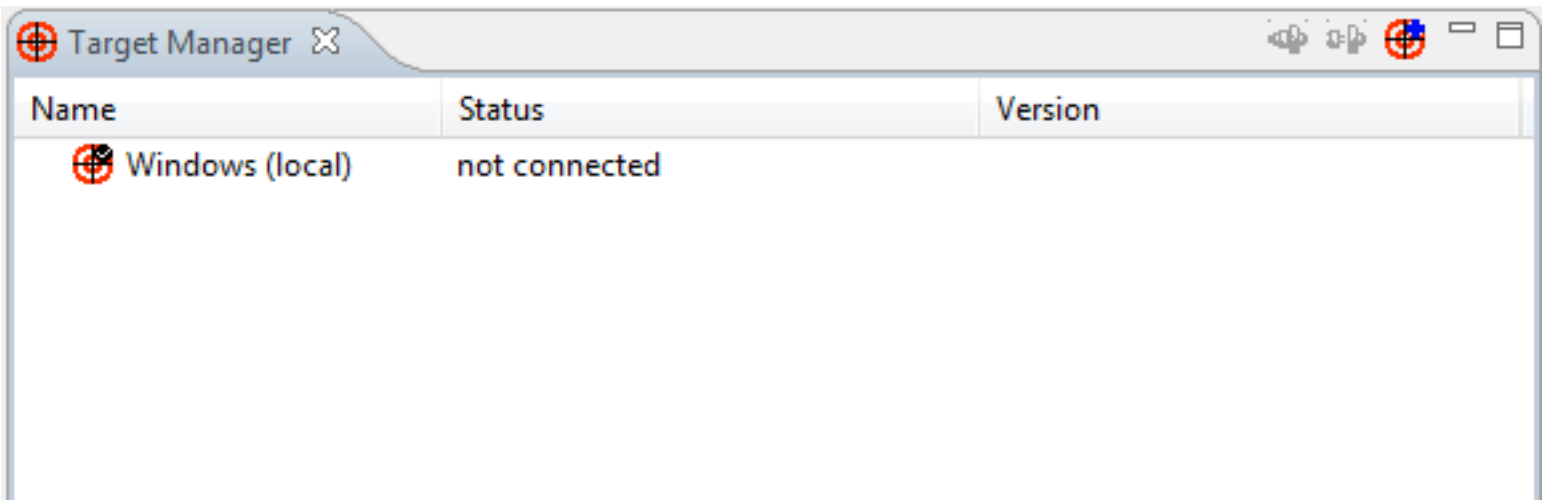


The image shows a Windows-style dialog box titled "Edit Target". Inside the dialog, the main heading is "Edit Target IP Address and Name". There are two text input fields: "IP Address:" with the value "192.168.0.2" and "Name:" with the value "target_192.168.0.2". Below these fields is a checkbox labeled "Set As Default Target" which is checked. At the bottom right of the dialog are two buttons: "OK" and "Cancel".

The **IP Address** text box is used to enter the IP address of the target. The format must be correct. The **Name** text box is used to enter the target name. A sample name is always generated automatically, but this can be changed to any target name that does not already exist in the project.


Note: Blank spaces are not allowed in the name.

The **Target Manager** view now lists the newly added target system with a **Status** of **not connected**:

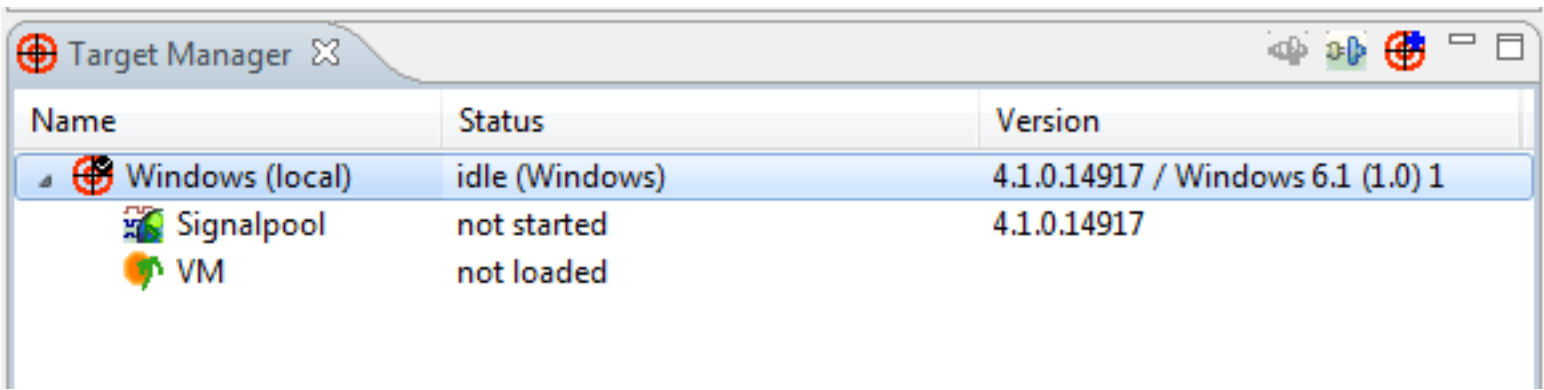


The above screen shot shows 2 targets: the Windows target (which is installed automatically) and an optional and the added target.

Connect a Target

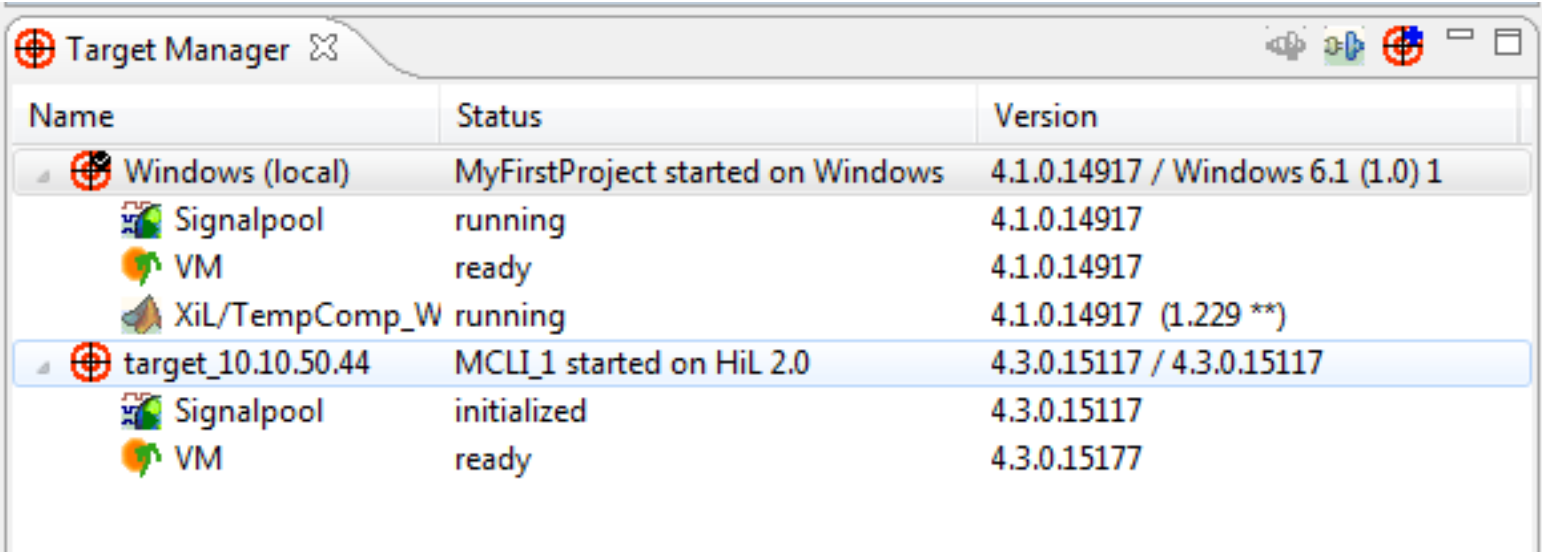
The target can be connected either by pressing the icon  or by calling **Context Menu** → **connect**.

The target can be disconnected by calling **Context Menu** → **Disconnect** or by pressing the  icon.



HiL Targets: the Hardware in the Loop target is used to connect to actual hardware which must be connected to the test system. This is done using an Ethernet cable and connection.

Several targets can be configured, these will all be listed in the **Target Manager** view. An example of a **Target Manager** view with several targets, one is running (the test configuration - e.g. HiL - has been started), is shown below.




Name	Status	Version
Windows (local)	MyFirstProject started on Windows	4.1.0.14917 / Windows 6.1 (1.0) 1
Signalpool	running	4.1.0.14917
VM	ready	4.1.0.14917
XiL/TempComp_W	running	4.1.0.14917 (1.229 **)
target_10.10.50.44	MCLI_1 started on HiL 2.0	4.3.0.15117 / 4.3.0.15117
Signalpool	initialized	4.3.0.15117
VM	ready	4.3.0.15177

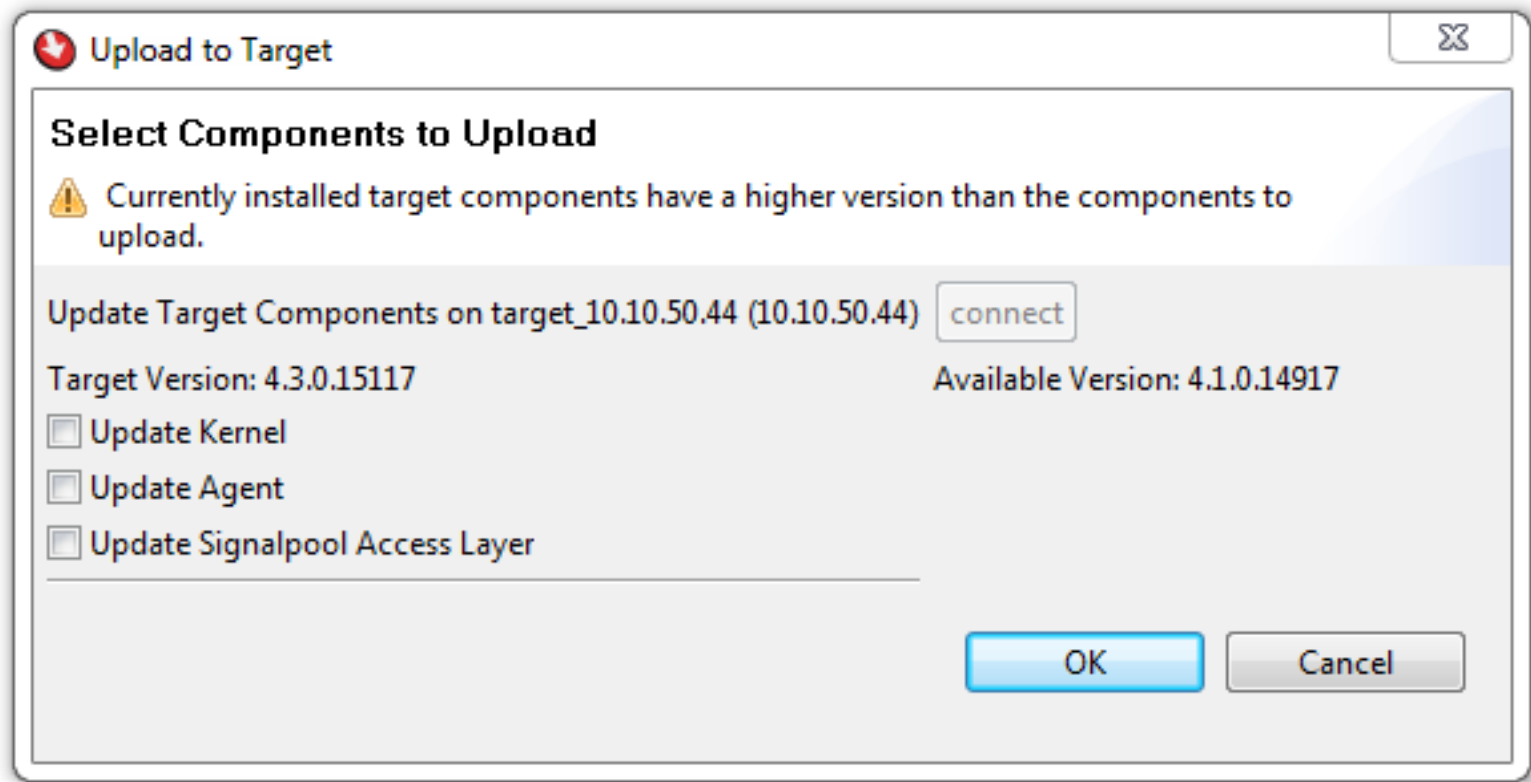
The **Status** column indicates the current state of the item listed in the **Name** column. The status of the **Signalpool** changes to **running**. The **Signalpool** is an active process within **MESSINA**.

The target settings can be edited any time by double clicking on the desired target and making the required changes directly in the dialog box that is called.

The context menu in the Target Manager offers the following options:


Context Menu → **Set as Default Target**: Sets the selected target as the default. The default target is indicated by the  icon in the **Target Manager** view.

Context Menu → **Upload Target Components**: This option is used to update the components of the selected target device. The following dialog is called:



Upload to Target

Select Components to Upload

 Currently installed target components have a higher version than the components to upload.

Update Target Components on target_10.10.50.44 (10.10.50.44)

Target Version: 4.3.0.15117 Available Version: 4.1.0.14917

☐ Update Kernel

☐ Update Agent

☐ Update Signalpool Access Layer

Context Menu → **Stop Target**: This option stops the currently running configuration

Context Menu → **Reboot**: This option reboots the operating system on the target device.

Context Menu → **Edit**: This option call the edit dialog (described above) and allows the settings of the target device to be modified.

Result Manager

The MESSINA **Result Manager** view is available in the **Executor** perspective.

The **Result Manager** is a view used for displaying the results of a **Test Case** or **Campaign** during and after execution. A tree structure/hierarchy is used to enable easy navigation through the different processes and the different levels of each process. Each execution process will create a new entry into the tree structure.

The highest level of the structure identifies the **Test Case** or **Campaign**. The name used is generated automatically using the project name, testenvironment name, and a time stamp.

An example of the **Result Manager** view is shown below. This information was generated using the **TempComp** example **Test Case** and **Campaign**.

Name	Result	Date	Duration
*MyFirstProject_XiL_170307161516	PASSED (PASSED:1 FAILED:0 ERROR:0)		0:00:41.327
Project Parameters			
speed = 130			
timeout = 30000			
air_temp = 5			
MyTestCase.java	PASSED	07.03.2017 16:15:16	0:00:41.327
*MyFirstProject_XiL_170307163029	FAILED (PASSED:3 FAILED:3 ERROR:0)		0:04:54.518
Iteration 1	FAILED (PASSED:1 FAILED:1 ERROR:0)		0:01:38.304
Iteration 2	FAILED (PASSED:1 FAILED:1 ERROR:0)		0:01:38.107
Iteration 3	FAILED (PASSED:1 FAILED:1 ERROR:0)		0:01:38.107
Campaign Parameters			
MyTestCampaign	FAILED (PASSED:1 FAILED:1 ERROR:0)		0:01:38.107
Campaign Parameters			
MyTestCase.java (<default>)	PASSED	07.03.2017 16:33:50	0:00:41.203
Campaign Parameters			
air_temp = 5			
timeout = 30000			
speed = 130			
MyTestCase.java (<default>)	Assertion failed: Temperature not reached (MyTestCase.java:34)	07.03.2017 16:34:32	0:00:56.904
Campaign Parameters			
air_temp = 1			
timeout = 30000			
speed = 130			
*MyFirstProject_XiL_170307163637	FAILED (PASSED:1 FAILED:1 ERROR:0)		0:01:38.323
Project Parameters			
MyTestCampaign	FAILED (PASSED:1 FAILED:1 ERROR:0)		0:01:38.323
Campaign Parameters			
MyTestCase.java (<default>)	PASSED	07.03.2017 16:36:37	0:00:41.422
Campaign Parameters			
air_temp = 5			
timeout = 30000			
speed = 130			
MyTestCase.java (<default>)	Assertion failed: Temperature not reached (MyTestCase.java:34)	07.03.2017 16:37:18	0:00:56.901
Campaign Parameters			
air_temp = 1			
timeout = 30000			
speed = 130			

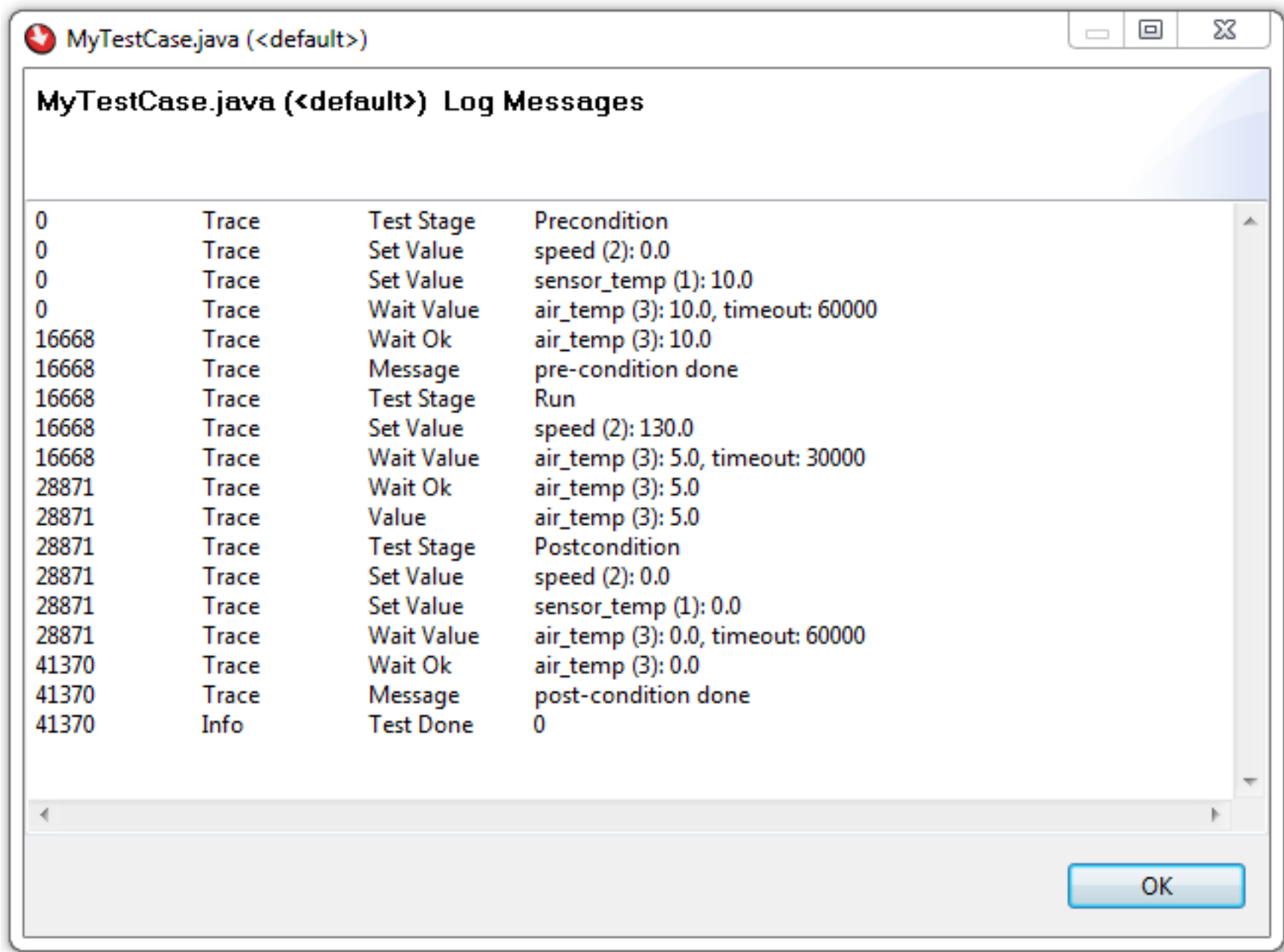
A time stamp is included with each entry (Date column), as is a duration. The result of the test is determined within the source code of the **Test Case**. The color coding (green and red) helps to visually identify the test result.

The entry in the above picture was created by executing a **Test Case** directly. By expanding the tree view, the name of the **Test Case** (MyTestCase.java) is displayed. If the next level is expanded, the parameters used in the **Test Case** are displayed. The result was a PASS.

The second entry in the list above was created by executing a **Campaign** 3 times. Each execution is placed in an **Iteration** folder so that each separate execution can be examined separately. The **Test Case** name and parameters can be viewed for each **Iteration** of the **Campaign** during execution.

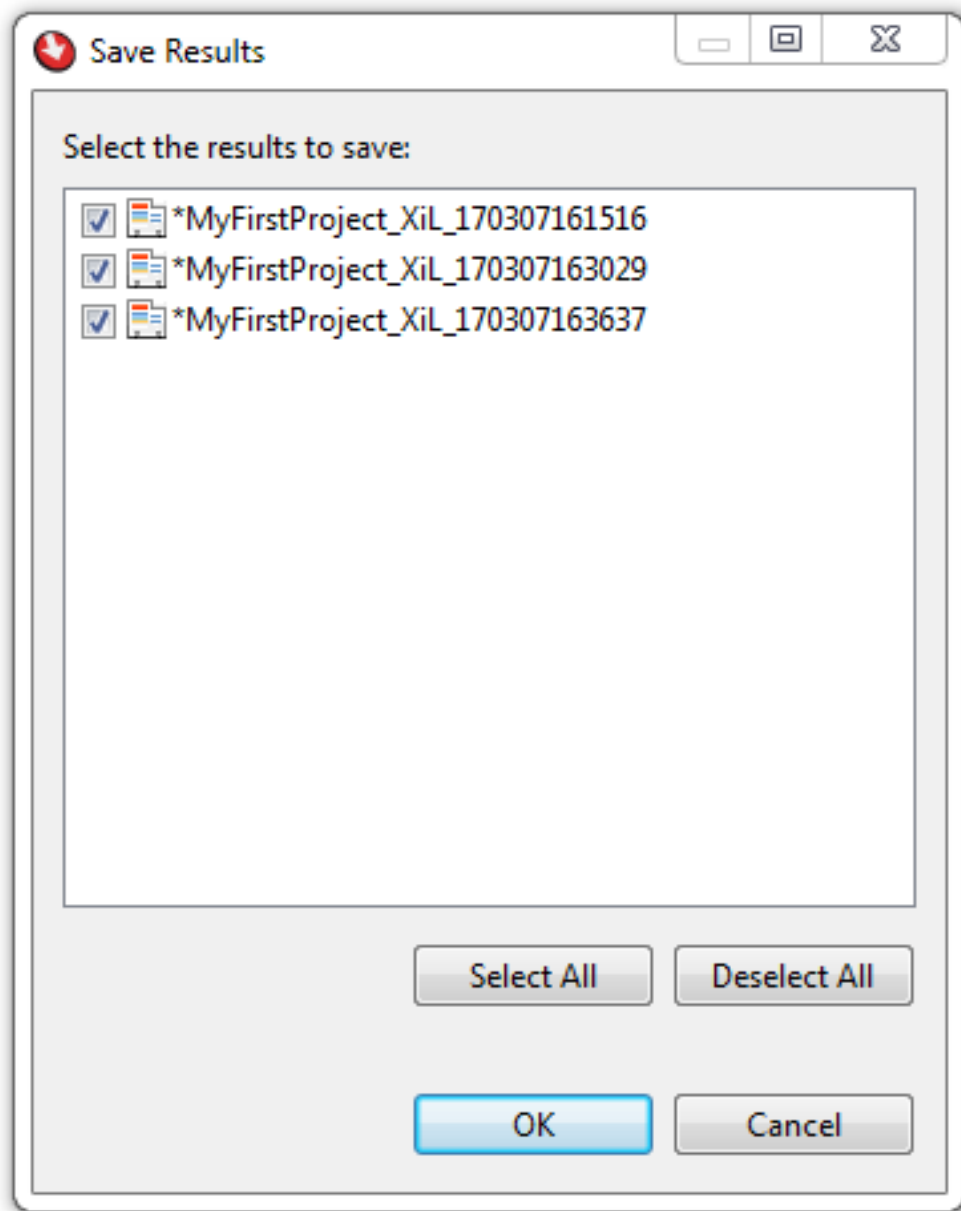
A **Test Case** can be executed again by marking the desired item in the **Result Manager** view and selecting **Context Menu** → **Rerun**. The **Test Case** will be executed immediately and a new entry will be made to the **Result Manager** view.

If logging information is available (a logging operation was active during execution), the logging information can be called using **Context Menu** → **ShowLog...**.The following picture shows how the logging information might look. This picture was generated with the **Log Level** set to **All** before executing the **TempComp Campaign**.



There is a direct connection between the **Result Manager** and the **Test Results** folder shown in the

[Project Explorer](#) view. The **Test Results** folder gives a list of all results that were displayed in the **Result Manager** and saved. The **Result Manager** view must have the focus before test results can be saved. This can be done by clicking somewhere in the **Result Manager** view. The **Save icon** in the menu bar (or **Main Menu** → **File** → **Save**) can be used to save the test results. The following dialog box is called:



The check-boxes can be used to selectively save desired test results. Press **OK** to save the results.

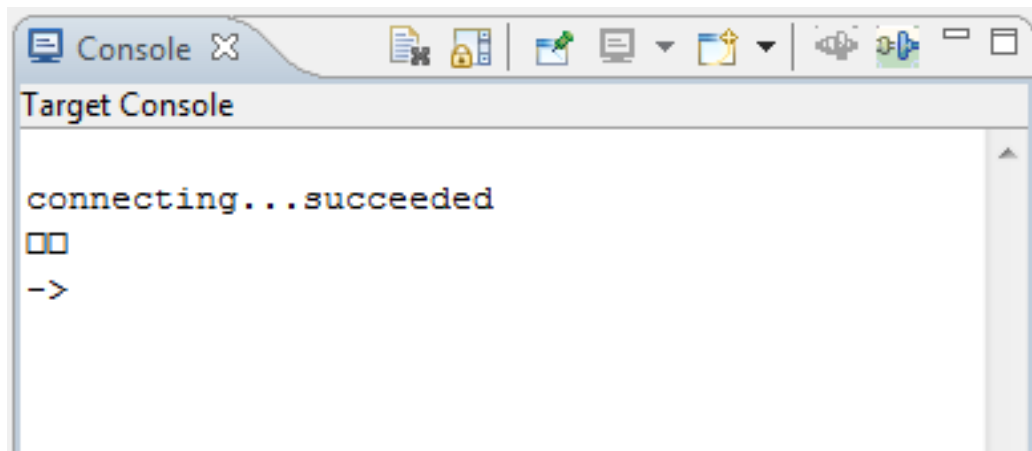
Warning: Data is permanently lost if MESSINA is closed and the test results are not saved.

A new item is added to the **Test Results** folder in the [Project Explorer](#) for each set of test results. This is the file where the test results are saved. The file name is generated automatically and includes a time stamp.






A [Test Report](#) can also be generated from the **Test Results**. A detailed description of test reports and how they are generated can be found in the **Test Report** section of this documentation.

Console

MESSINA offers the ability to connect to a target console. This can be very convenient for checking and monitoring activity on the target device. In order to connect a console, a target device must already be connected in the [Target Manager](#). An example of a console is shown below:



Icons:

-  Clear Console: pressing this icon will clear the console display.
-  Scroll Lock: pressing this icon will lock the scrolling operation of the console
-  Pin Console: pressing this icon will pin the console (attach it to a fixed position in the main window)
-  Display Selected Console: pressing the arrow beside the icon allows a selection of available consoles for display
-  Open Console: pressing the arrow beside the icon allows a selection of available consoles to open.

To properly connect and use a console, the following steps are required:

1. A target device must be connected (using the [Target Manager](#))
2. Press the arrow next to the Open Console icon, and select the **Target Console** option
3. Press the **Connect** icon

Amongst others the console is used to display the target information. In the picture below, the console is connected to the VxWin target. By entering the " i " and pressing return, the target information is

displayed. The tasks running on the target system can be seen.

```

connecting...succeeded
[]
-> i
i

```

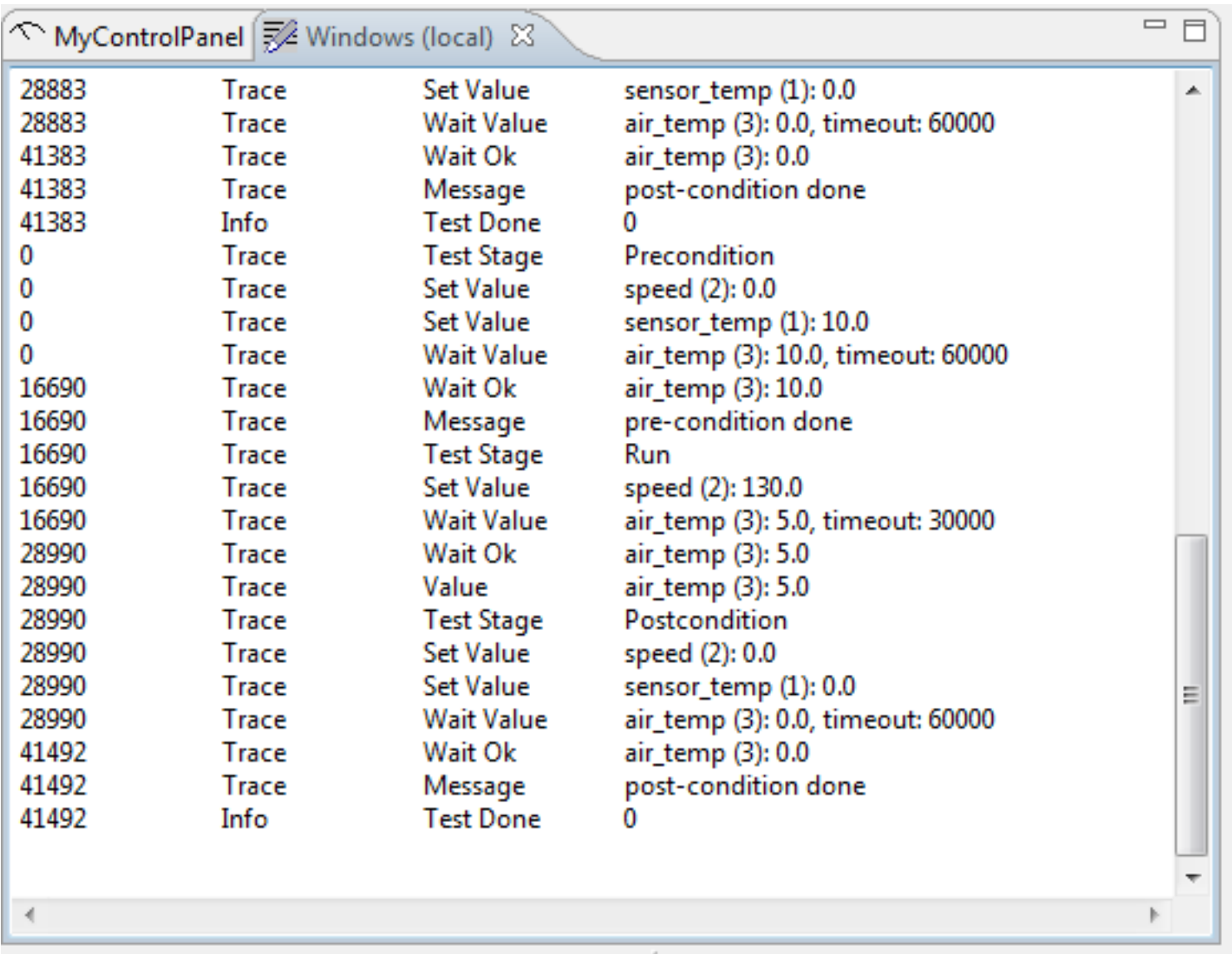
NAME	ENTRY	TID	PRI	STATUS	PC	SP	ERRNO	CPU #
tIsr0	47fa90	e9f3010	0	PEND	60084c	db34b560	0	-
tJobTask	530140	e9f5650	0	PEND	60084c	db34f460	0	-
tLogTask	logTask	e9f5ae0	0	PEND	5ff69f	db3527b0	0	-
tShellRem2>	shellTask	f12d9a0	1	READY	60a5f0	11322680	0	0
tWdbTask	5f9f10	eb39010	3	PEND	60084c	db584880	0	-
tErfTask	4c5860	e9f9d70	10	PEND	601569	db355830	0	-
tTickCount>	_ZN9Schedule	f0039a0	10	READY	43ce80	f92defc	0	3
SAL_0_1_11	_ZN6Thread3r	ecad5d0	11	DELAY	60839e	db640a90	3d0002	-
ipcom_tick>	61f9a0	eb45c90	20	PEND	60060a	db4bc5c0	0	-
tVxdbgTask	4e3520	eb35d40	25	PEND	60084c	db5808a0	0	-
tAhciMon	48d3c0	e9fd010	49	PEND	5ff69f	e9fd560	0	-
tAhciSv000	48ccd0	ea03a50	50	PEND	601569	db35f190	0	-
tNet0	ipcomNetTask	ea13b10	50	READY	62065d	db36f738	3d0001	-
ipcom_sysl>	4ec520	ea1d640	50	PEND	601569	db4c7340	380003	-
tNetConf	5260a0	ea2ba38	50	PEND	60060a	db4edca0	0	-
ipcom_telnet>	ipcom_telnet	ea2fc08	50	PEND	60084c	db4f8290	0	-
tAcpi	AcpiThread	ea49430	50	PEND	5ff69f	db578830	0	-
tVxbC200	4d1e40	eb47d30	50	PEND	60084c	db5a8900	0	-
ipcom_telnet>	ipcom_telnet	f023010	50	PEND	60084c	1130b7f0	0	-
tStdioProx>	4f74b0	f02d3e8	50	READY	602e7d	1134e520	0	1
tLoginf023>	4f79e0	efed2f8	50	PEND	601569	1130f200	0	-
tPortmapd	portmapd	ea3db00	54	PEND	60084c	db4fc950	16	-

Log View and Log Levels

The MESSINA **Log View** is available in the **Executor** perspective.

It is possible to do a visual logging of the test process using the built-in test settings. See the **Preferences** for more detailed information.

The **Log View** displays information based on the **Log Level** setting. There is one **Log View** for each target. The **Log View** for the currently running target is automatically called (gets focus) when the execution of a **Test Case** or **Campaign** begins and the **Log Level** is anything other than off (if any form of logging is active). The logging information is written to the **Log View** continually during a running test process. An example of a **Log View** for a Windows target is shown below. The **TempComp** example **Test Case** was executed once with the **Log Level** set to **Trace** to create the following output.



The information displayed in the **Log View** is not saved to a file. It is intended as information to be displayed during a running test process. The **Log View** is a text display so the information can be marked, copied, and saved to another file if required. The information displayed is currently limited to approx. 10k of data. Once this limit is reached, the oldest information is discarded when new information is added.

Log Levels

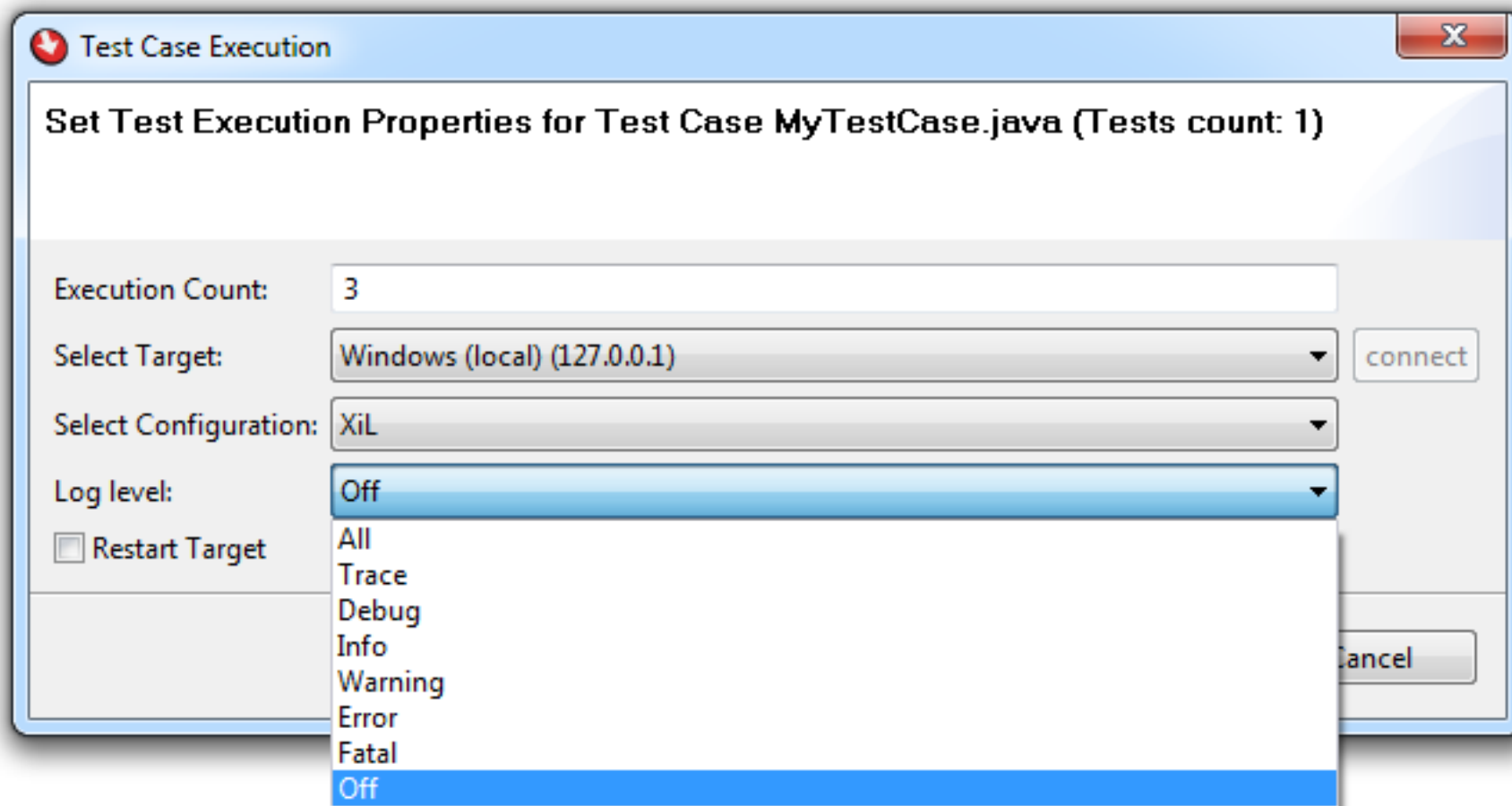
The MESSINA **Log Levels** are based on the Java Log4j standards for logging processes. The **Log Level** is set using a pull down list which is described in detail in the **Preferences** section of this documentation. The **Log Level** defines which information is displayed in the **Log View** for the currently running target.

The selections are cumulative. This means all options including and below the selected option are also active. For example, if the **Debug** option is selected, this would mean that the **Info**, **Warning**, **Error** and **Fatal** options are also active.

The following table summarizes the **Log Level** settings and their associated constants:

Log Level	Test Case Constant Name	Description
All	ITargetLogger.ALL	all logging functions are active
Trace	ITargetLogger.TRACE	logs the stage (e.g. pre-condition, run, post-condition) and signal changes
Debug	ITargetLogger.DEBUG	User defined logging of messages and values in the Test Case (e.g. function call: log(ITargetLogger.DEBUG, "hello")
Info	ITargetLogger.INFO	logs the test status (e.g. done) and the return value
Warning	ITargetLogger.WARNING	makes a log entry when a "wait" operation fails (e.g. waitTrigger, waitValue)
Error	ITargetLogger.ERROR	makes a log entry when an error occurs
Fatal	ITargetLogger.FATAL	makes a log entry when a fatal error (exceptions) occurs
Off (default)	-	nothing is logged

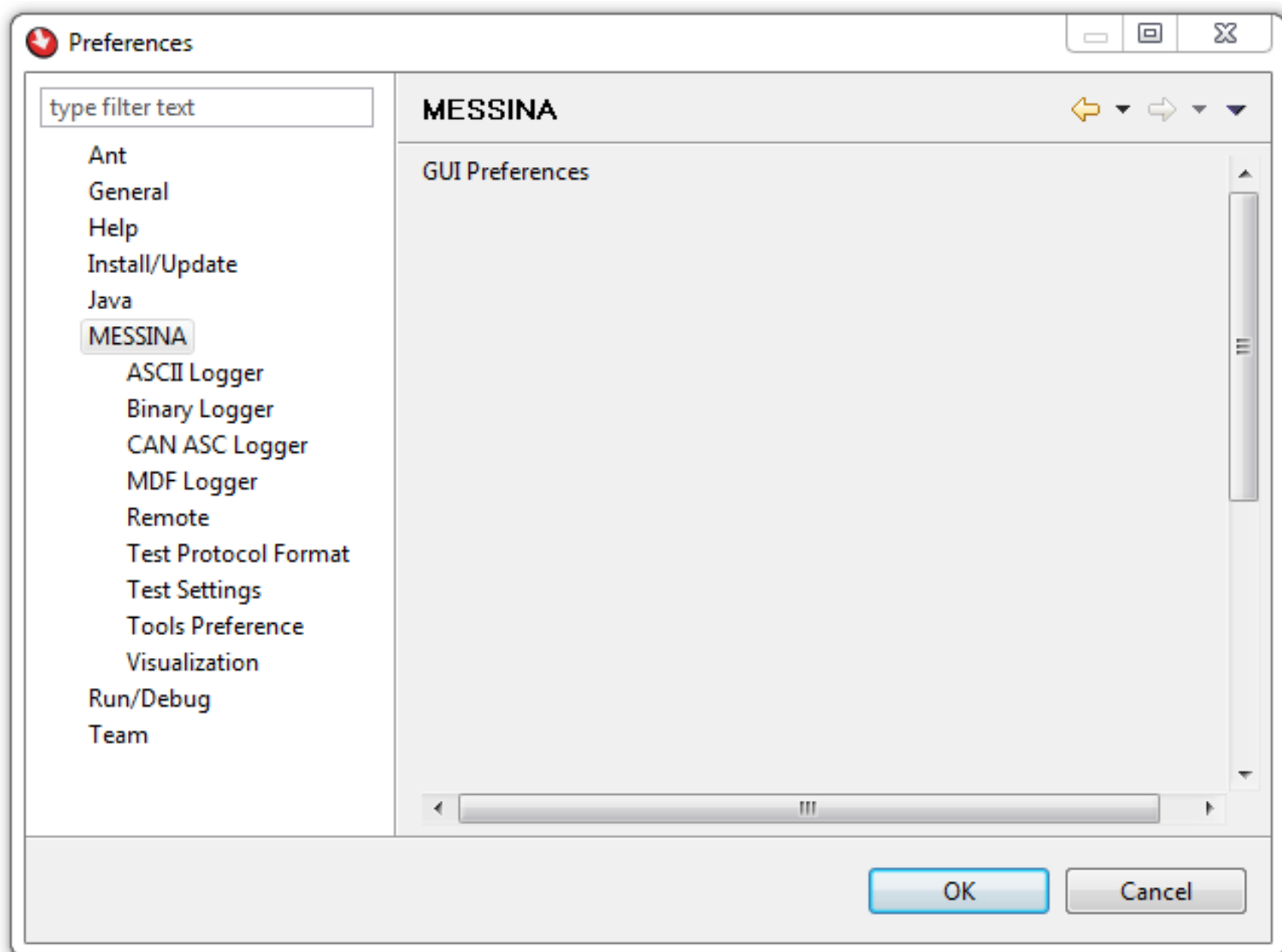
Note: The setting made in the **Preferences** is used as the default selection when a **Test Case** or **Campaign** is executed. It is possible to override this setting for the execution of a **Test Case** or **Campaign** which was started using the **Context Menu** → **Execute...** option. The following picture shows the **Execute** dialog:



The **Log Level** pull-down list can be used to override the default setting for this execution process only.

Preferences

The following screen shot shows the **Preferences** dialog box which can be called from the **Main Menu** → **Window** → **Preferences**.



The following sections describe each setting in detail.

ASCII Logger

- **Description:** The ASCII Logger is used to log signals in a normal ASCII text format. This format can be viewed using any standard Windows text editor application (e.g. Notepad.exe)
- **Setting:** Press the **Browse** button and navigate to the desired text editor application.

Binary Logger

- **Description:** The Binary Logger is used to log signals in a binary format. This format will require a viewer that can display hexadecimal character representations (hex viewer).

- **Setting:** Press the **Browse** button and navigate to the desired editor application.

CAN ASC Logger

- **Description:** The CAN ASC Logger visualization offers the ability to log and store particular CAN signals available in the Signalpool.
- **Setting:** Activate or deactivate the logging function.

MDF Logger

- **Description:** The MDF logger is used to log signals in MDF format. This format is very common in the automobile industry and can be used for a very flexible formatting of data (e.g. tables, graphs etc). To view files in this format, a special viewer tool is required (e.g. Canape32.exe). This tool is not part of MESSINA and must be installed separately.
- **Setting:** Press the **Browse** button and navigate to the desired editor application (e.g. Canape32.exe).

Remote

- **Description:** This setting is used to set the remote connection to MESSINA.
- **Setting:** Enter the desired values directly.

Test Protocol Format

- **Description:** This setting defines the style sheet used for the standard test report.
- **Setting:** Press the Browse button and navigate to the style sheet. An example style sheet is part of the MESSINA installation. It is found in the following directory:
...\MESSINA\workspace\Library\ReportStyleSheet\MessinaTestResults.xsl

Test Settings

- **Description:** The settings listed here are for the data displayed in the [Log View](#) in MESSINA.
- **Setting:** Use the pull-down list to select the desired logging level. The selections are cumulative. This means all options including and below the selected option are also active. For example, if the **Debug** option is selected, this would mean that the **Info**, **Warning**, **Error** and **Fatal** options are also active.
The possible options are:
 - All: All logging functions are active

- Trace: Logs the stage (e.g. pre-condition, run, post-condition)
- Debug: Logs the debug messages in the test case source code
- Info: Logs the test status (e.g. running, done) and the return value
- Warning: Makes a log entry when a "wait" operation fails (e.g. waitTrigger, waitValue)
- Error: Makes a log entry when an error occurs
- Fatal: Makes a log entry when a fatal error (exception) occurs
- Off (default): Nothing is logged

Note: The setting made here is used as the default selection when a **Test Case** or **Campaign** is executed. It is possible to change the state of this setting before a **Test Case** or **Campaign** is run when the process is started using the **Context Menu** → **Execute...** option to start the execution of a **Test Case** or **Campaign**.

Visualization

- **Description:** This option sets how MESSINA reacts when opening a visualization.
- **Setting:** Activate the radio button for the desired operation.

Creating and Executing Tests

Programming Test Cases

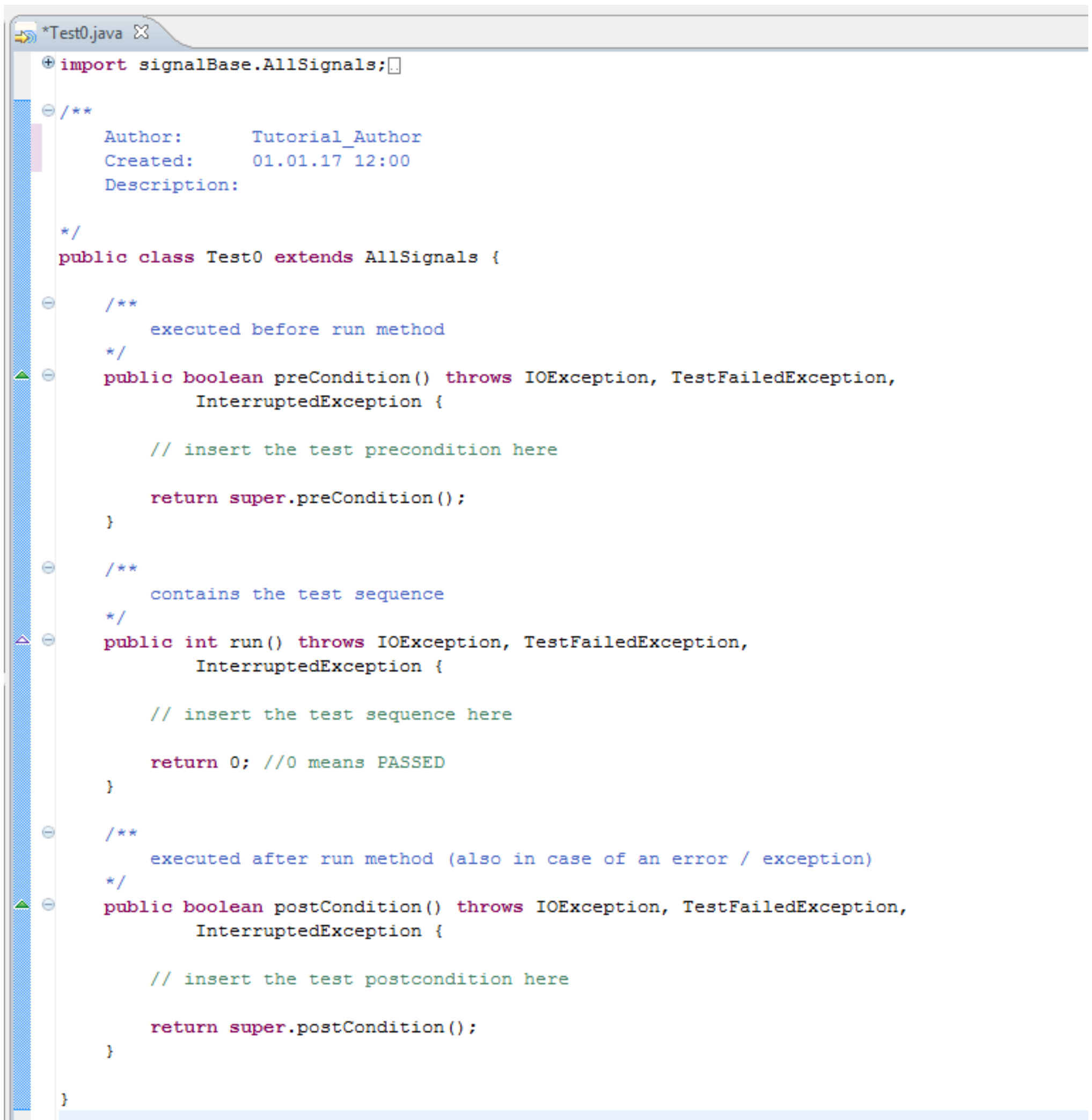
Test Cases are programmed using Java. Existing functions can be used to cover almost all test situations. All existing functions are described in the *Using MESSINA → Creating Tests → [Signal Commands](#)* section of this documentation.

Test Cases will always contain a "**run**" function. This function is created automatically and initially contains no functional source code. It is possible to add a "**preCondition**" and/or a "**postCondition**" function to the **Test Case**.

The **preCondition** function is executed before the run function. It can be used, for example, to perform an initialization or to make the required settings before a test can be executed. The **postCondition** function is executed after the **run** function. It can be used, for example, to reset items that may have been changed or to restore a default condition. The location of the **preCondition**, **run**, and **postCondition** functions within the **Test Case** class does not matter. Thus, the execution order within a **Test Case** will always be:

1. **preCondition** function
2. **run** function
3. **postCondition** function

The picture below shows a newly created **Test Case**.



```
*Test0.java
import signalBase.AllSignals;

/**
 * Author: Tutorial_Author
 * Created: 01.01.17 12:00
 * Description:
 */
public class Test0 extends AllSignals {

    /**
     * executed before run method
     */
    public boolean preCondition() throws IOException, TestFailedException,
        InterruptedException {

        // insert the test precondition here

        return super.preCondition();
    }

    /**
     * contains the test sequence
     */
    public int run() throws IOException, TestFailedException,
        InterruptedException {

        // insert the test sequence here

        return 0; //0 means PASSED
    }

    /**
     * executed after run method (also in case of an error / exception)
     */
    public boolean postCondition() throws IOException, TestFailedException,
        InterruptedException {

        // insert the test postcondition here

        return super.postCondition();
    }
}
```

The source code shows the functions generated when the **Test Case** is created.

A **Parameter** can be added to a **Test Case** by typing "**params.**" within a function (**run**, **preCondition**, or **postCondition**) and selecting the desired parameter from the list shown.

Once a Test Case has been created, it can be added to a **Campaign**. See the [Test Campaign](#) section of this documentation for more details.

Logging and Log Levels in Test Cases

The **Test Cases** in MESSINA allow the logging of required information to the [Log View](#) display. This can be very helpful for monitoring signal values and for debugging during development. The following table gives a summary of the **Log Levels**, the constants associated with them, and a description of what the **Log Level** does.

Log Level	Test Case Constant Name	Description
All	ITargetLogger.ALL	all logging functions are active
Trace	ITargetLogger.TRACE	logs the stage (e.g. pre-condition, run, post-condition) and signal changes
Debug	ITargetLogger.DEBUG	User defined logging of messages and values in the Test Case (e.g. function call: log(ITargetLogger.DEBUG, "hello"))
Info	ITargetLogger.INFO	logs the test status (e.g. done) and the return value
Warning	ITargetLogger.WARNING	makes a log entry when a "wait" operation fails (e.g. waitTrigger, waitValue)
Error	ITargetLogger.ERROR	makes a log entry when an error occurs
Fatal	ITargetLogger.FATAL	makes a log entry when a fatal error (exceptions) occurs
Off (default)	-	nothing is logged

It is possible to access the **Log Level** in the **Test Case** using either a discrete value or a pre-defined constant. Refer to the [logValue](#) function for a detailed description of how to perform these actions.

Examples of logging functions within a **Test Case** could be as follows:

```
log(ITargetLogger.DEBUG, "hello"); // logs the "hello" message when executed

air_temp.logValue(ITargetLogger.INFO); // logs the value of air_temp at the defined
level (INFO here)
```

Signal Capture

Refer to the [Signal Capture](#) section of this documentation to learn how to start and stop logging signals automatically using test cases..

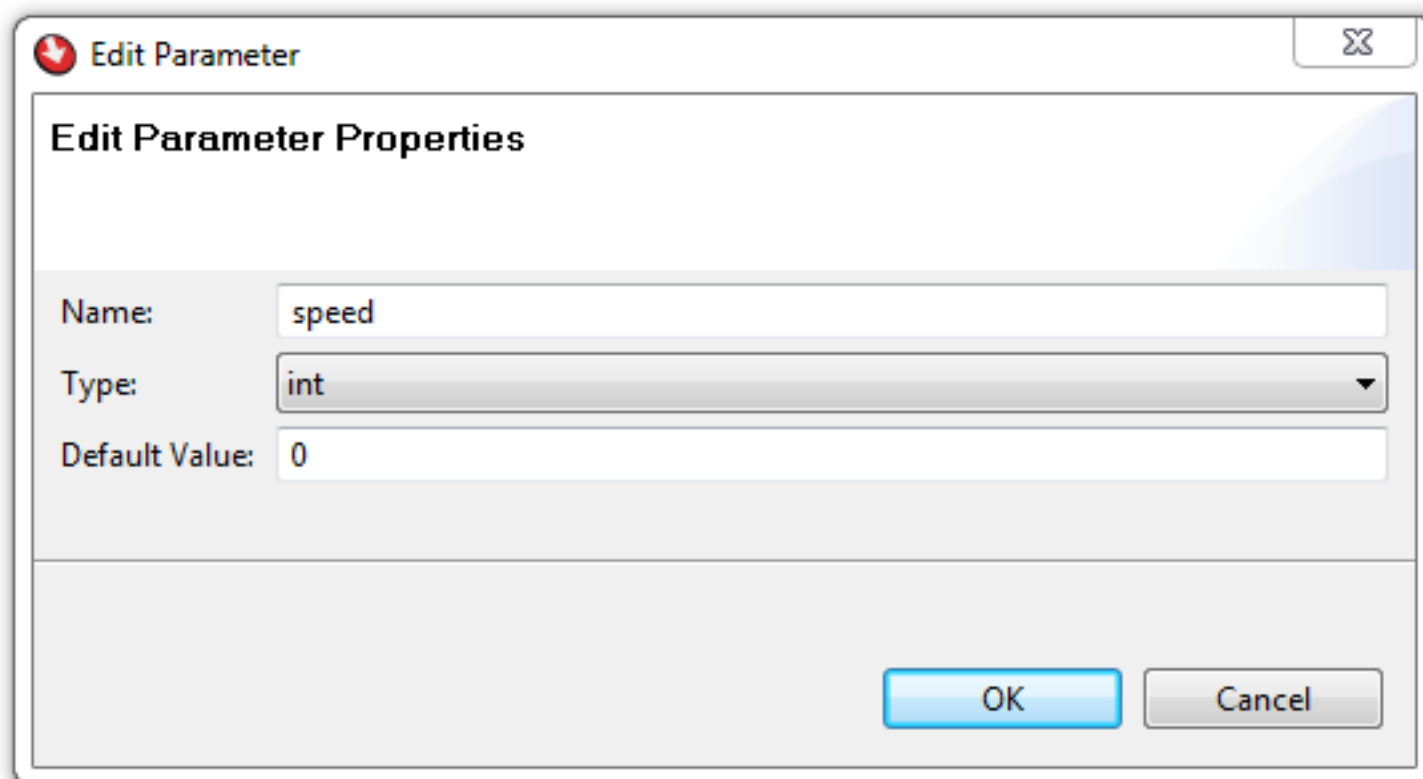
Using Parameters

The MESSINA **Parameters** can be viewed in the [Parameter Manager](#) window from the **Designer** perspective.

Parameters in the MESSINA development environment are like variables used within the MESSINA project. **Parameters** can be set and used at different points in MESSINA. For example, **Parameters** can be added in the [Project Explorer](#) view under the **Parameter** item. This can be used, for example, to set **Parameters** to default values within a MESSINA project. **Parameters** can also be used to set values for use only within a **Campaign**. It is also possible to assign **Parameters** for use only within a given **Test Case**. This provides flexibility by enabling different tests to be performed using the same **Test Case** with different parameter values.

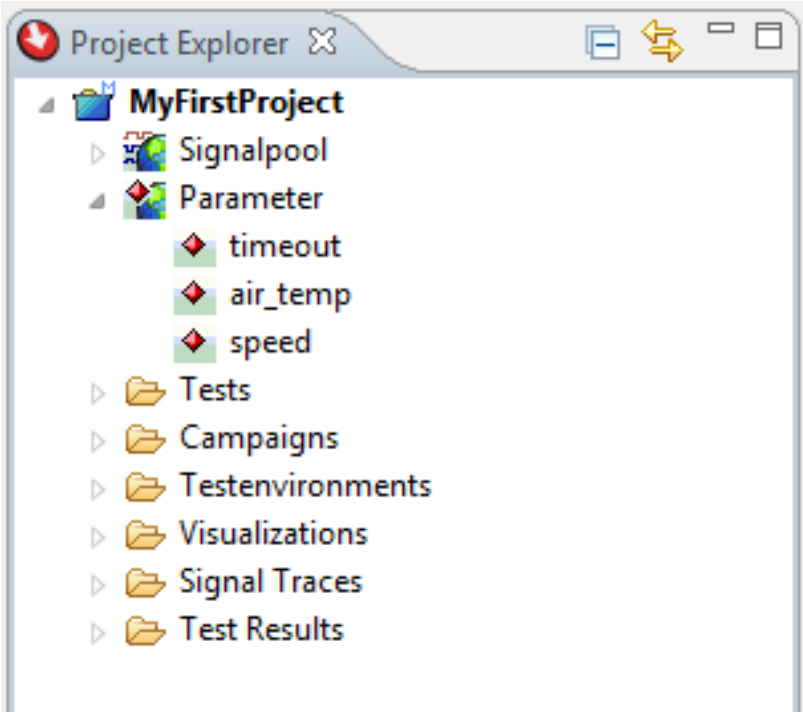
Adding Parameters

Parameters can be added by calling **Context Menu** → **New Parameter**. The following dialog is called:



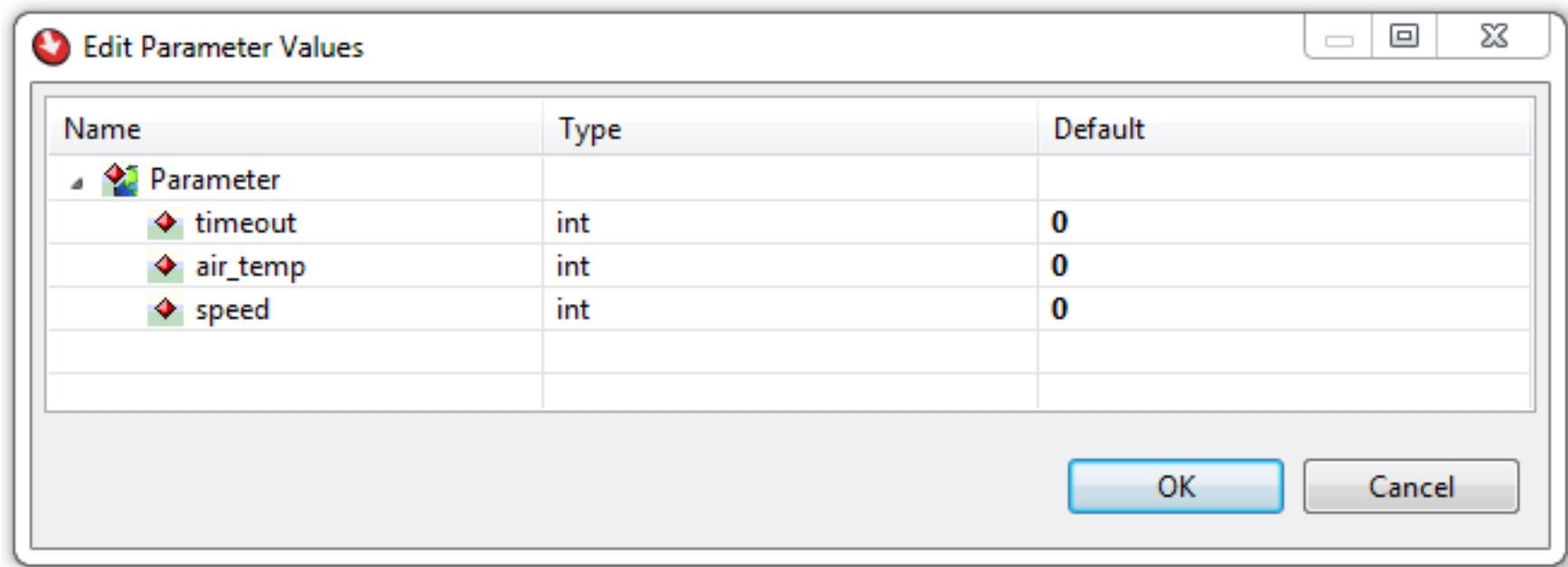
The screenshot shows a standard Windows-style dialog box titled "Edit Parameter". Inside the dialog, there is a section titled "Edit Parameter Properties". This section contains three input fields: "Name:" with the value "speed", "Type:" with a dropdown menu currently set to "int", and "Default Value:" with the value "0". At the bottom right of the dialog, there are two buttons: "OK" and "Cancel".

Enter the parameter **Name** as shown (**speed** in this case), the **Type** and **Default Value**. After pressing **OK**, the **Parameter** is added to the [Project Explorer](#) (expand the **Parameter** item) and to the [Parameter Manager](#) window. An example of the **Project Explorer** view with several parameters added is shown below.



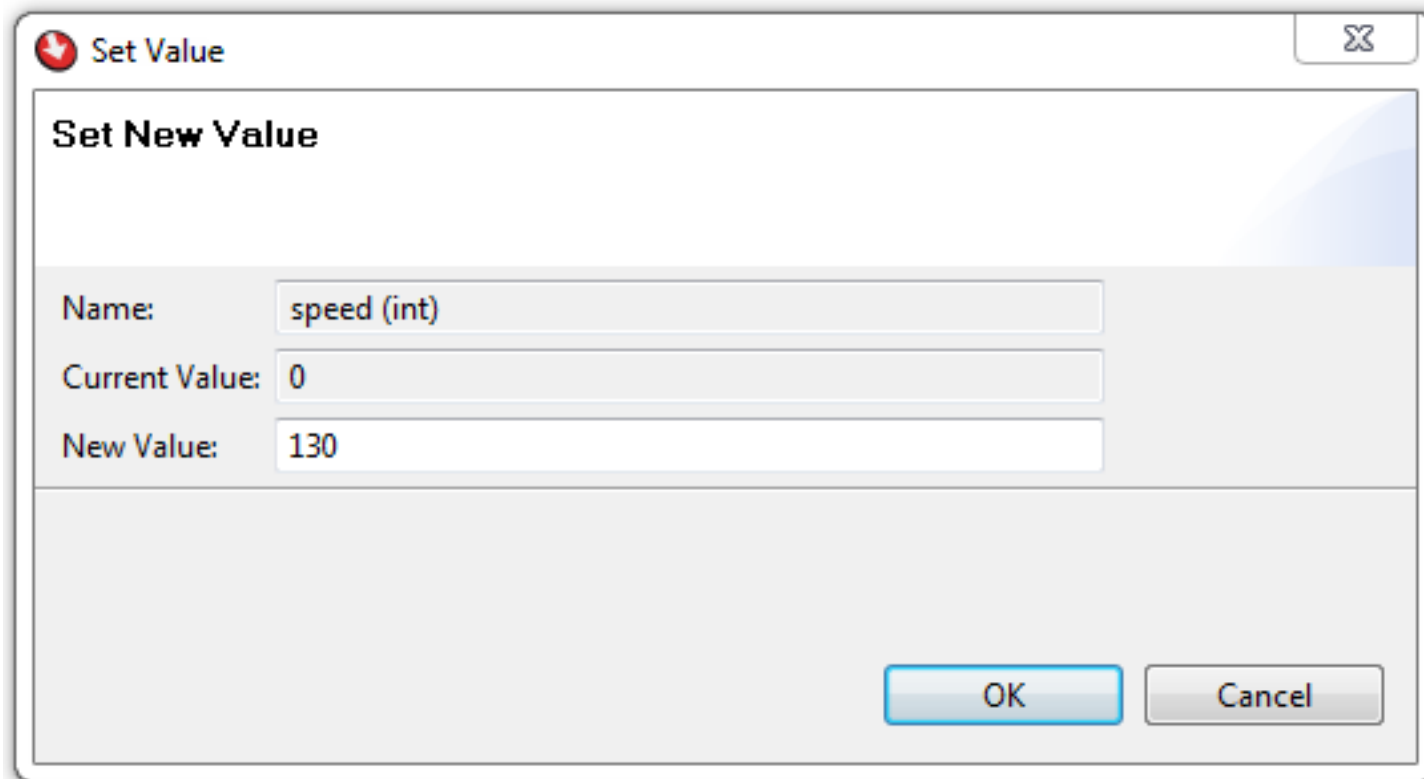
Setting Parameters for a Campaign

When a new **Campaign** is created a **Campaign Parameters** item is automatically created under that **Campaign** name. Initially it will be empty. Double click on the **Campaign Parameters** item, and the following dialog appears:



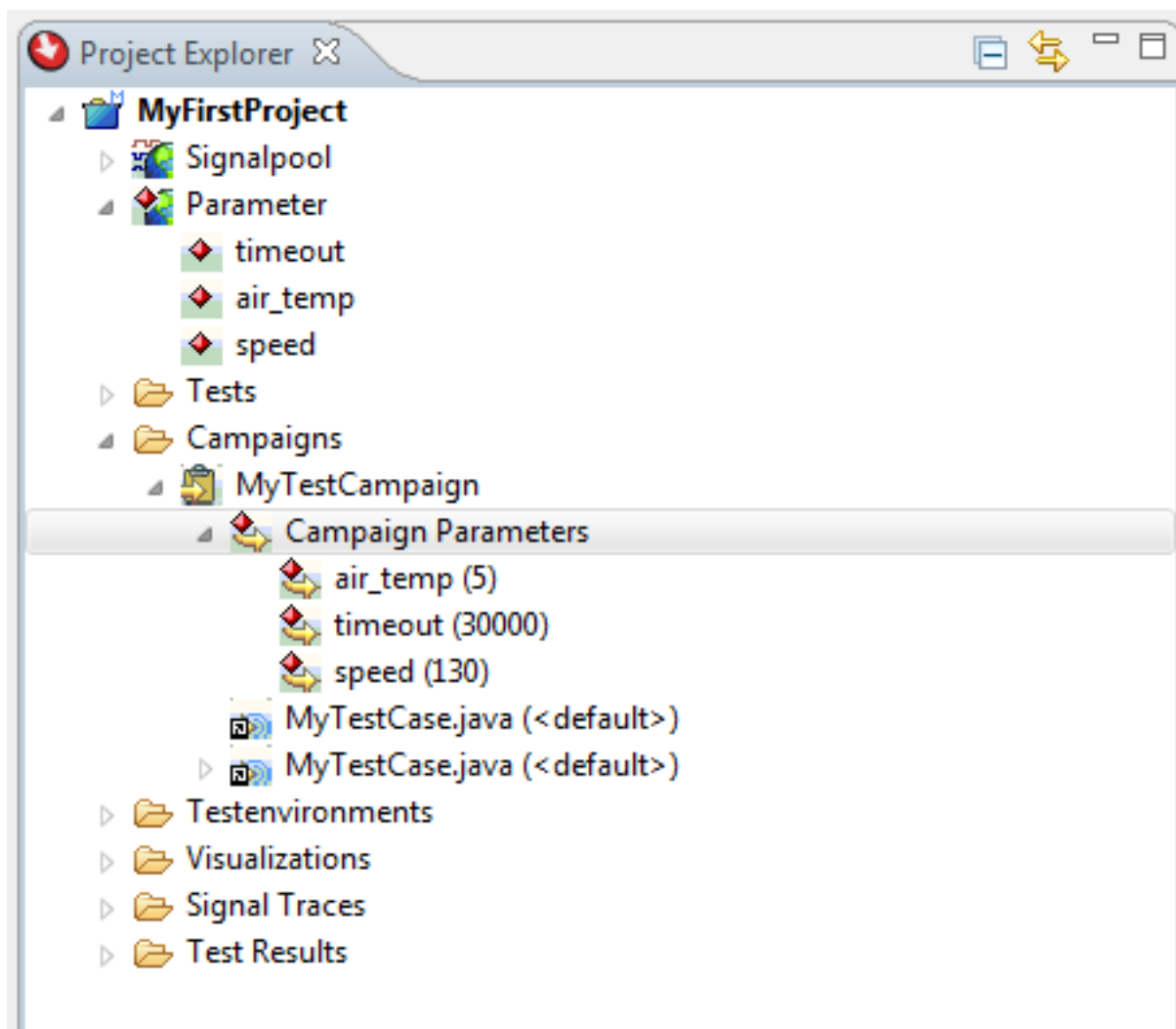
The **Type** and **Default** value of the **Parameter** are shown. The **Default** value is the value we set when the **Parameter** for the project was created.

Double-click on a parameter in the list. The following dialog appears:

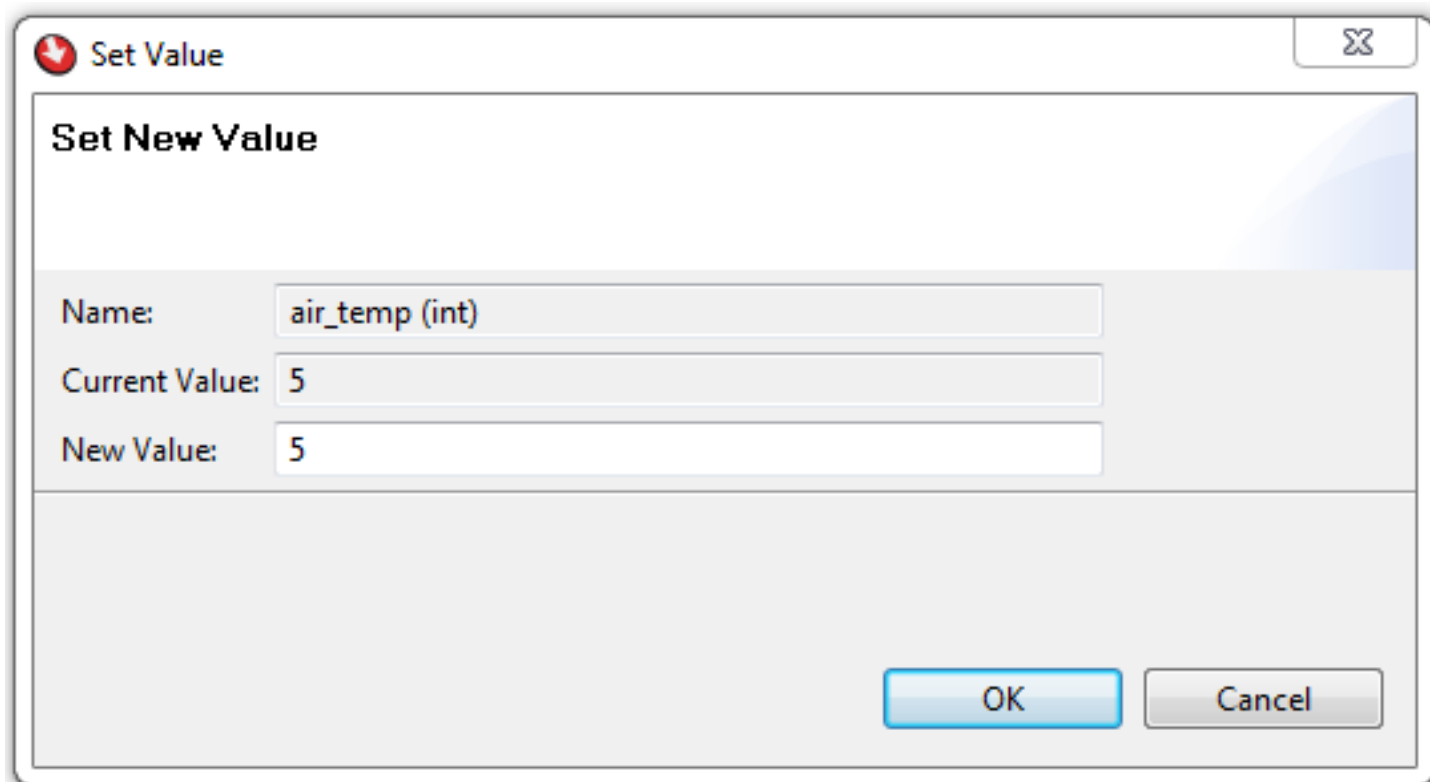


This dialog allows us to set a **New Value** for the selected **Parameter**. The value set here will only be valid within the **Campaign** where it is listed. The above example shows how to set the **New Value** of the **speed Parameter** to 130.

The [Project Explorer](#) view should have the project parameters and **Campaign** parameters. An example is shown below.



To change a **Parameter** value within the **Campaign**, double-click on the desired parameter and change the value directly in the dialog box as shown below.



To remove a parameter from the **Campaign**, use the **Context Menu** → **Un-override** option. This removes the **Parameter** value from the **Campaign** only, the **Parameter** itself remains unchanged.

Parameter Access within a Test Case

Parameters are referenced within the Test Case using "**params.**" prefix. The list of available parameters can be called by typing "**params.**". The list is expanded automatically. Use the arrow keys to move to the desired parameter. An example is shown in the picture below.

```
public int run() throws IOException, TestFailedException,
    InterruptedException {
    // set speed value to parameter
    speed.setValue(params.speed);
    // wait for air_temp to reach the parameter value
    ASSERT(air_temp.waitValue(params.timeout),
        "Temperature not reached");
    air_temp.logValue(LogLevel.TRACE);
    return 0; // 0 means PASSED
}

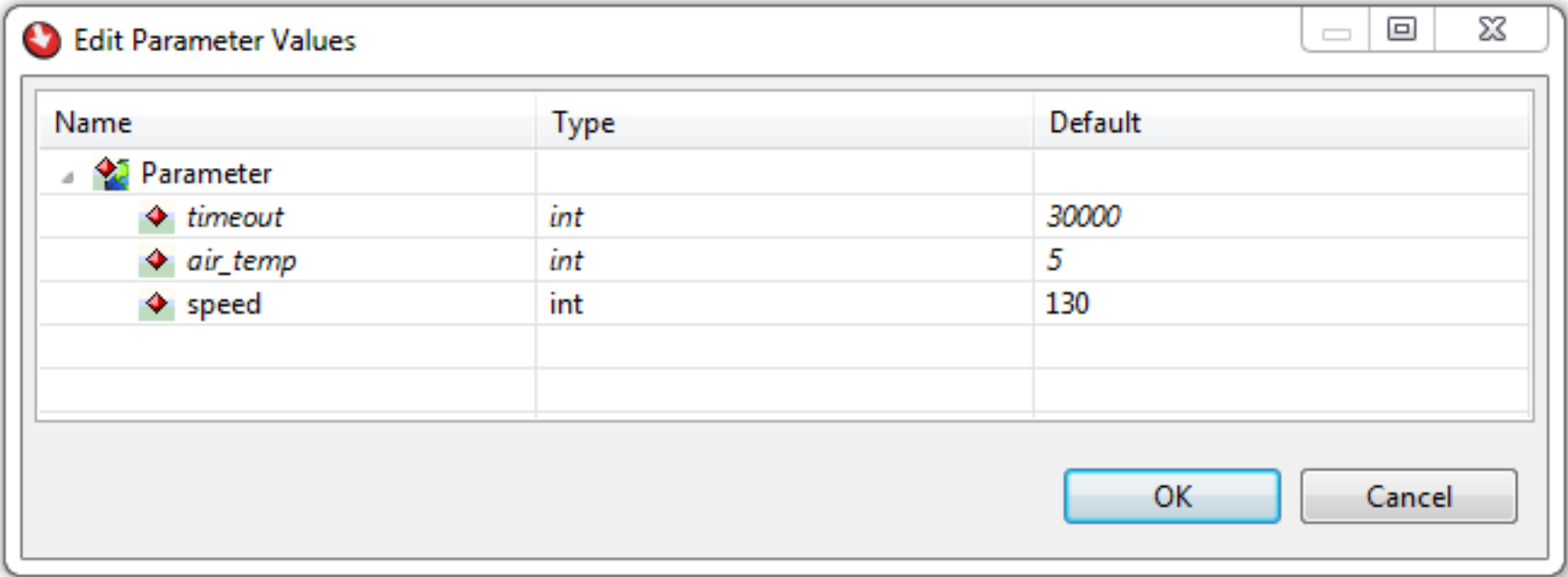
/**
 * executed after run method (also in
```

```
air_temp : int - params
speed : int - params
timeout : int - params
class : Class<signalBase.params>
this
```

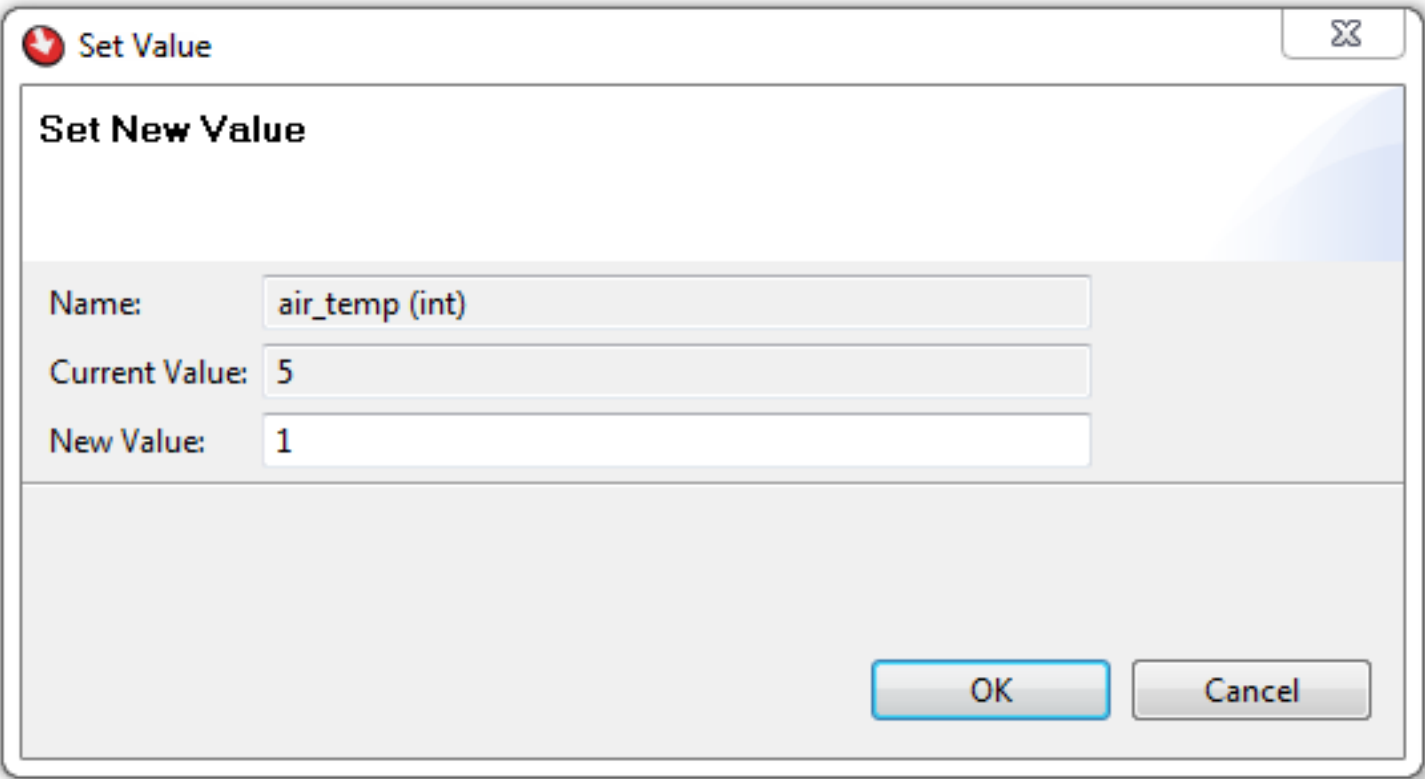
Setting Parameters for a Test Case

Parameters can be set differently not only for each **Campaign**, but also for each **Test Case**. This feature can be used to create 2 different tests based on the same **Test Case**, but using different parameters.

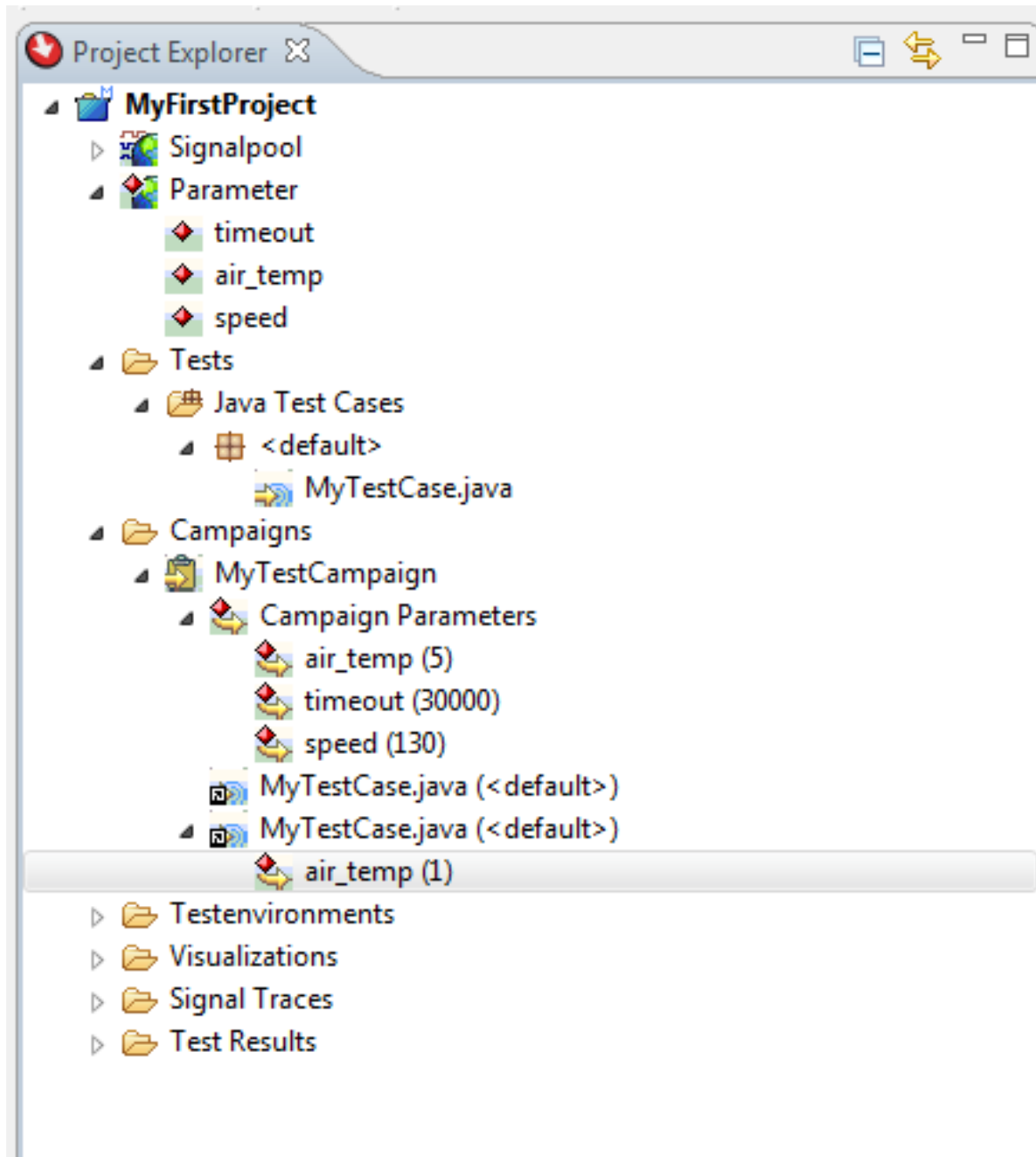
Double-clicking on the name of a **Test Case** will call the following dialog:



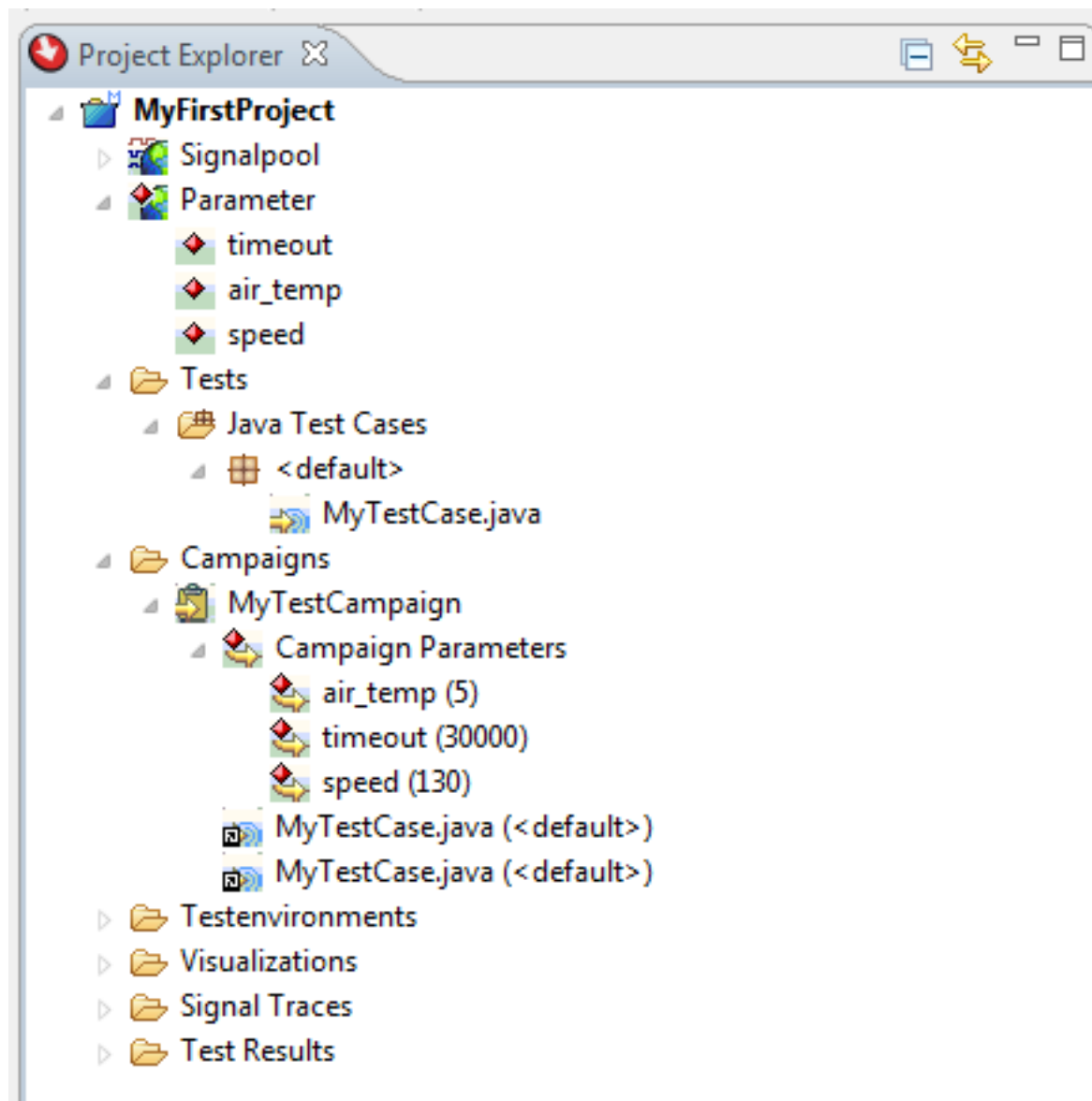
The values shown are those set for the **Campaign** and are currently relevant for all **Test Cases** within that **Campaign**. Double clicking on an entry in this list will call the following dialog:



The New Value entered here will be used for the selected **Test Case** only. An example of a [Project Explorer](#) view with parameters, **Campaign** parameter values, and **Test Case** parameter values is shown below:



If a parameter is not assigned a new value for a **Test Case**, the value for the **Campaign** is used. If a parameter is not assigned a new value for a **Campaign**, then the default parameter value is used. To remove a parameter assignment, select the desired item and use **Context Menu** → **Un-override**.



The parameter value will be removed from the **Test Case** (in the example above). This will mean that the **Campaign** value is used for the **Test Case**.

Test Campaigns

The MESSINA **Campaigns** can be viewed in the [Project Explorer](#) from any perspective under the **Campaigns** folder.

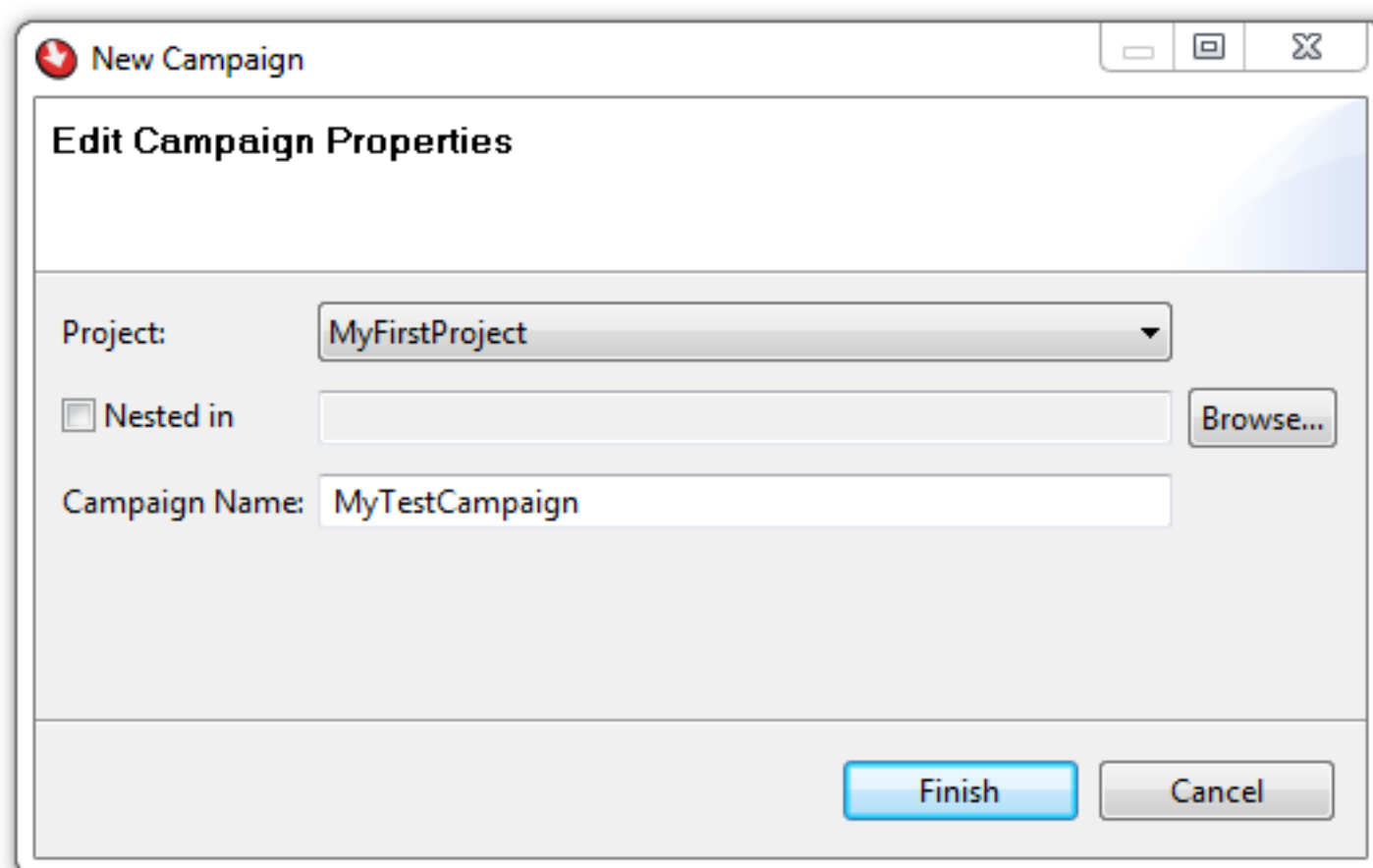
A **Test Campaign** consists of one or more **Test Cases** put together into a test sequence. All **Test Cases** belonging to a particular **Campaign** are listed under the **Campaign** name folder in the order that they are performed when the **Campaign** is executed.

Add a Campaign

MESSINA **Campaigns** are created and edited using the **Designer** view. From the main menu, open the dialog **File** → **New** → **Campaign**. It is also possible to open the **File** → **New** → **Other** dialog and select the **Campaign wizard** in the MESSINA folder.

Alternatively, the **Campaign Wizard** can be called using the **Context Menu** → **New Wizards** → **Campaign**.

Note: When calling a wizard from the context menu, the desired wizard is not always shown directly. The options available in the context menu depend on the current perspective. Choose the **Other** option which opens the **Select a Wizard** dialog box shown below and select **MESSINA** → **Campaign**. The following picture shows the **New Campaign** dialog.



The Project pull-down list can be used to select the project where the **Campaign** is to be created. All projects listed in the [Project Explorer](#) will be available in the pull-down list. The currently active project

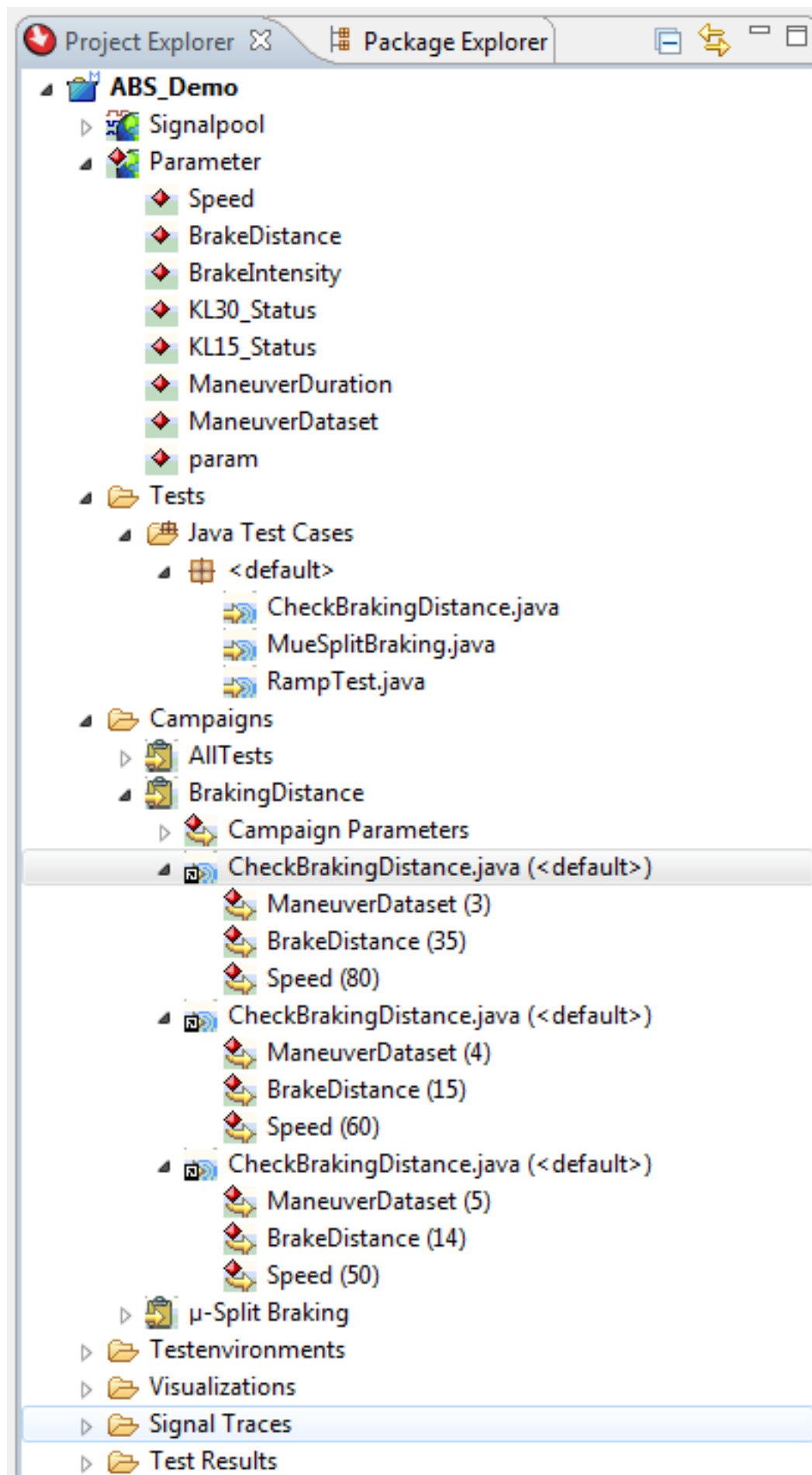
is automatically set as the selected item. It is possible to "nest" campaigns (have a campaign within another campaign). If this is required, the ***Nested in*** box must be checked, and the location where the new ***Campaign*** is to be created must be selected by hand.

The ***Campaign Name*** text box is used to enter the ***Campaign*** name. This will be the name displayed in the ***Project Explorer***. A sample name is always generated automatically, but this can be changed to any ***Campaign*** name that does not already exist in the project. Press ***Finish*** to create the ***Campaign***.

Note: Blank spaces are not allowed in the name.

Once the ***Campaign*** is created, any existing ***Test Cases*** can be added to the ***Campaign*** by marking the ***Test Case*** and dragging them to the ***Campaign*** item directly in the ***Project Explorer*** view. It is also possible to change the order of the Test Cases by drag-and-drop (select a Test Case and drag it to a new position within the Campaign).

An example of a project with several ***Campaigns*** and ***Parameters*** set for ***Test Cases*** within the ***Campaign*** is shown in the picture below.

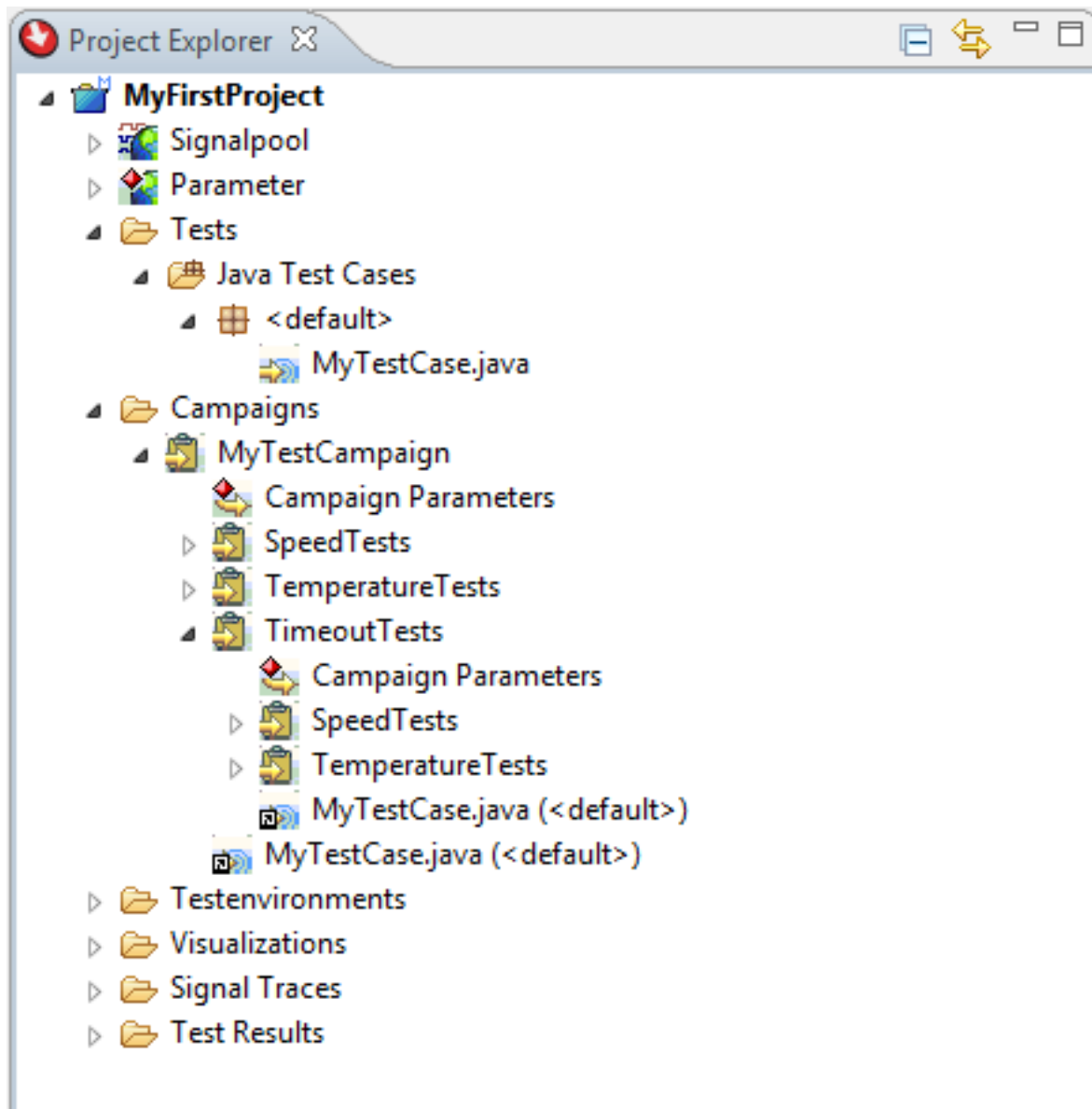


The above example shows a project with several campaigns (**AllTests**, **BrakingDistance** and **u-SplitBraking**), each consisting of different **Test Cases**. Note that it is also possible to add **Parameters** to a **Campaign**. This feature allows different settings to be made at the **Campaign** level. The example shown above uses a different value for the **ManeuverDataset**, **BrakeDistance** and **Speed** parameters for each of the 3 **CheckBrakingDistance Test Cases** within the **BrakingDistance Campaign**.

Nested Campaigns

MESSINA supports the use of nested campaigns. This feature allows test processes to be organized and executed in logical groups. The picture below shows an example of how campaigns can be

organized into logical groups designed to test specific parameters.



The campaigns can be moved and organized easily with the standard drag and drop operation.

Nested campaigns offer the following execution possibilities:

1. execute all campaigns: MyTestCampaigns
2. execute all SpeedTests
3. execute all TemperatureTests
4. execute SpeedTests and TemperatureTests as part of a TimeoutTest campaign

Executing Tests

Executing tests in MESSINA is done using the [Project Explorer](#) view from the **Executor** perspective.

Execution can apply to **Test Cases** or **Campaigns**. To execute a **Test Case** or **Campaign**, select the desired item and use the **Context Menu** → **Execute** option as shown in the picture below.

There are two options listed for test execution: **Execute** or **Execute 1x on default target**.

Selecting the **Execute 1x on default target** option requires that the default target is connected and that the configuration has been started. The selected item (**Test Case** or **Campaign**) is executed directly. The results can be seen in the [Result Manager](#) view.

Selecting the **Execute** option from the context menu will call the following dialog:

Execution count: the number of times to execute the **Test Case** or **Campaign**.

Select Target: The target where the **Test Case** or **Campaign** will run (only available targets are listed)

Select Configuration: the **Testenvironment** used to run the **Test Case** or **Campaign**

Log level: the level for logging

Restart Target: when active, this will cause the target to be restarted before execution.

Restart Configuration before each Test Case Execution: self explanatory

Connect button: Connects the target before execution

Pressing the **OK** button will start the execution. It will be repeated the number of times specified in the **Execution Count** text box.

The results can be seen in the [Result Manager](#) view as the **Test Cases** and **Campaigns** are completed.

Signal Commands

- [waitTrigger\(\)](#)
- [waitValue\(\)](#)
- [setValue\(\)](#)
- [getValue\(\)](#)
- [getLocalValue\(\)](#)
- [register\(\)](#)
- [registerNotification\(\)](#)
- [log\(\)](#)
- [logValue\(\)](#)
- [Coregion\(\)](#)
- [sleep\(\)](#)
- [ASSERT\(\)](#)
- [ASSERT_EQUALS\(\)](#)

waitTrigger()

Function Name: `waitTrigger`

Description:

This function waits for the trigger condition before returning. If a timeout occurs, the function returns false, otherwise it returns true.

Declaration Syntax: `signalname.waitTrigger (value, timeout, slope)`

Parameters:

1. value (data type): defines the trigger level
2. timeout (long): timeout period in milli-seconds
3. slope (boolean): **true** = positive, **false** = negative

Return Value:

- **false**: TIMEOUT
- **true**: TRIGGERED

Valid Data Types: All signal types except byte array

Example:

The following example code can be used to set a trigger condition to occur when the `TestSignal` value changes from less than 100 to greater or equal to 100. If the trigger condition is met the function `waitTrigger` will return **true**. If the timeout is reached the function will return **false**.

```
boolean bRet;  
  
bRet = TestSignal.waitTrigger(100, 3000, true);
```

Alternative:

```
ASSERT(TestSignal.waitTrigger(100, 3000, true));
```

waitValue()

Function Name: `waitValue`

Description:

This function waits for a specified value before returning. If a timeout occurs, the function returns **false**, otherwise it returns **true**.

Declaration Syntax (overloaded):

```
signalname.waitValue (value, timeout)
```

```
signalname.waitValue (value, timeout, tolerance)
```

Parameters:

1. value (data type): defines the trigger level
2. timeout (long): timeout period in milli-seconds
3. tolerance (data type): signal tolerance

Return Value:

- **false**: TIMEOUT
- **true**: Value Reached

Valid Data Types: All signal types except byte array, the **tolerance** parameter can not be of type boolean.

Example:

The following example code demonstrates a case that will wait until the `TestSignal` reaches 100 (tolerance ± 10). The `waitValue` function will return **true** when this occurs. If the timeout is reached the function will return **false**.

```
boolean bRet;
```

```
bRet = Testsignal.waitValue(100, 3000, 10);
```

Alternative:

```
ASSERT (Testsignal.waitValue(100, 3000, 10));
```

setValue()

Function Name: `setValue`

Description:

This function sets the value of the specified signal the value of the parameter passed.

Declaration Syntax: `signalname.setValue (value)`

Parameters:

1. `value`: the signal will be set to this value

Return Value:

- ***false***: Error
- ***true***: OK

Valid Data Types: The type of the value must match the type of the signal

Example:

The following example code will set the `TestSignal` value to 10.0 :

```
boolean bRet;  
  
bRet = Testsignal.setValue(10.0);
```

Alternative:

```
ASSERT (Testsignal.setValue(10.0));
```

getValue()

Function Name: `getValue`

Description:

This function is used to read the current value of a signal. Calling this function will send a request to the signalpool and waits for the response. In the case that the signal value should be immediatly available in the test case it is recommended to use [getLocalValue\(\)](#) (in combination with [register\(\)](#)) instead

Declaration Syntax: `signalname.getValue ()`

Parameters: none

Return Value: value of signal

Valid Data Types: the type of the return value must match the type of the signal

Example:

The following example code will read the current value of `TestSignal` and store it in `newValue`. In this example it is assumed that `TestSignal` and `newValue` are of type double (the `newValue` type and the `TestSignal` type must match):

```
double newValue;  
  
newValue = Testsignal.getValue();
```


getLocalValue()

Function Name: `getLocalValue`

Description:

This function is used to read the value of the local representation of a signal in the test case instance. Compared to `getValue()` no request of getting the value is sent to the Signalpool instead the last known value in the test case instance is used (local cache). This function is normally used after registering the signal (see [register\(\)](#)). Registering keeps the local cache up to date (i.e., every signal change will be pushed to this test case cache).

Declaration Syntax: `signalname.getLocalValue ()`

Parameters: none

Return Value: value of signal

Valid Data Types: the type of the return value must match the type of the signal

Example:

The following example code will read the cached value of `TestSignal` and store it in `newValue`. In this example it is assumed that `TestSignal` and `newValue` are of type double (the `newValue` type and the `TestSignal` type must match):

```
Testsignal.register()  
  
...  
  
double newValue;  
  
newValue = Testsignal.getLocalValue();
```

register()

Function Name: `register`

Description:

This function is used to register for signal change notifications from at the signal pool. After calling register, all signal changes will be sent from the signal pool to the test case and stored in the test case's local cache. the function can be used in comination with [getLocalValue\(\)](#) to access signal values in a fast way.

Declaration Syntax: `signalname.register()`

Parameters: none

Return Value: none

Example:

The following example code will read the cached value of `TestSignal` and store it in `newValue`. In this example it is assumed that `TestSignal` and `newValue` are of type double (the `newValue` type and the `TestSignal` type must match):

```
Testsignal.register()  
  
...  
  
double newValue;  
  
newValue = Testsignal.getLocalValue();
```

registerNotification()

Function Name: `registerNotification`

Description:

This function is used to place a callback function for signal changes. This callback function will be called on every change of the signal value until `unregisterNotification()` is called or the test case ends. The callback will be called independent of the current flow of the test case. Since this callback is called in the context of the signal notification from the signalpool, calls of wait Operations and `getValue()` is not allowed in the callback.

Declaration Syntax: `signalname.registerNotification()`

Parameters: The interface instance of the callback class

Return Value: none

Example:

The following example code will start the callback function of `air_temp`. In the case that subsequent changes of `air_temp` happen, the loginfo will be written to the testlog:

```
air_temp.registerNoitification(new ISignalValueChangedReceiver() {  
  
    public void OnValueChanged(SignalValue arg0) {  
  
        log(ITargetLogger.INFO, "Air temp has changed to " + air_temp.getLocalValue());  
  
    }  
  
});
```

log()

Function Name: log

Description:

This function is used to create an entry in the [Log View](#) for the currently active target during a running test process.

Declaration Syntax: log(logLevel, user message)

Parameters:

- 1. logLevel (integer/long): the log level used for the required signal. Possible log levels are all, trace, debug, info, warning, error and fatal.
- 2. user message (string): user message text that is displayed in the **Log View**

Constants for setting the **Log Level** in a **Test Case**:

Log Level	Test Case Constant Name	Description
All	ITargetLogger.ALL	all logging functions are active
Trace	ITargetLogger.TRACE	logs the stage (e.g. pre-condition, run, post-condition) and signal changes
Debug	ITargetLogger.DEBUG	User defined logging of messages and values in the Test Case (e.g. function call: log(ITargetLogger.DEBUG, "hello")
Info	ITargetLogger.INFO	logs the test status (e.g. done) and the return value
Warning	ITargetLogger.WARNING	makes a log entry when a "wait" operation fails (e.g. waitTrigger, waitValue)
Error	ITargetLogger.ERROR	makes a log entry when an error occurs
Fatal	ITargetLogger.FATAL	makes a log entry when a fatal error (exceptions) occurs
Off (default)	-	nothing is logged

Return Value: -none-

Valid Data Types: string

Example:

The following example code will cause the user message "hello" to be added to the [Log View](#) display at the TRACE log level.

```
log(ITargetLogger.TRACE, "hello");
```

logValue()

Function Name: logValue

Description:

This function is used to create an entry in the [Log View](#) for the currently active target during a running test process.

Declaration Syntax: signalname.logValue(logLevel)

Parameters:

- 1. logLevel (pre-defined constant): the log level used for the required signal. Possible log levels are all, trace, debug, info, warning, error and fatal.

Constants for setting the **Log Level** in a **Test Case**:

Log Level	Test Case Constant Name	Description
<i>All</i>	ITargetLogger.ALL	all logging functions are active
<i>Trace</i>	ITargetLogger.TRACE	logs the stage (e.g. pre-condition, run, post-condition) and signal changes
<i>Debug</i>	ITargetLogger.DEBUG	User defined logging of messages and values in the Test Case (e.g. function call: log(ITargetLogger.DEBUG, "hello")
<i>Info</i>	ITargetLogger.INFO	logs the test status (e.g. done) and the return value
<i>Warning</i>	ITargetLogger.WARNING	makes a log entry when a "wait" operation fails (e.g. waitTrigger, waitValue)
<i>Error</i>	ITargetLogger.ERROR	makes a log entry when an error occurs
<i>Fatal</i>	ITargetLogger.FATAL	makes a log entry when a fatal error (exceptions) occurs
<i>Off (default)</i>	-	nothing is logged

Return Value: -none-

Valid Data Types: integer (long)

Example:

The following example code will cause the value of `TestSignal` to be added to the [Log View](#) display at the TRACE log level.

```
Testsignal.logValue (ITargetLogger.TRACE) ;
```

Coregion()

Function Name: `Coregion`

Description:

This function is used to execute one or more `wait*()` functions (e.g. `waitValue`, `waitTrigger`) simultaneously. If at least one of the `wait*()` functions returns a FAIL, then the `Coregion` function returns a FAIL. The process of using this function involves instantiating it, adding the `wait*()` functions, and then running the `Coregion` function. The evaluation of all `wait*()` functions begins simultaneously when the `Coregion` function is run.

Declaration Syntax: `Coregion myCoregionName = new Coregion ();`

wait* () function Syntax: see `waitTrigger` and `waitValue` function descriptions

Function run Syntax: `myCoregionName.run ()`

Parameters:

1. expression (boolean): any valid boolean expression
2. (optional) user message (string): user message text that is displayed when returning with a FAIL

Return Value:

- true: all `wait*()` functions are true
- false: one or more `wait*()` functions are false

Valid Data Types: not relevant

Example:

The following example code will create an instance of a `Coregion` function, add several `wait*()` functions, and then run the `Coregion` function. Once started, the `Coregion` function will only return false and display the message "One signal failed" if one or more of the `wait*()` functions returns a FAIL.

```
Coregion myCoregionName = new Coregion ();
```

```
// define wait*()-functions whose execution will start simultaneously
```



```
myCoregionName.addWaitValue(bTestsignal, true, waitTime);
```

```
myCoregionName.addWaitTrigger(bTestsignal, waitTime, true);
```

```
myCoregionName.addWaitValue(fTestsignal, 100.0f, waitTime, 10.0f);
```

```
myCoregionName.addWaitTrigger(fTestsignal, 100.0f, waitTime, true);
```

```
myCoregionName.addWaitValue(i32Testsignal, 100, waitTime, 10);
```

```
myCoregionName.addWaitTrigger(i32Testsignal, 100, waitTime, true);
```

```
ASSERT(myCoregionName.run(), "One signal failed");
```

sleep()

Function Name: `sleep`

Description:

This function "sleeps" for the specified time given in milli-seconds.

Declaration Syntax: `sleep (milli-seconds)`

Parameters:

1. value (long): milli-seconds

Return Value: -none-

Valid Data Types: long

Example:

The following example code demonstrates a sleep of 5 seconds:

```
sleep (5000);
```

ASSERT()

Function Name: ASSERT

Description:

This function returns from the current **Test Case** with a FAIL and creates a log entry if the expression (first parameter) is false. A user message can be shown when returning with a FAIL.

Declaration Syntax (overloaded):

```
ASSERT (expression)
```

```
ASSERT (expression, user message)
```

Parameters:

1. expression (boolean): any valid boolean expression
2. (optional) user message (string): user message text that is displayed when returning with a FAIL

Return Value:

- **true**: expression is true
- **false**: expression is false

Valid Data Types: boolean, string (optional)

Example:

The following example code will return from the **Test Case** with FAIL, make a log entry, and show the user message "Value not reached" if the value of TestSignal does not reach 100 (tolerance ± 10) within the 3000 milli-second timeout.

```
ASSERT(Testsignal.waitValue(100, 3000, 10), "Value not reached");
```

ASSERT_EQUALS()

Function Name: ASSERT_EQUALS

Description:

This function returns from the current **Test Case** with a FAIL and creates a log entry if the expected value and current value are not equal. A user message can be shown when returning with a FAIL. The data type for the current value and expected value can be either long, double, or boolean, but they must be the same.

Declaration Syntax (overloaded):

```
ASSERT_EQUALS (expected value, current value)
```

```
ASSERT_EQUALS (expected value, current value, user message)
```

Parameters:

1. expected value (long, double, boolean): expected value used for comparison to the current value
2. current value (long, double, boolean): current value used for comparison to the expected value
3. (optional) user message (string): user message text that is displayed when returning with a FAIL

Return Value:

- **true**: expected value and current value are equal
- **false**: expected value and current value are NOT equal

Valid Data Types: long, double, string (optional)

Example:

The following example code will compare the current value of `TestSignal` to 42. If they are equal the function will return **true**, if they are NOT equal, the function will return **false**, make a log entry, and show a user message reading "Value is not 42".

```
ASSERT_EQUALS(42, Testsignal.getValue(), "Value is not 42");
```

Visualization

Control Panels

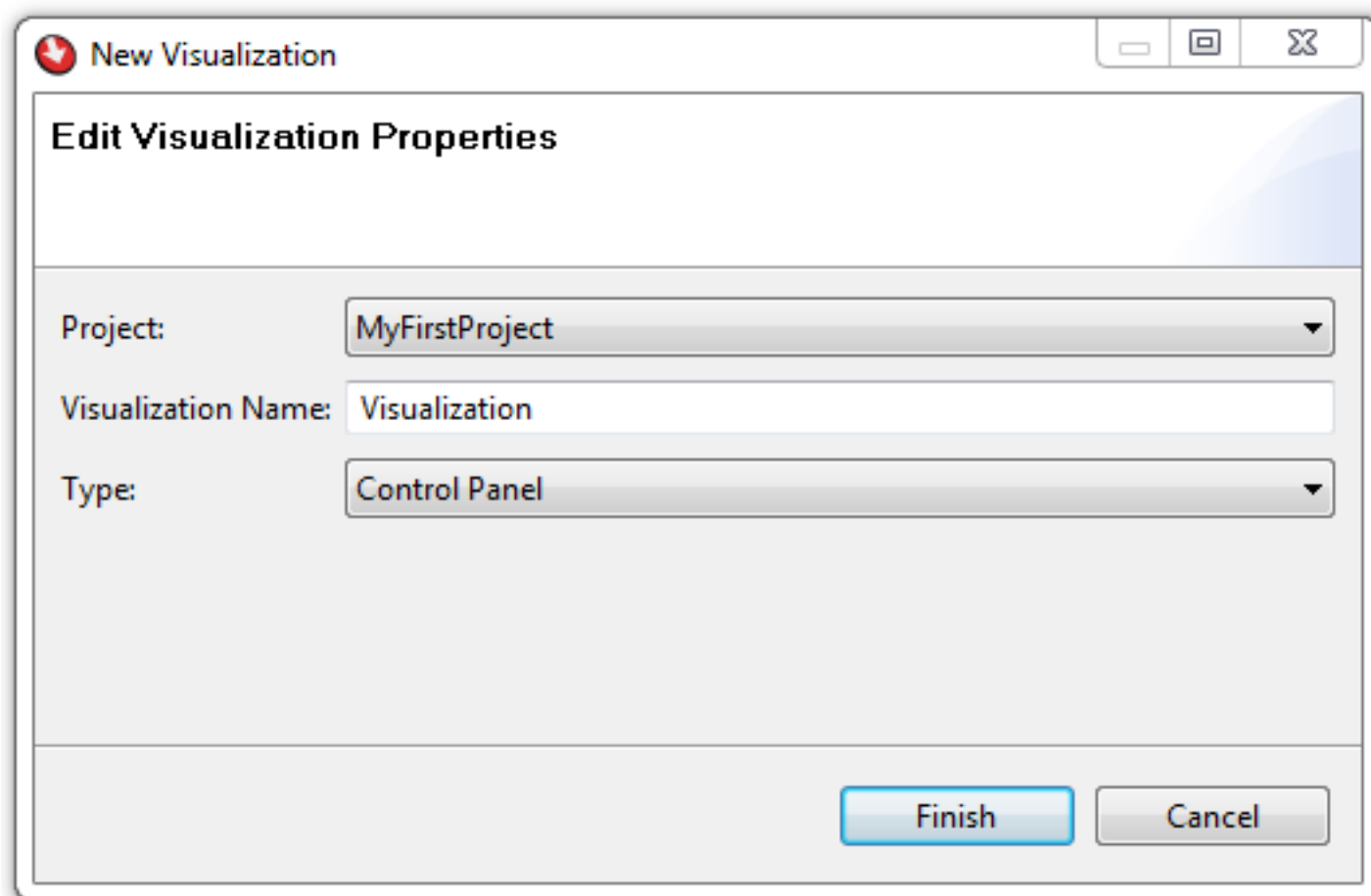
The MESSINA **Control Panel** visualization offers many operation and display elements which can be used for controlling a model or **Test Case**, monitoring values and displaying test results. Signals can be connected to the various elements by drag-and-drop from the signalpool. The sections below describe the header icons and the available control elements.

Adding a Control Panel Visualization

MESSINA **Visualizations** are created and edited using the Executor perspective which can be selected using the tabs at the top right of the main window. From the main menu, open the dialog **File** → **New** → **Visualization**. It is also possible to open the **File** → **New** → **Other** dialog and select the **Visualization Wizard** in the MESSINA folder.


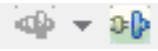



Alternatively, the **Visualization Wizard** can be called using the **Context Menu** → **New Wizards** → **Visualization**.

The following dialog box appears when the **Visualization Wizard** is called:










Select the **Control Panel** item from the **Type** pull-down list and press **Finish**. The empty **Control Panel visualization** is displayed.

Header Icons

-  Connect visualization (current state is disconnected): Pressing the active icon will connect the current visualization with the default target
-  Disconnect visualization (current state is connected): Pressing the active icon will disconnect the current visualization
-  Change to Edit Mode: Change to or exit Edit Mode.
-  Remove signals: Remove all signals from all control elements
-  Remove controls: Remove all controls from the control panel

Available Controls

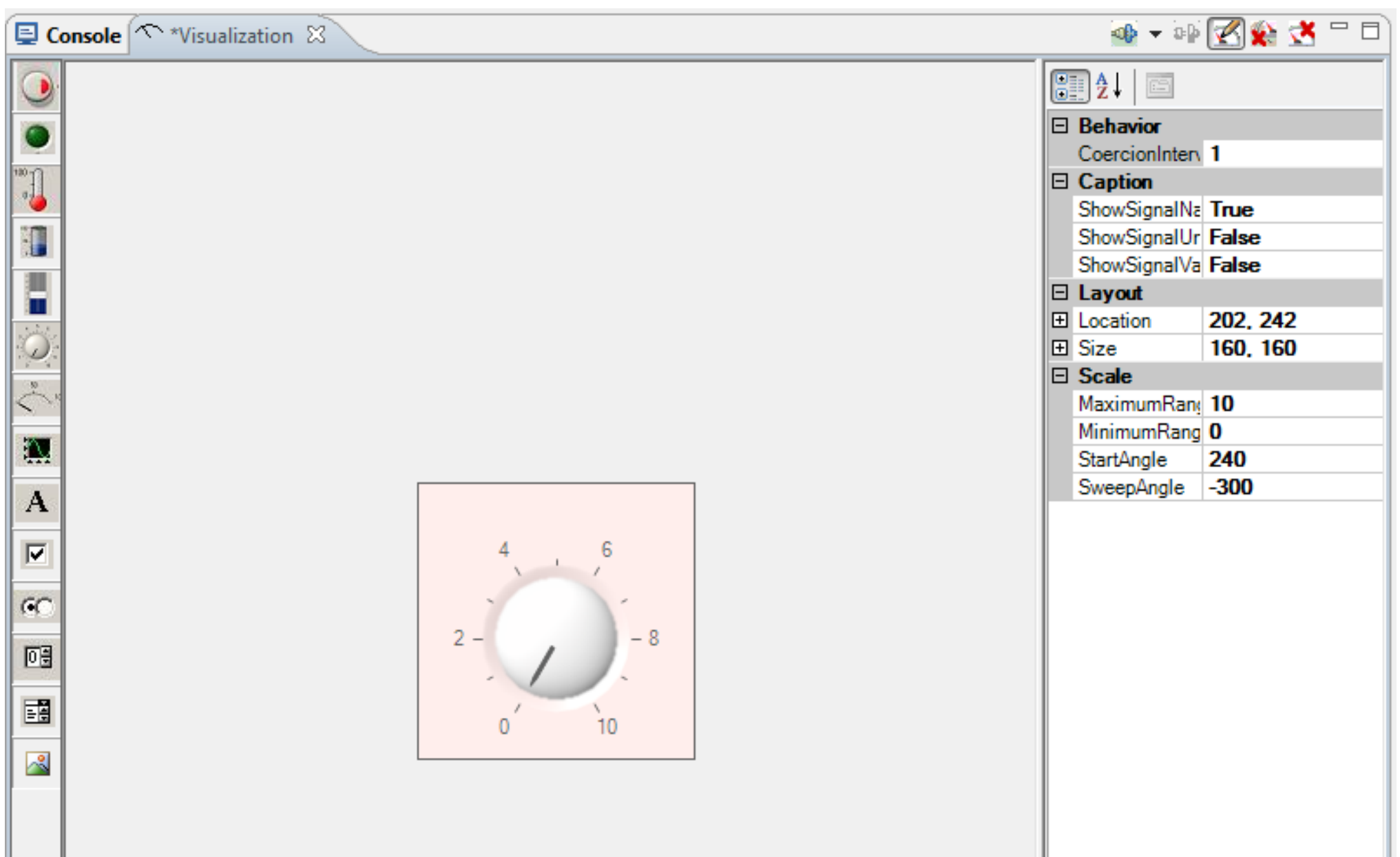
The following controls are available. Refer to the matching section in this documentation for a detailed explanation of each control.

-  Button
-  LED
-  Thermometer
-  Tank
-  Slider
-  Dial/Knob
-  Meter
-  Graph
-  Label
-  Checkbox
-  Radio buttons
-  Numeric Edit

-  Combo box
-  Image

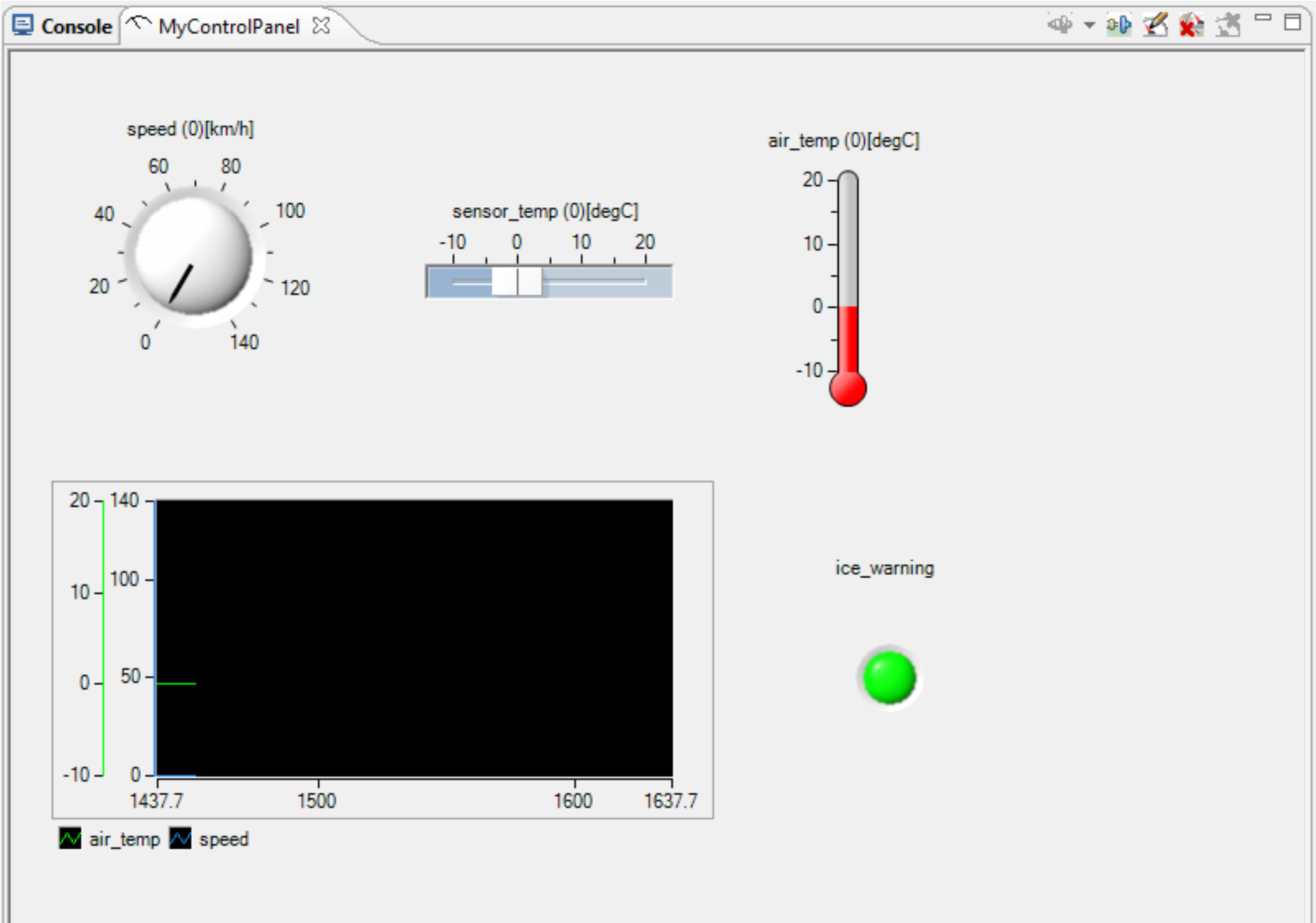
Control Properties

The display elements have a pink background in the **Edit Mode**. This indicates that the control has not been connected to a signal. Each control element has a set of properties associated with it. These can be changed/modified as required by the application. Properties for a control can only be set/changed in the **Edit Mode** when the desired control is selected. An example of the properties for the Dial/Knob control is shown below.

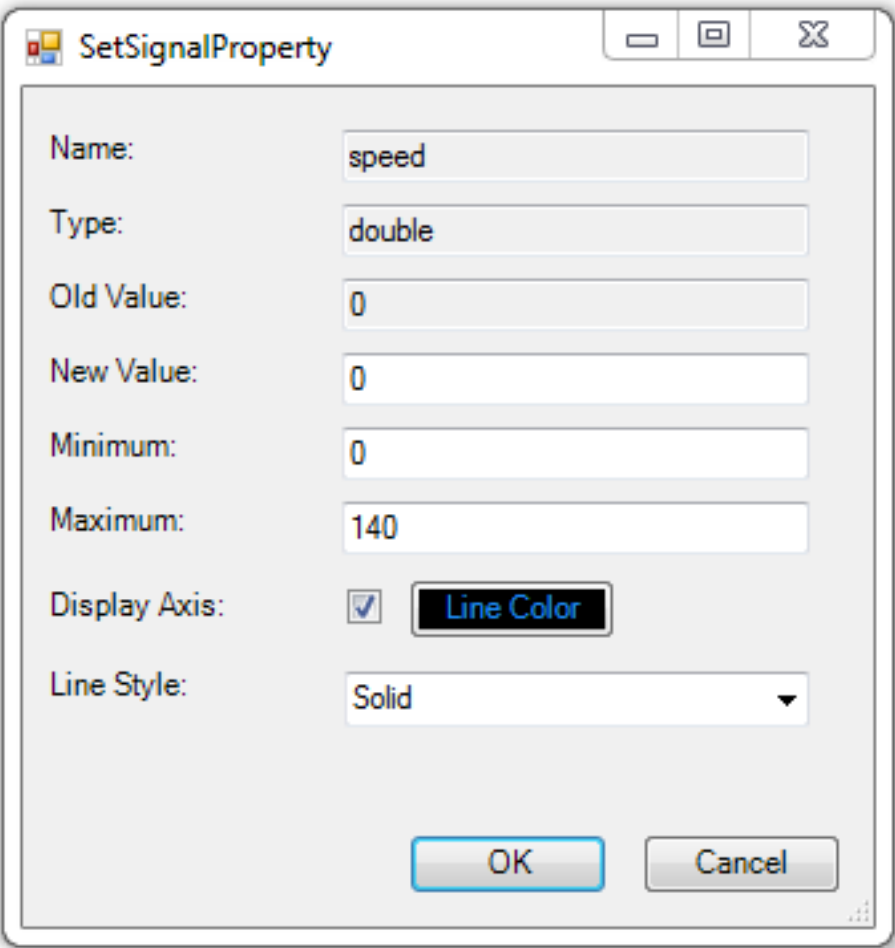


The properties can be set/changed by selecting them and making the desired modification. A brief description of each property is displayed at the bottom of the property list.

Signals can be attached to each control element by drag-and-drop. An example of a **Control Panel** visualization with signals attached is shown below.



Note: to set the scaling on the graph element, double click on the desired signal in the legend (bottom left). The following dialog appears.



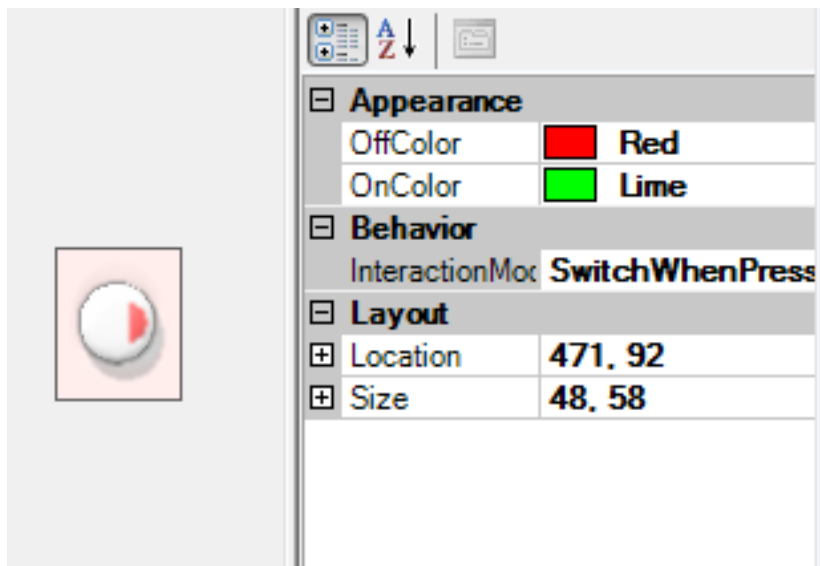
Enter the desired limits and start value and press **OK**. The scale of the graph is updated.

See the [Graph Display](#) section of this documentation for a more detailed description of graph visualizations.

Button

Icon: 

Control Panel element and available settings:



Appearance: Defines the *On* and *Off* colours.

Layout: Defines the location and size of the button.

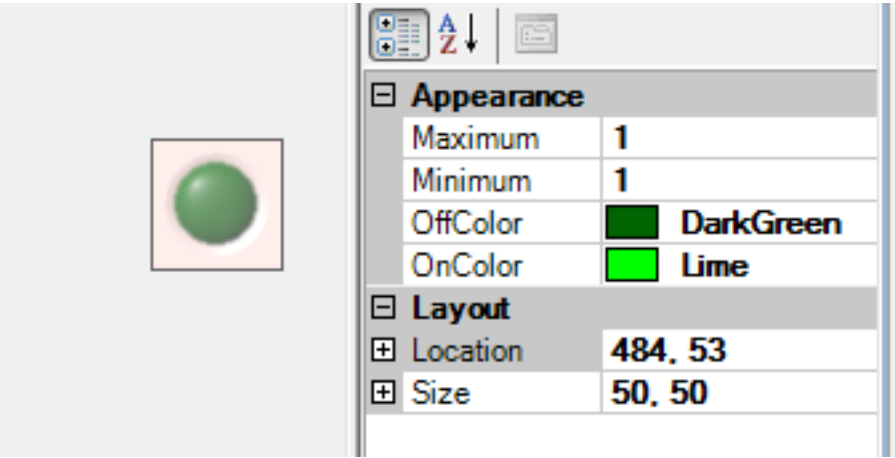
Operation: Defines the mouse handling mechanism for the button

- SwitchWhenPressed: The switch toggles when pressed
- SwitchUntilReleased: The switch toggles when pressed and remains in the same state until releasing the mouse button.
- SwitchWhenReleased: The switch toggles when the mouse button is released
- Indicator (display only): The switch displays the current state of the signal. Its state can not be changed by the user.

LED

Icon: 

Control Panel element and available settings:



Appearance: Defines the *On* and *Off* colours.

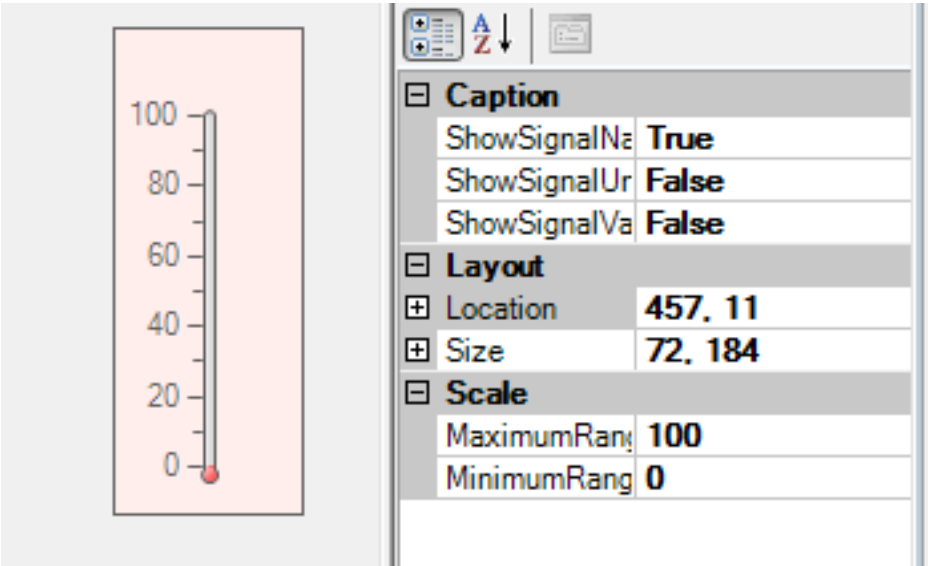
Layout: Defines the location and size of the button.

Operation: This control element will change state depending on the value of the signal mapped to it. Only *On* or *Off* are possible, therefore the signal must be of type boolean.

Thermometer

Icon: 

Control Panel element and available settings:



Caption:

- ShowSignalName: When **True**: the name of the signal mapped to this control is displayed, when **False**: signal name is not displayed
- ShowSignalUnit: When **True**: the unit of the signal mapped to this control is displayed, when **False**: units are not displayed
- ShowSignalValue: When **True**: the value of the signal mapped to this control is displayed, when **False**: the value is not displayed

Scale:

- MaximumRange: Defines the maximum value allowable for this control element.
- MinimumRange: Defines the minimum value allowable for this control element.

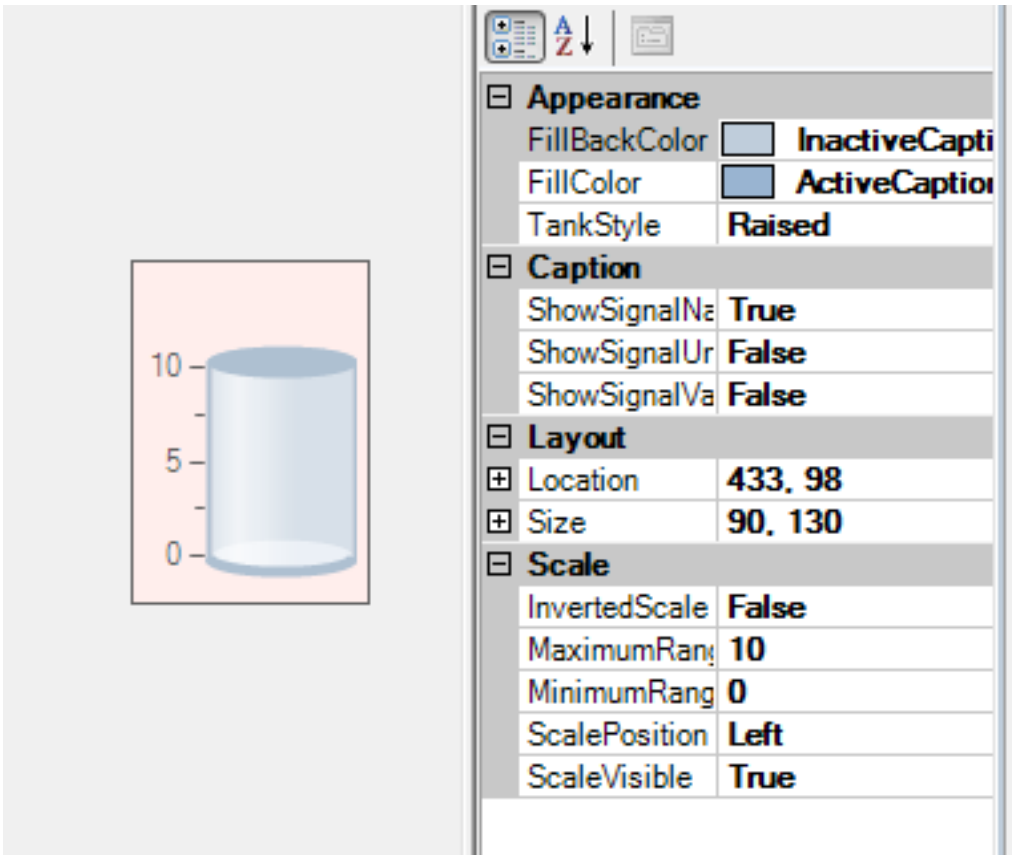
Layout: Defines the location and size of the thermometer.

Operation: This control is used for display only. It displays the current value of the signal mapped to it.

Tank

Icon: 

Control Panel element and available settings:



Caption:

- ShowSignalName: When **True**: the name of the signal mapped to this control is displayed, when **False**: signal name is not displayed
- ShowSignalUnit: When **True**: the unit of the signal mapped to this control is displayed, when **False**: units are not displayed
- ShowSignalValue: When **True**: the value of the signal mapped to this control is displayed, when **False**: the value is not displayed

Appearance: Defines the **Fill** and **Background** colours and Tank appearance.

Scale:

- InvertedScale: Defines whether the scale is from top to bottom, or from bottom to top
- MaximumRange: Defines the maximum value allowable for this control element.
- MinimumRange: Defines the minimum value allowable for this control element.
- ScalePosition: Defines the location of the scale in relation to the tank object
- ScaleVisible: Defines whether the scale is visible or not

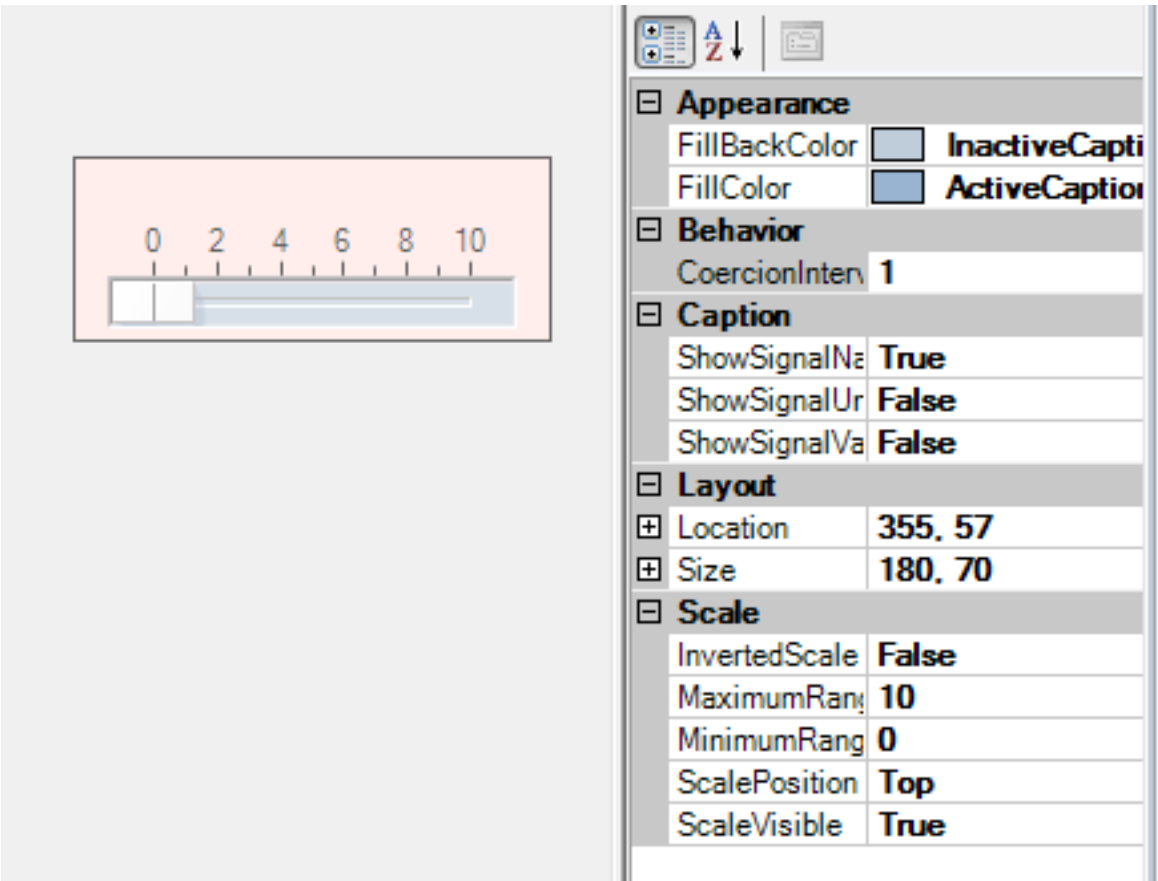
Layout: Defines the location and size of the tank.

Operation: This control is used for display only. It displays the current value of the signal mapped to it.

Slider

Icon: 

Control Panel element and available settings:



Caption:

- ShowSignalName: When **True**: the name of the signal mapped to this control is displayed, when **False**: signal name is not displayed
- ShowSignalUnit: When **True**: the unit of the signal mapped to this control is displayed, when **False**: units are not displayed
- ShowSignalValue: When **True**: the value of the signal mapped to this control is displayed, when **False**: the value is not displayed

Appearance: Defines the **Fill** and **Background** colours of the slider.

Scale:

- InvertedScale: Defines whether the scale is from top to bottom, or from bottom to top
- MaximumRange: Defines the maximum value allowable for this control element.
- MinimumRange: Defines the minimum value allowable for this control element.
- ScalePosition: Defines the orientation and position of the scale.
- ScaleVisible: When **True**: scale is visible

Layout: Defines the location and size of the slider.

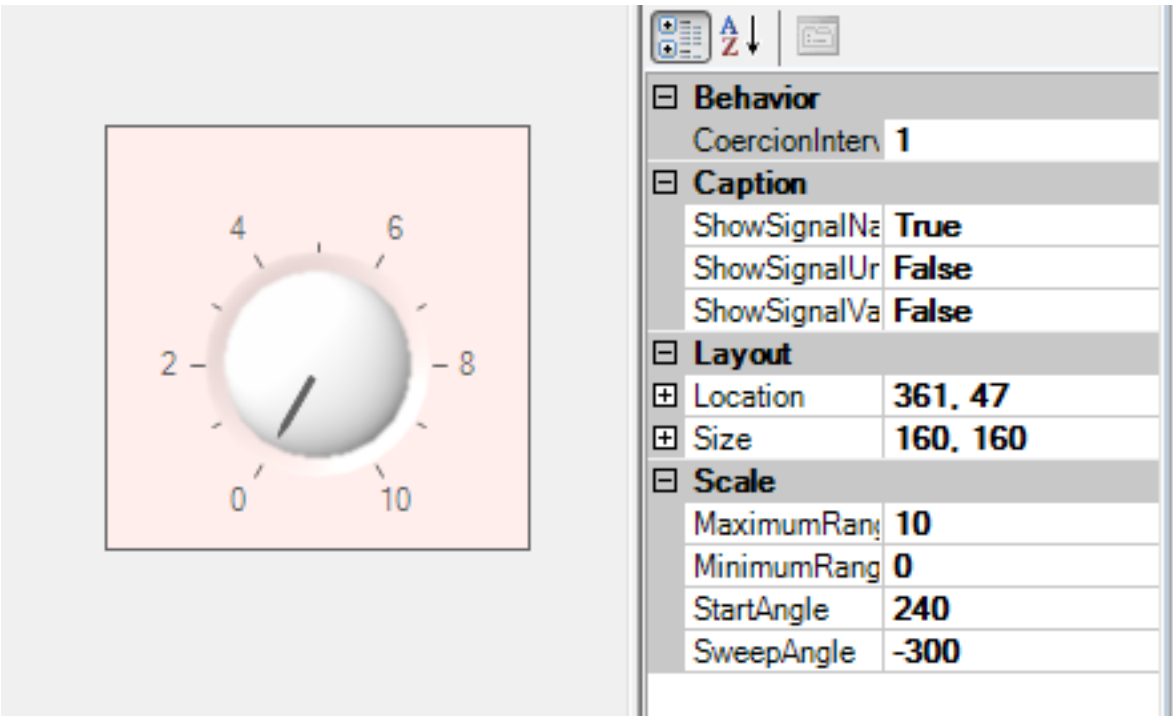
CoercionInterval: The interval used for coercing the value of the control (i.e. with what increments can the control value be set)

Operation: This control can be used to set the value of the signal mapped to it.

Dial

Icon: 

Control Panel element and available settings:



Caption:

- ShowSignalName: When **True**: the name of the signal mapped to this control is displayed, when **False**: signal name is not displayed
- ShowSignalUnit: When **True**: the unit of the signal mapped to this control is displayed, when **False**: units are not displayed
- ShowSignalValue: When **True**: the value of the signal mapped to this control is displayed, when **False**: the value is not displayed

Scale:

- MaximumRange: Defines the maximum value allowable for this control element.
- MinimumRange: Defines the minimum value allowable for this control element.
- StartAngle: Defines the start position of the needle on the dial element.
- SweepAngle: Defines the angle of movement available on the dial between the min and max values.

Layout: Defines the location and size of the button.

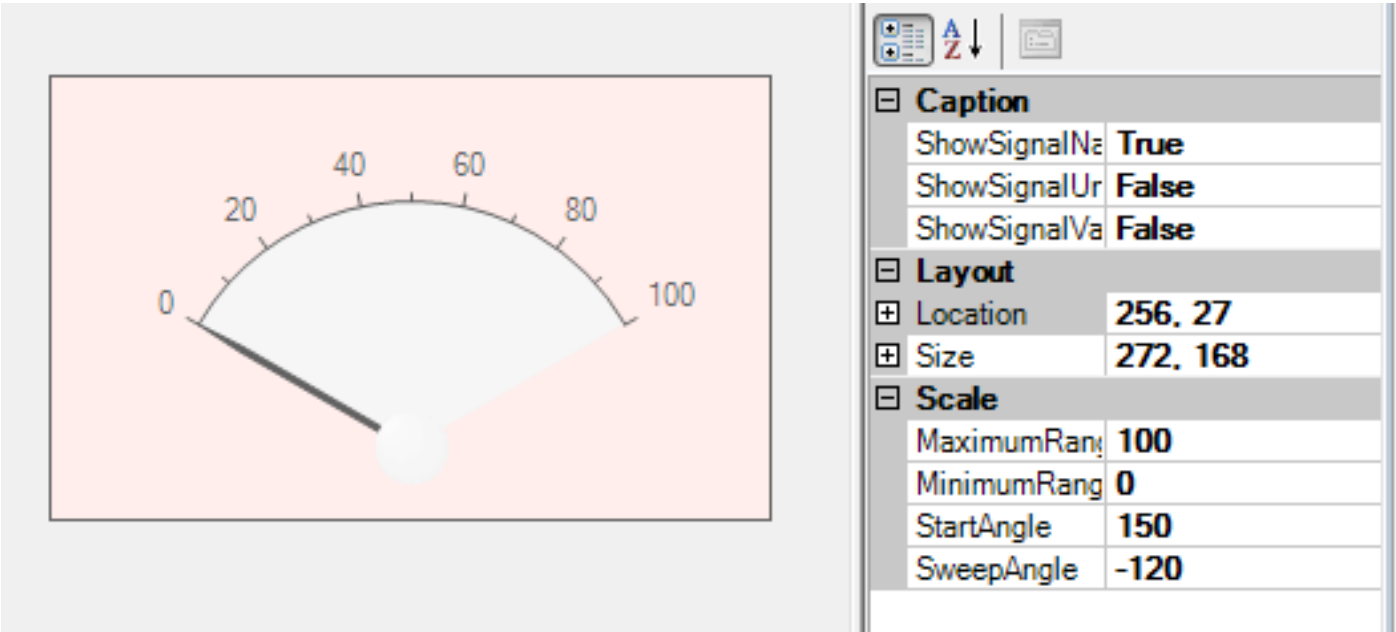
CoercionInterval: The interval used for coercing the value of the control (i.e. with what increments can the control value be set)

Operation: This control can be used to set the value of the signal mapped to it.

Meter

Icon: 

Control Panel element and available settings:



Caption:

- ShowSignalName: When **True**: the name of the signal mapped to this control is displayed, when **False**: signal name is not displayed
- ShowSignalUnit: When **True**: the unit of the signal mapped to this control is displayed, when **False**: units are not displayed
- ShowSignalValue: When **True**: the value of the signal mapped to this control is displayed, when **False**: the value is not displayed

Scale:

- MaximumRange: Defines the maximum value allowable for this control element.
- MinimumRange: Defines the minimum value allowable for this control element.
- StartAngle: Defines the start position of the needle on the meter element.
- SweepAngle: Defines the angle of movement available on the meter between the min and max values.

Layout: Defines the location and size of the meter.

Operation: This control is used for display only. It displays the current value of the signal mapped to it.

Graph

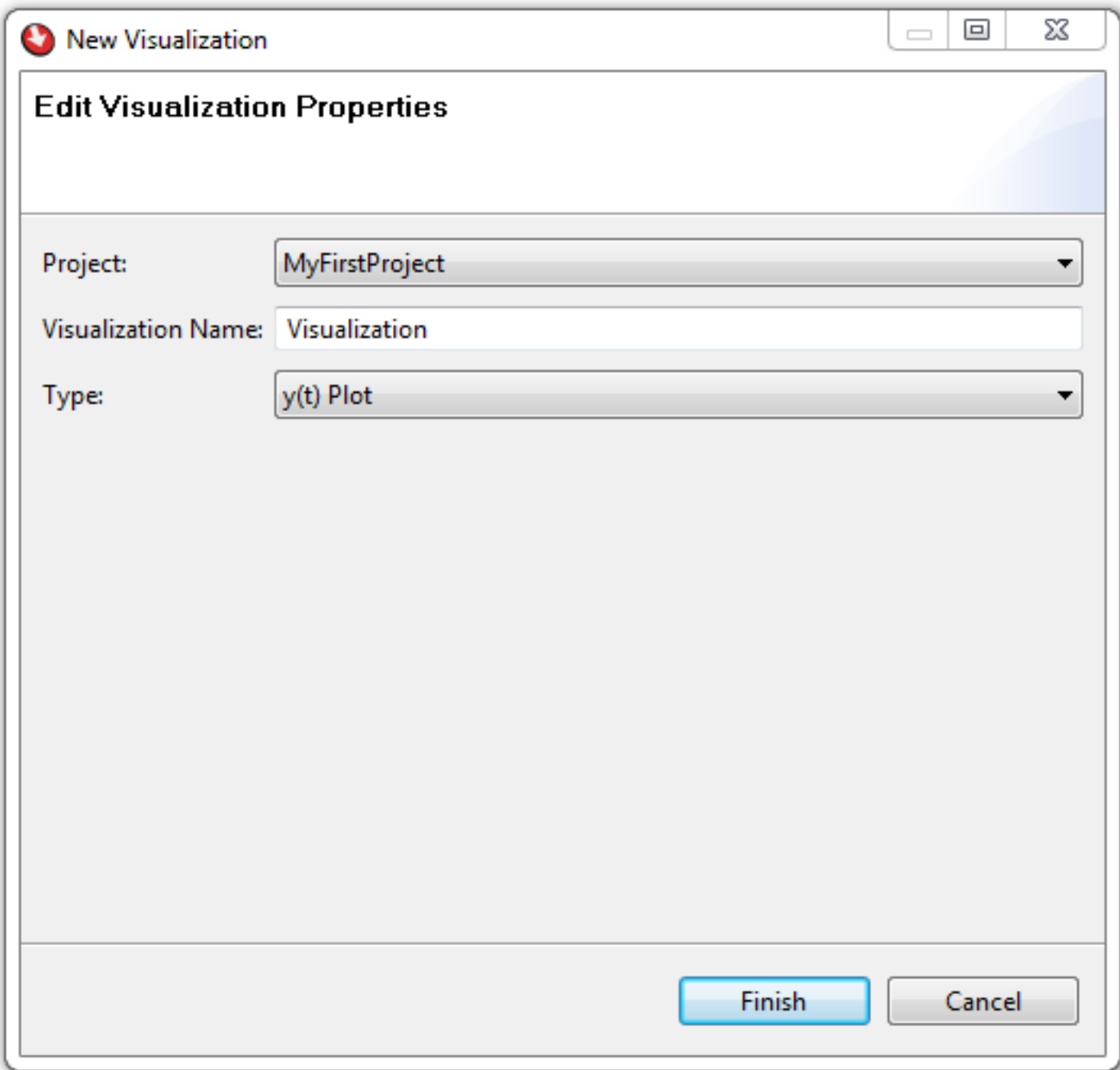
The MESSINA **y(t) Plot** (graph) visualization offers the ability to graphically trace a signal over time. Signals can be connected to the graph by drag-and-drop from the **Signalpool**. The sections below describe the header icons and the graph display.

Adding a y(t) Plot Visualization

MESSINA **Visualizations** are created and edited using the **Executor** perspective which can be selected using the tabs at the top right of the main window. From the main menu, open the dialog **File** → **New** → **Visualization**. It is also possible to open the **File** → **New** → **Other** dialog and select the **Visualization wizard** in the MESSINA folder.

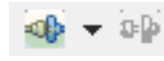
Alternatively, the **Visualization Wizard** can be called using the **Context Menu** → **New Wizards** → **Visualization**.


The following dialog box appears when the **Visualization Wizard** is called:




Select the ***y(t) Plot*** item from the **Type** pull-down list and press **Finish**. The empty visualization is displayed.

Header Icons

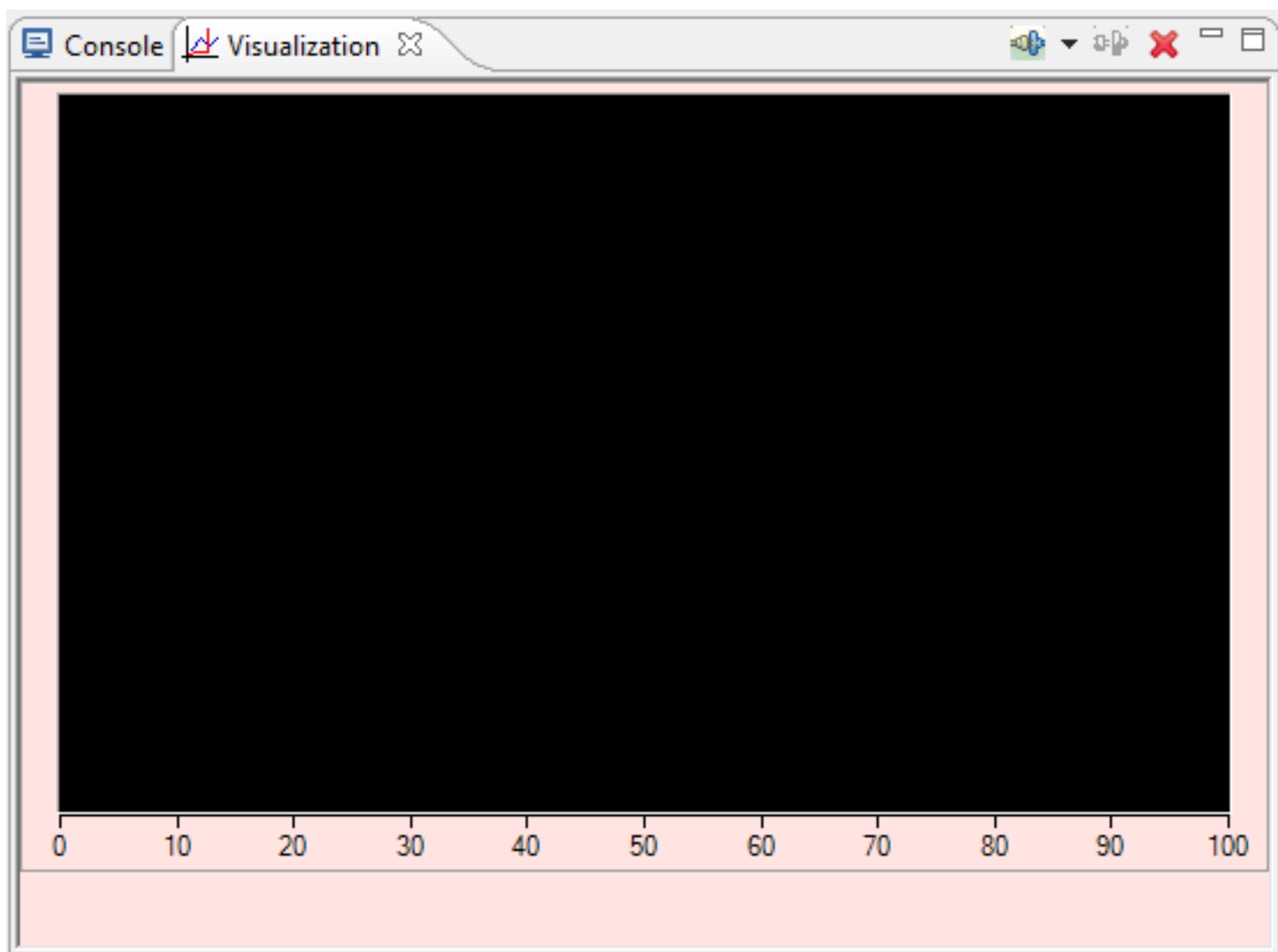
 **Connect visualization** (current state is disconnected): Pressing the active icon will connect the current visualization with the default target. Clicking on the down arrow between the icons allows any available target to be selected.

 **Disconnect visualization** (current state is connected): Pressing the active icon will disconnect the current visualization. Clicking on the down arrow between the icons allows any available target to be selected.

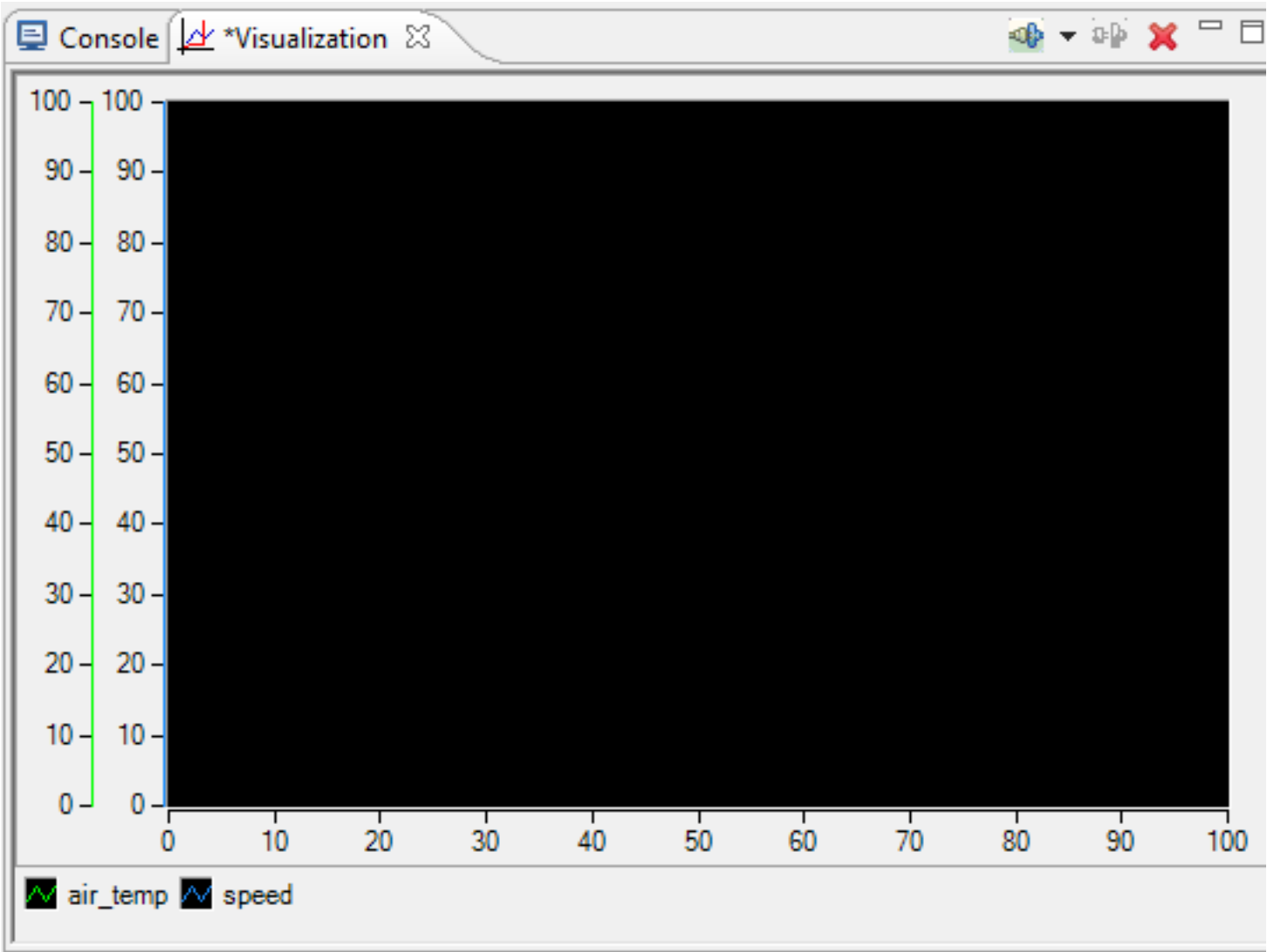
 **Remove all signals**: Pressing this icon will remove all signals from the graph display

Control Properties

The display will first have a pink background to indicate that the control has not been connected to a signal.



Signals can be attached to the graph element by drag-and-drop. An example of a ***y(t) Plot*** visualization with the ***air_temp*** and ***speed*** signals attached is shown below.



Set the Scale of a Signal and Line Properties

Note: These options can also be used for the graph element on [Control Panel](#) visualizations.

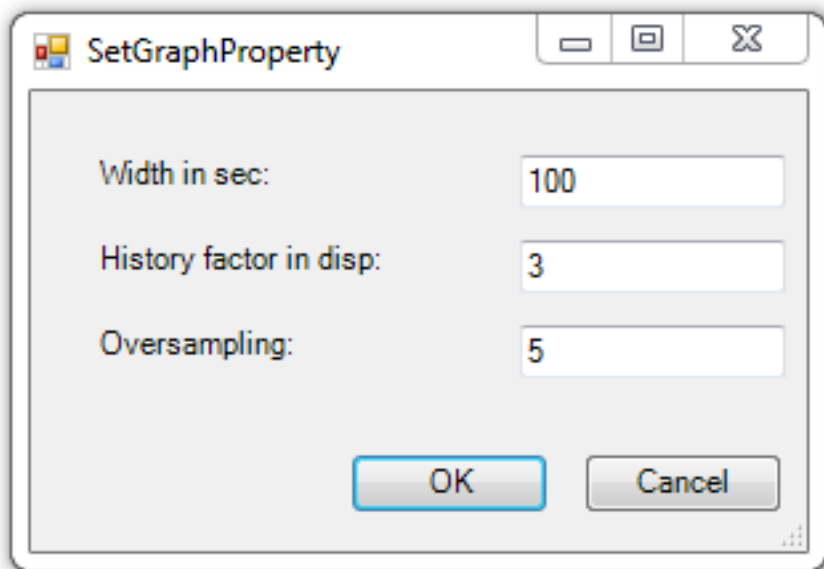
To set the scaling on the graph element, double click on the desired signal in the legend (bottom left). The following dialog appears.

The screenshot shows a dialog box titled 'SetSignalProperty'. It contains several input fields and a checkbox. The 'Name' field is set to 'speed'. The 'Type' field is set to 'double'. The 'Old Value' field is set to '0'. The 'New Value' field is set to '0'. The 'Minimum' field is set to '0'. The 'Maximum' field is set to '100'. The 'Display Axis' checkbox is checked, and a 'Line Color' button is visible next to it. The 'Line Style' dropdown menu is set to 'Solid'. At the bottom of the dialog, there are 'OK' and 'Cancel' buttons.

Enter the desired line properties, limits, and new value and press **OK**. The scale of the graph and the signal line properties are updated.

Set Graph Properties

The properties for a graph can be set by double clicking within the graph element. The following dialog is called:



Make the settings as required and press **OK**. The graph is updated with the new settings.

Panning and Zooming

Panning:

- Press and hold the **CTRL** key, then press and hold the left mouse button. Moving left/right and up/down is done by moving the mouse
- Press and hold the **CTRL** key, then use the arrow keys to scroll left/right or up/down

The scale remains unchanged.

Zooming:

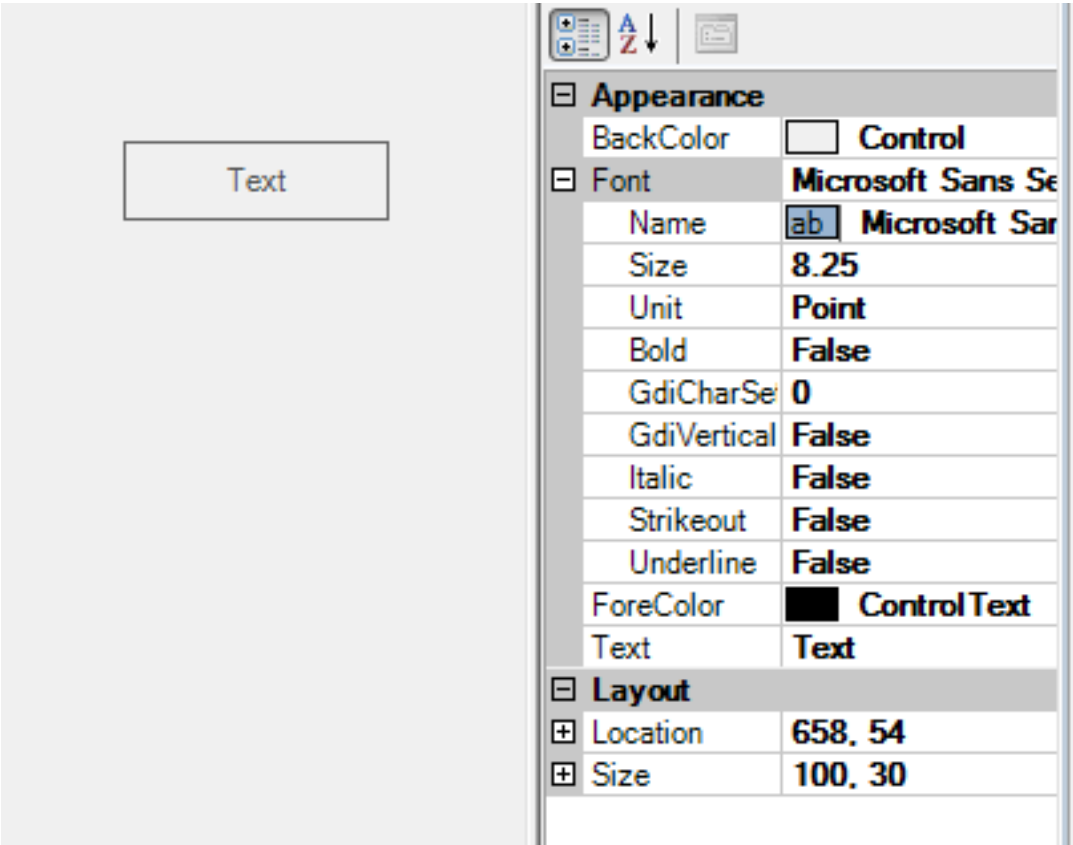
- Press and hold the **Shift** key and turn the mouse wheel
- Press and hold the **Shift** key and use the up/down arrow keys
- Press and hold the **Shift** key, then press and hold the left mouse button to mark an area, then release the left mouse button
- Double click on the desired signal name in the legend (bottom left), then set the min/max values as desired.

The scale can be set back by double clicking on the desired signal name in the legend and setting the min/max values to the normal scale. Each signal must be set separately.

Label

Icon: 

Control Panel element and available settings:



Appearance: Defines the font and text settings and text name to be displayed.

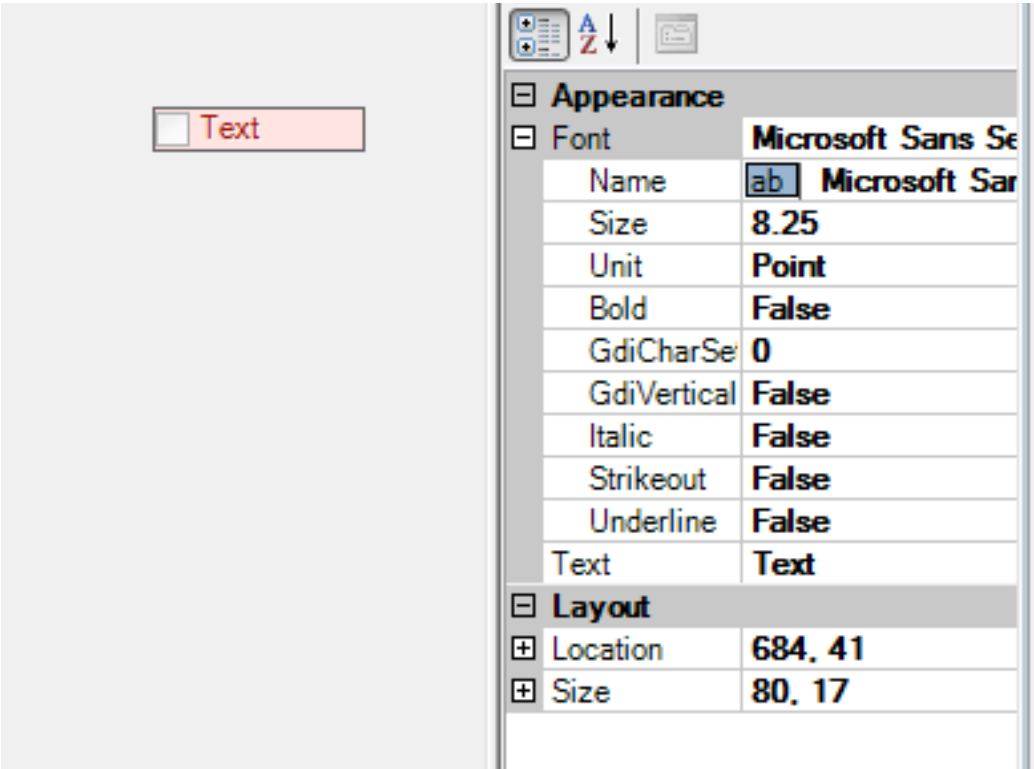
Layout: Defines the location and size of the text box and associated text name.

Operation: This control is used for displaying the text entered in the Text field.

Check Box

Icon: 

Control Panel Element and available settings:



Appearance: Defines the font and text settings and text name to be displayed.

Layout: Defines the location and size of the text box and associated text name.

Operation: This control toggles between the checked and not checked state when clicked.

Radio Button

Icon: 

Control Panel element and available settings:

default Selection

Item 0 (10)

Item 1 (20)

Item 2 (30)

Appearance

ShowValueByRadioButton

True

ShowValueNameByRadioButton

True

Caption

ShowSignalName

True

ShowSignalUnit

False

Data

radioButtons

RadioButtonInfo[]-Array

[0]

BMS.NIControlLibrary.RadioButtonsControl+RadioButtonInfo

[1]

BMS.NIControlLibrary.RadioButtonsControl+RadioButtonInfo

[2]

BMS.NIControlLibrary.RadioButtonsControl+RadioButtonInfo

Default Radio Button

NameDefRadioButton

default Selection

ShowDefRadioButton

True

ShowValueDefRadioButton

False

Layout

Location

165, 56

Size

150, 218


Caption:

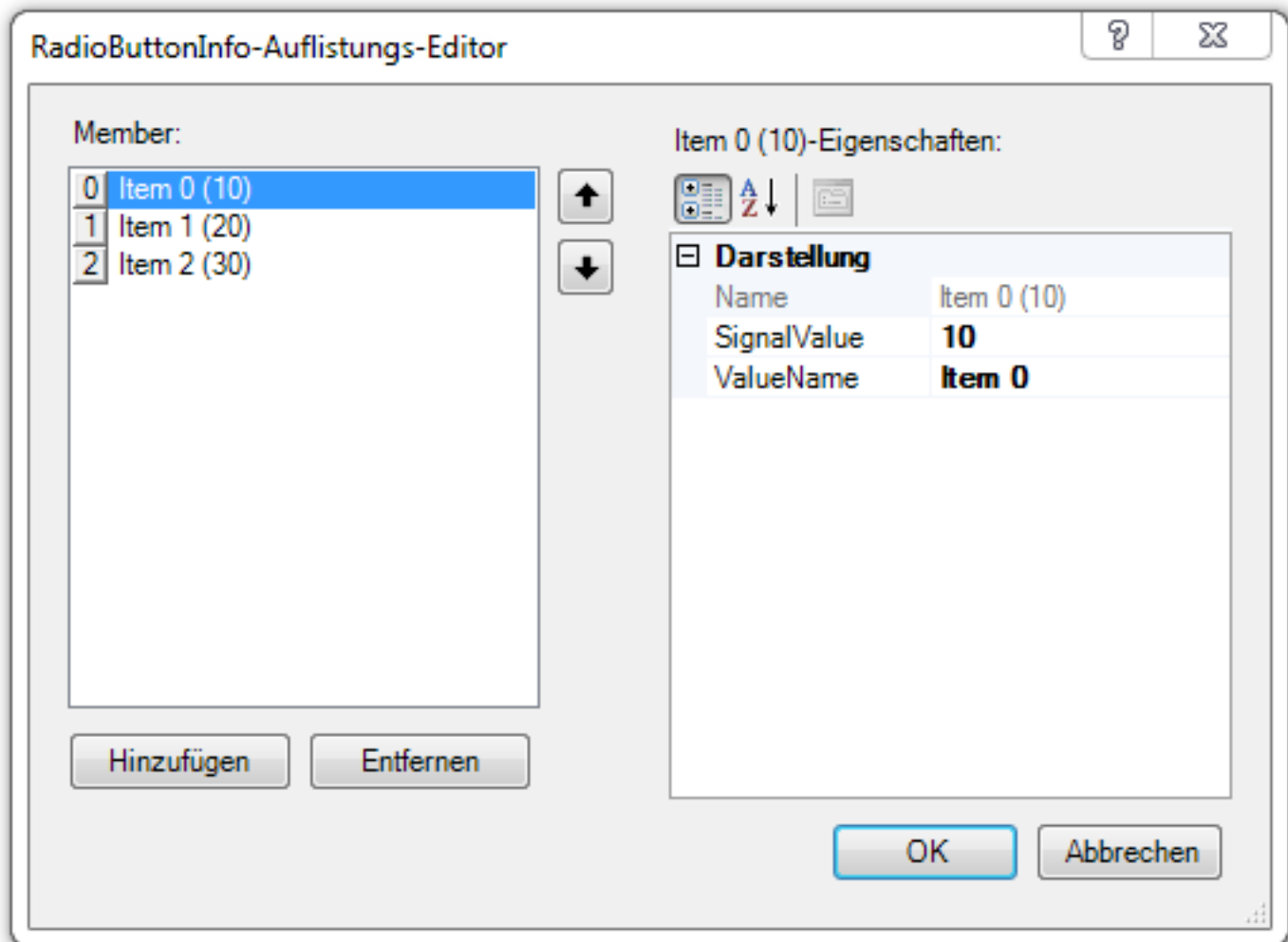
- ShowSignalName: When **True**: the name of the signal mapped to this control is displayed, when **False**: signal name is not displayed
- ShowSignalUnit: When **True**: the unit of the signal mapped to this control is displayed, when **False**: units are not displayed

Appearance:

- ShowValueByRadioButton: When **True**, the control lists the radioButtons items including their name, when **False**, the name does not appear in the radioButtons list. It is possible to have both the name and value in the radioButtons list.
- ShowValueNameByRadioButton: When **True**, the control lists the radioButtons items including their value, when **False**, the value does not appear in the radioButtons list. It is possible to have both the name and value in the radioButtons list.

Data:

- radioButtons: This is a list of **RadioButton** items. The list will appear as shown in the picture above. When this item is selected and the  icon is pressed, the following dialog appears.



This dialog is used to enter the names and values of each item in the list. By pressing **Add** (Hinzufügen), a new item is added. By pressing **Remove** (Entfernen), the selected item is removed from the list. The **ValueName** and **SignalValue** items are entered directly in the matching field on the right side of the dialog.

DefaultRadioButton:

- NameDefRadioButton: This defines the name of the default radio button (active at start).
- ShowDefRadioButton: When **True**, the default radio button is displayed, when **False**, the default radio button is not displayed.
- ShowValueDefRadioButton: When **True**, the default radio button value is displayed, when **False**, the default radio button value is not displayed.

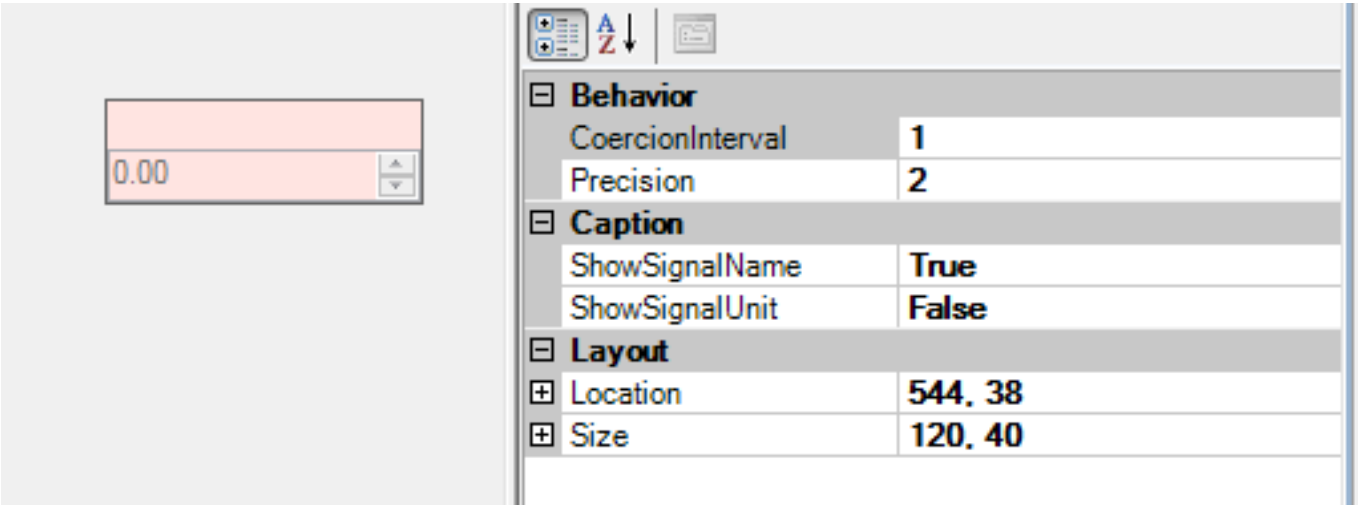
Layout: Defines the location and size of the radio buttons.

Operation: This control can be used to select one of the **RadioButtons** to be active. Only one **RadioButton** can be active at a given time. If a non-active **RadioButton** is activated, the previously active **RadioButton** is automatically de-activated.

Numeric Edit

Icon:

Control Panel element and available settings:



Caption:

- ShowSignalName: When **True**: the name of the signal mapped to this control is displayed, when **False**: signal name is not displayed
- ShowSignalUnit: When **True**: the unit of the signal mapped to this control is displayed, when **False**: units are not displayed

Layout: Defines the location and size of the display.

CoercionInterval: The interval used for coercing the value of the control (i.e. with what increments can the control value be set)

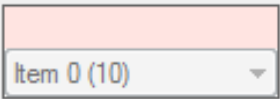
Precision: Defines how many decimal places are available for the user to set.

Operation: This control can be used to set the value of the signal mapped to it. The user can set the value by directly entering a value into the display or by using the up/down arrows to increment the new value.

Combo Box

Icon: 

Control Panel element and available settings:



Appearance	
DropDown	True
ShowValueByItem	True
ShowValueNameByItem	True
Caption	
ShowSignalName	True
ShowSignalUnit	False
Data	
ComboBoxItems	ItemInfo[]-Array
[0]	BMS.NIControlLibrary.ComboBoxControl+ItemInfo
[1]	BMS.NIControlLibrary.ComboBoxControl+ItemInfo
[2]	BMS.NIControlLibrary.ComboBoxControl+ItemInfo
Layout	
Location	195, 128
Size	120, 41


Caption:

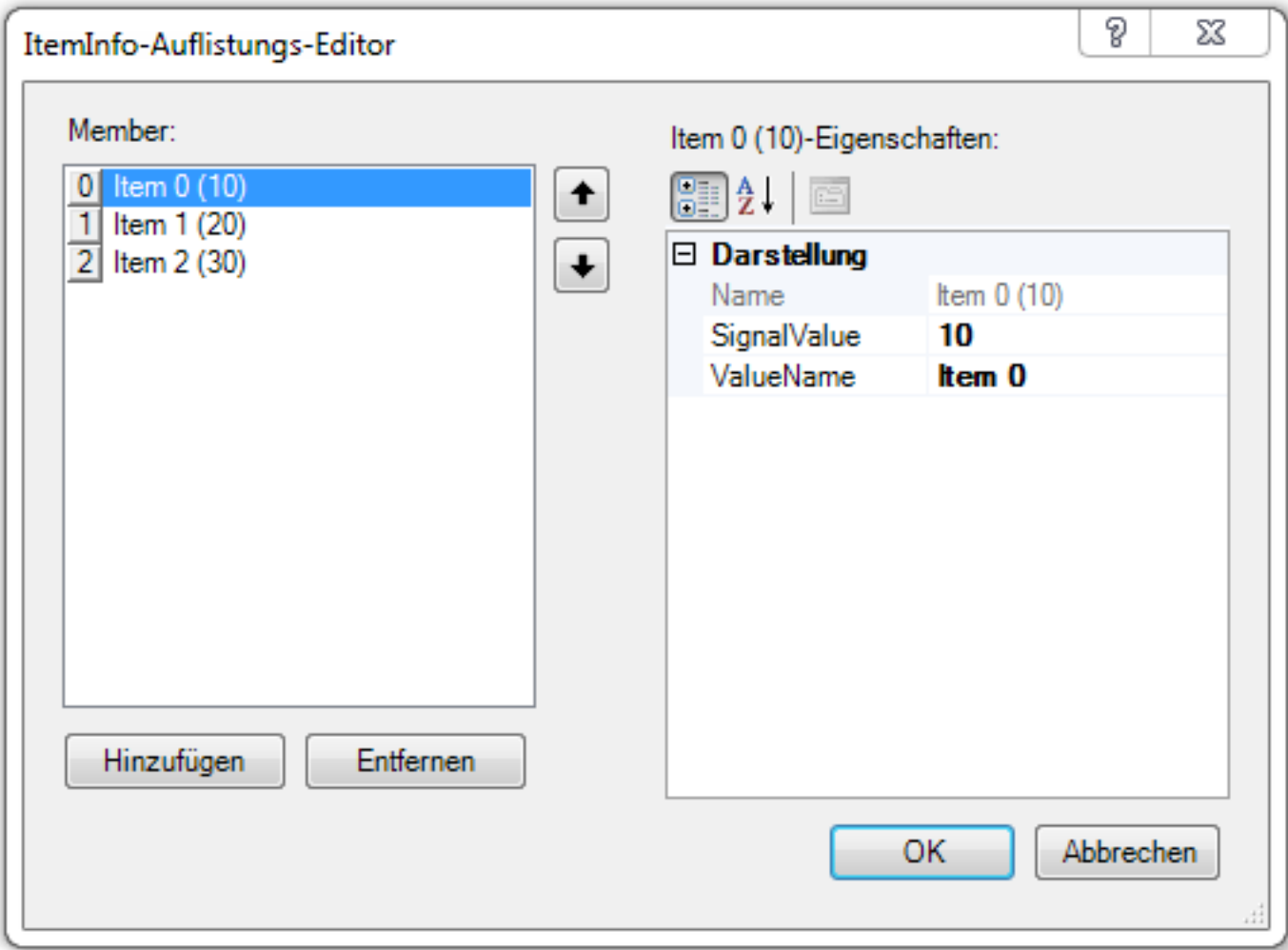
- ShowSignalName: When **True**: the name of the signal mapped to this control is displayed, when **False**: signal name is not displayed
- ShowSignalUnit: When **True**: the unit of the signal mapped to this control is displayed, when **False**: units are not displayed

Appearance:

- DropDown: When **True**, the control appears as a drop down list, when **False**, it appears as a text box (use arrow keys to move up/down the **ComboBoxItems** list)
- ShowValueByItem: When **True**, the control lists the **ComboBoxItems** including their name, when **False**, the name does not appear in the **ComboBoxItems** list. It is possible to have both the name and value in the **ComboBoxItems** list.
- ShowValueNameByItem: When **True**, the control lists the **ComboBoxItems** including their value, when **False**, the value does not appear in the **ComboBoxItems** list. It is possible to have both the name and value in the **ComboBoxItems** list.

Data:

- ComboBoxItems: This is a list of the items to appear in the list. When this item is selected and the  icon is pressed, the following dialog appears



This dialog is used to enter the names and values of each item in the list. By pressing **Add**, a new item is added. By pressing **Remove**, the selected item is removed from the list. The **ValueName** and **SignalValue** items are entered directly in the matching field on the right side of the dialog.

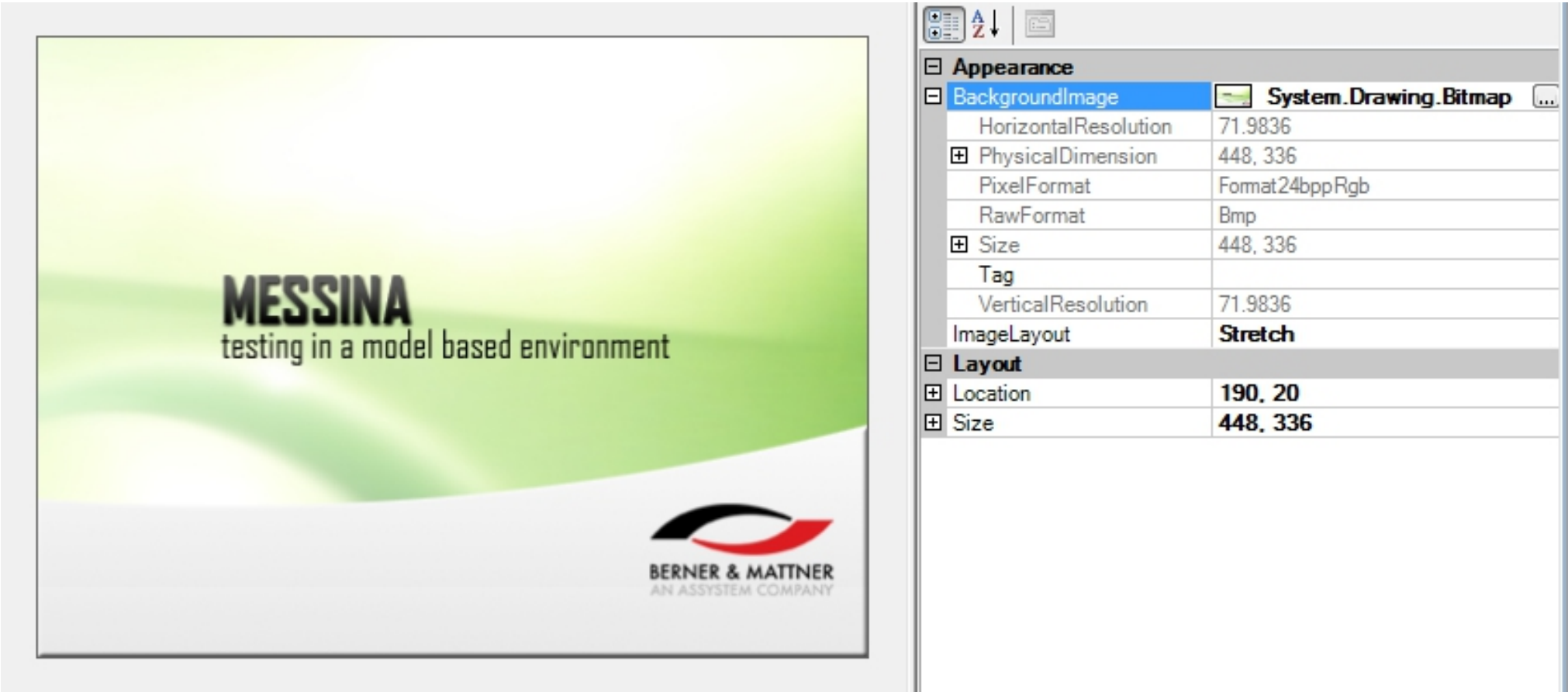
Layout: Defines the location and size of the box.

Operation: This control can be used to set the value of the signal mapped to it to a value available in the list.

Image

Icon: 

Control Panel element and available settings:



This control is used as a container for displaying an image. The settings displayed above are used to define the type, size, format, and location of the container and its associated image.

Graph Visualizations

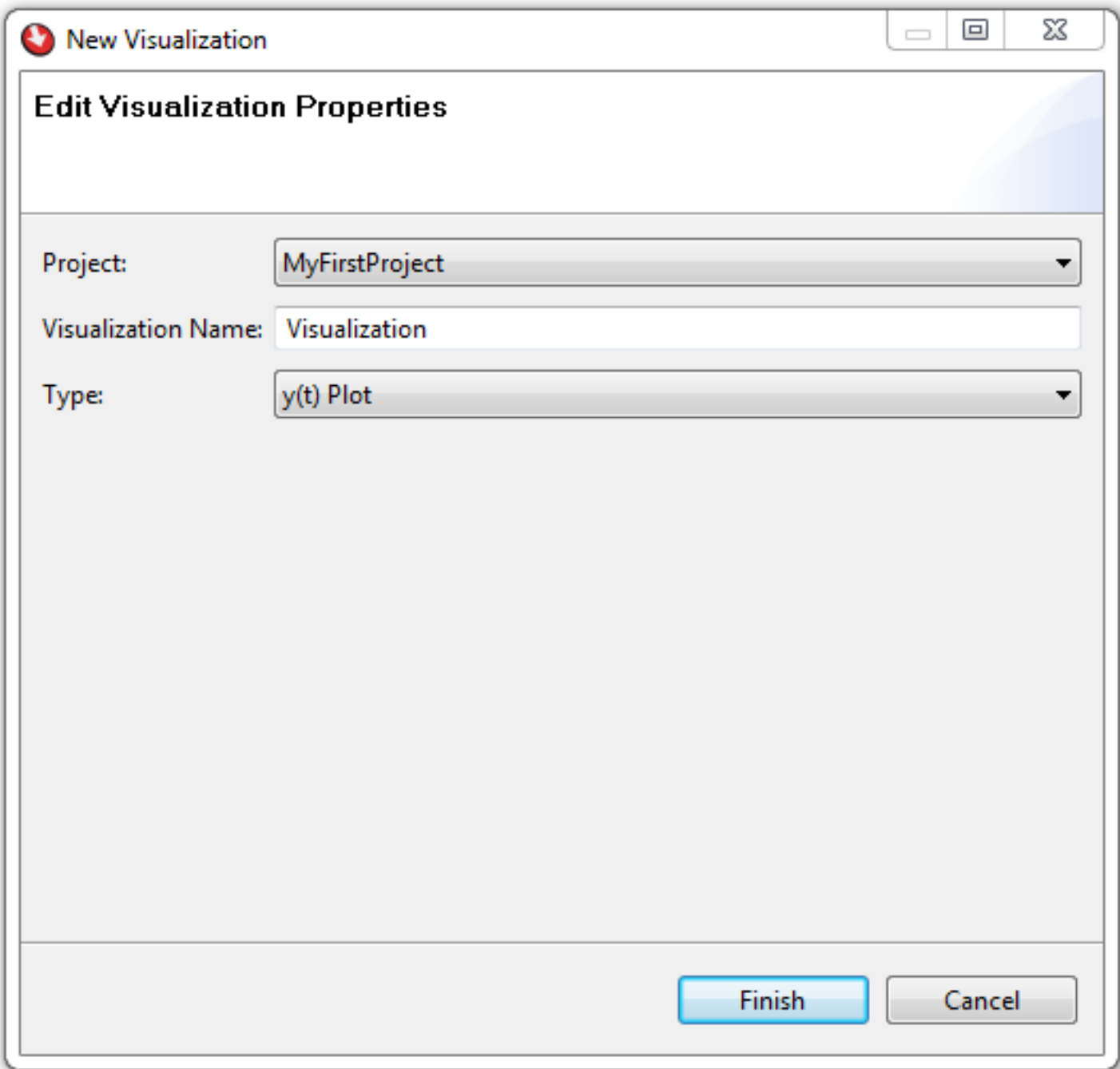
The MESSINA **y(t) Plot** (graph) visualization offers the ability to graphically trace a signal over time. Signals can be connected to the graph by drag-and-drop from the **Signalpool**. The sections below describe the header icons and the graph display.

Adding a y(t) Plot Visualization

MESSINA **Visualizations** are created and edited using the **Executor** perspective which can be selected using the tabs at the top right of the main window. From the main menu, open the dialog **File** → **New** → **Visualization**. It is also possible to open the **File** → **New** → **Other** dialog and select the **Visualization wizard** in the MESSINA folder.

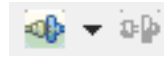
Alternatively, the **Visualization Wizard** can be called using the **Context Menu** → **New Wizards** → **Visualization**.


The following dialog box appears when the **Visualization Wizard** is called:



Select the ***y(t) Plot*** item from the **Type** pull-down list and press **Finish**. The empty visualization is displayed.

Header Icons

 **Connect visualization** (current state is disconnected): Pressing the active icon will connect the current visualization with the default target. Clicking on the down arrow between the icons allows any available target to be selected.

 **Disconnect visualization** (current state is connected): Pressing the active icon will disconnect the current visualization. Clicking on the down arrow between the icons allows any available target to be selected.

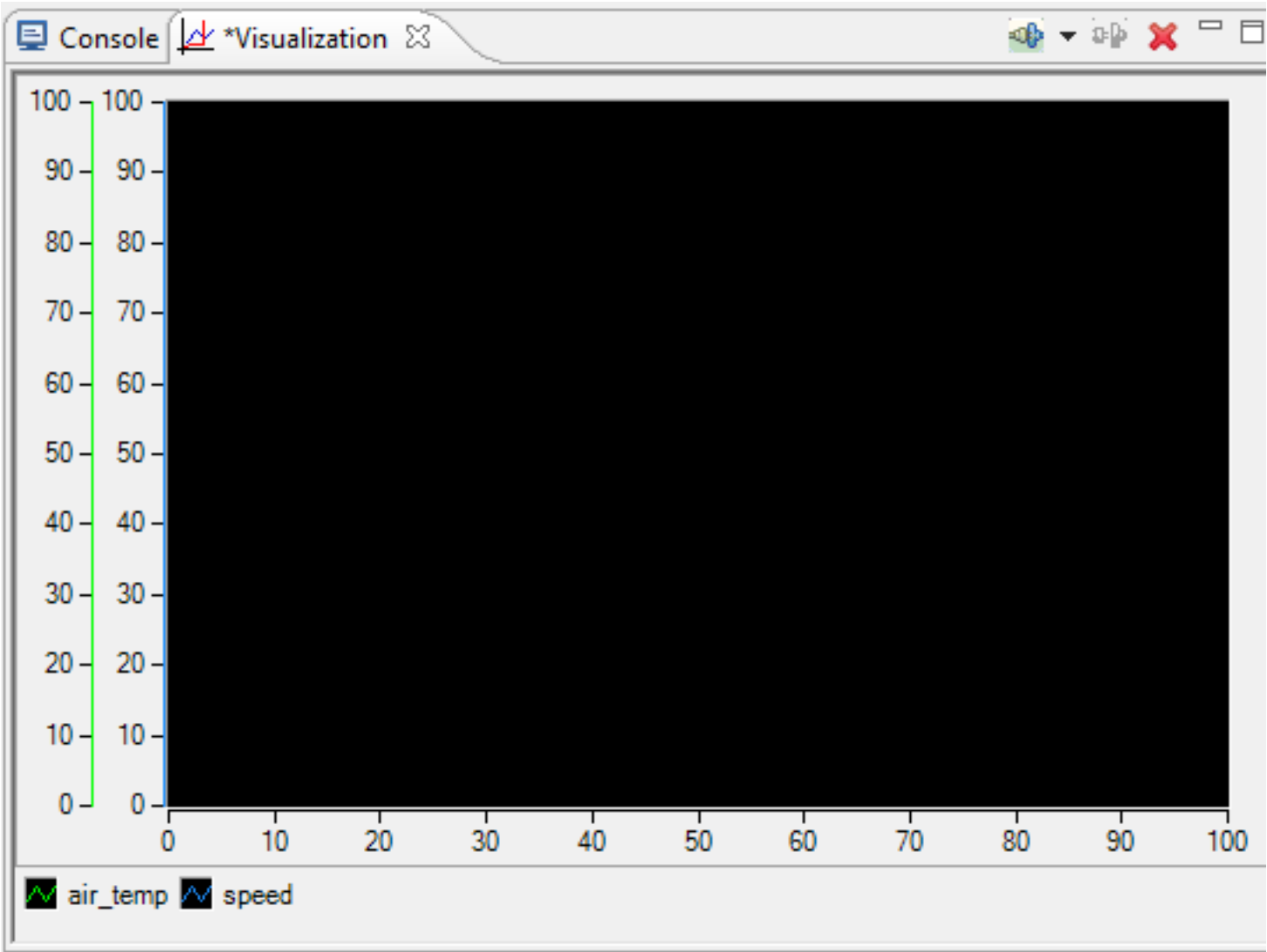
 **Remove all signals**: Pressing this icon will remove all signals from the graph display.

Control Properties

The display will first have a pink background to indicate that the control has not been connected to a signal.



Signals can be attached to the graph element by drag-and-drop. An example of a ***y(t) Plot*** visualization with the ***air_temp*** and ***speed*** signals attached is shown below.



Set the Scale of a Signal and Line Properties

Note: These options can also be used for the graph element on [Control Panel](#) visualizations.

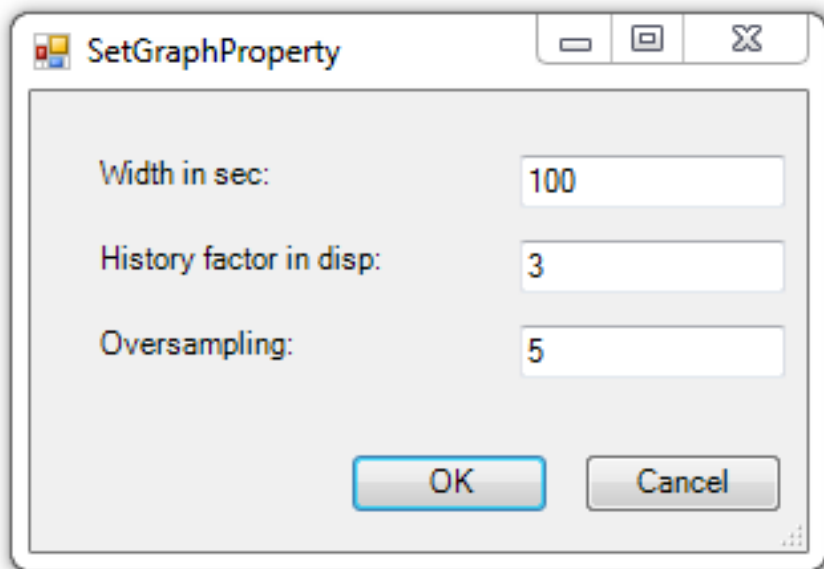
To set the scaling on the graph element, double click on the desired signal in the legend (bottom left). The following dialog appears.

The screenshot shows a dialog box titled 'SetSignalProperty'. It has several input fields and a dropdown menu. The 'Name' field contains 'speed'. The 'Type' field contains 'double'. The 'Old Value' field contains '0'. The 'New Value' field contains '0'. The 'Minimum' field contains '0'. The 'Maximum' field contains '100'. The 'Display Axis' field has a checked checkbox and a 'Line Color' button. The 'Line Style' field is a dropdown menu showing 'Solid'. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

Enter the desired line properties, limits, and new value and press **OK**. The scale of the graph and the signal line properties are updated.

Set Graph Properties

The properties for a graph can be set by double clicking within the graph element. The following dialog is called:



Make the settings as required and press **OK**. The graph is updated with the new settings.

Panning and Zooming

Panning:

- Press and hold the **CTRL** key, then press and hold the left mouse button. Moving left/right and up/down is done by moving the mouse
- Press and hold the **CTRL** key, then use the arrow keys to scroll left/right or up/down

The scale remains unchanged.

Zooming:

- Press and hold the **Shift** key and turn the mouse wheel
- Press and hold the **Shift** key and use the up/down arrow keys
- Press and hold the **Shift** key, then press and hold the left mouse button to mark an area, then release the left mouse button
- Double click on the desired signal name in the legend (bottom left), then set the min/max values as desired.

The scale can be set back by double clicking on the desired signal name in the legend and setting the min/max values to the normal scale. Each signal must be set separately.

Signal Capture

The MESSINA **Signal Capture** offers the ability to start and stop the Visualization Logger out of the test case.

To use this function it is necessary to create the corresponding visualization first. The **Signal Capture** works with [Signal Logger](#) and [MDF Logger](#).

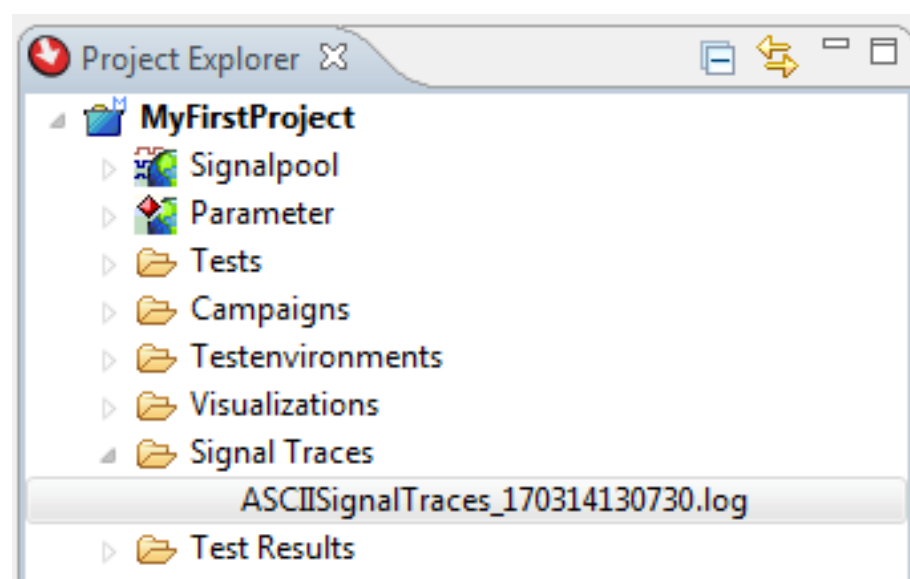
The next step is to create a [test case](#). The **Signal Capture** commands are part of the run method. The command `startCapture` is used to start the logger. The command `stopCapture` is used to stop the logger.

```
/**
 * contains the test sequence
 */
public int run() throws IOException, TestFailedException,
    InterruptedException {

    // insert the test sequence here
    ASSERT(startCapture("Visualization", null));
    for (int i = 0; i <= 10; i++) {
        sleep(100);
        signal_1.setValue(i);
    }
    stopCapture("Visualization");
    return 0; // 0 means PASSED
}
```

This example contains a **Signal Logger** called `visualization`. The logger and the log file name must be set in the `startCapture` command. The log file name can either be set to a certain filename or to `null`. If it is set to `null` MESSINA will use the internal logger name including a timestamp as file name.

Executing the test case will activate and deactivate the **Signal Logger** automatically.



Log files are located in the ***Signal Traces*** folder of the ***Project Explorer***.

See also: [***Traces and Logging***](#)

Table View Visualizations

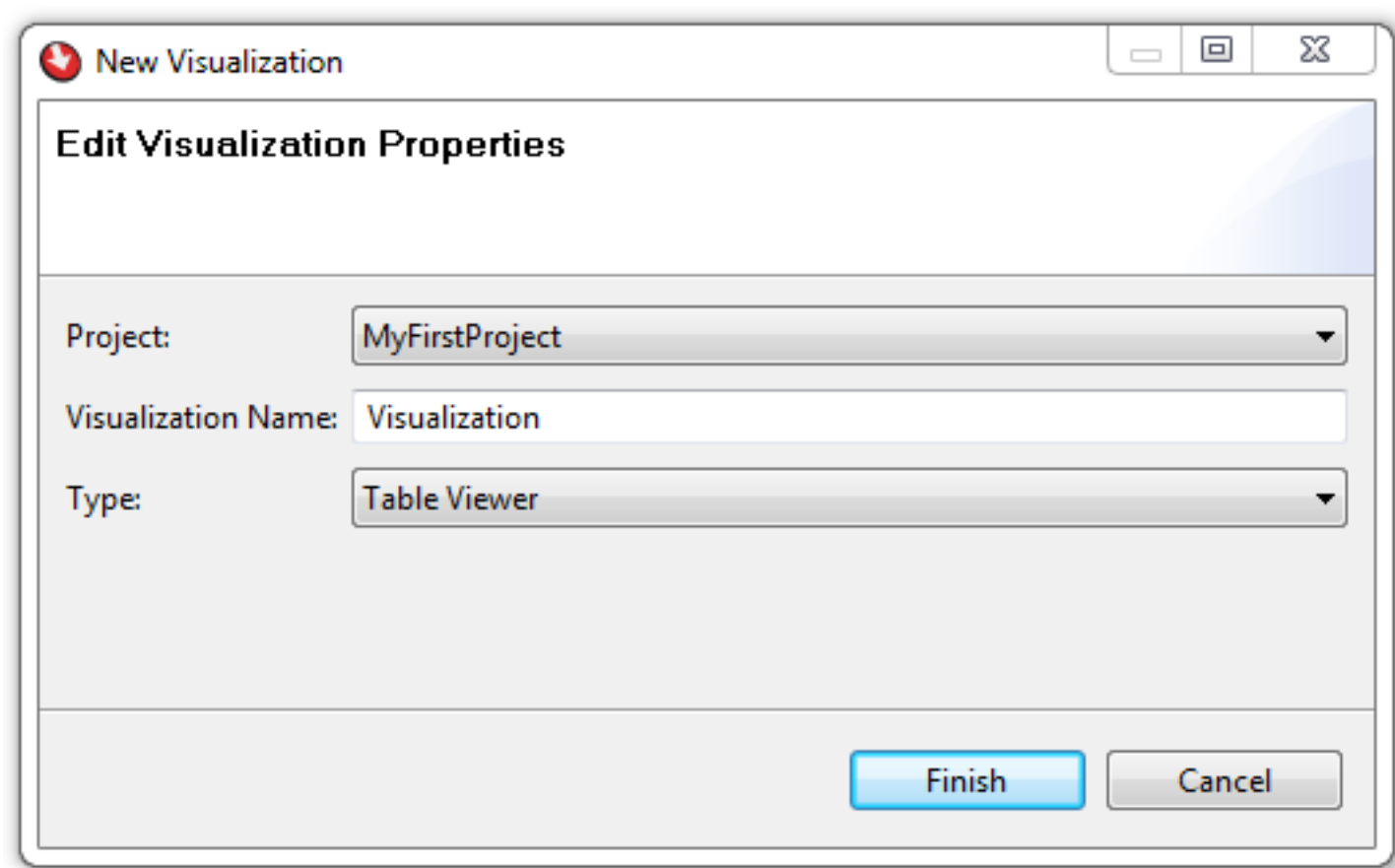
The MESSINA **Table Viewer** visualization offers the ability to list all signals in a table display. Signals can be connected to the table by drag-and-drop from the signalpool. The sections below describe the header icons and the table viewer display.

Adding a Table View Visualization

MESSINA **Visualizations** are created and edited using the **Executor** perspective which can be selected using the tabs at the top right of the main window. From the main menu, open the dialog **File** → **New** → **Visualization**. It is also possible to open the **File** → **New** → **Other** dialog and select the **Visualization wizard** in the MESSINA folder.

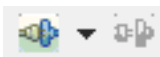
Alternatively, the **Visualization Wizard** can be called using the **Context Menu** → **New Wizards** → **Visualization**.

The following dialog box appears when the **Visualization Wizard** is called:






Select the **Table Viewer** item from the **Type** pull-down list and press **Finish**. The empty visualization is displayed.


Header Icons



Connect visualization (current state is disconnected): Pressing the active icon will connect

the current visualization with the default target. Clicking on the down arrow between the icons allows any available target to be selected.

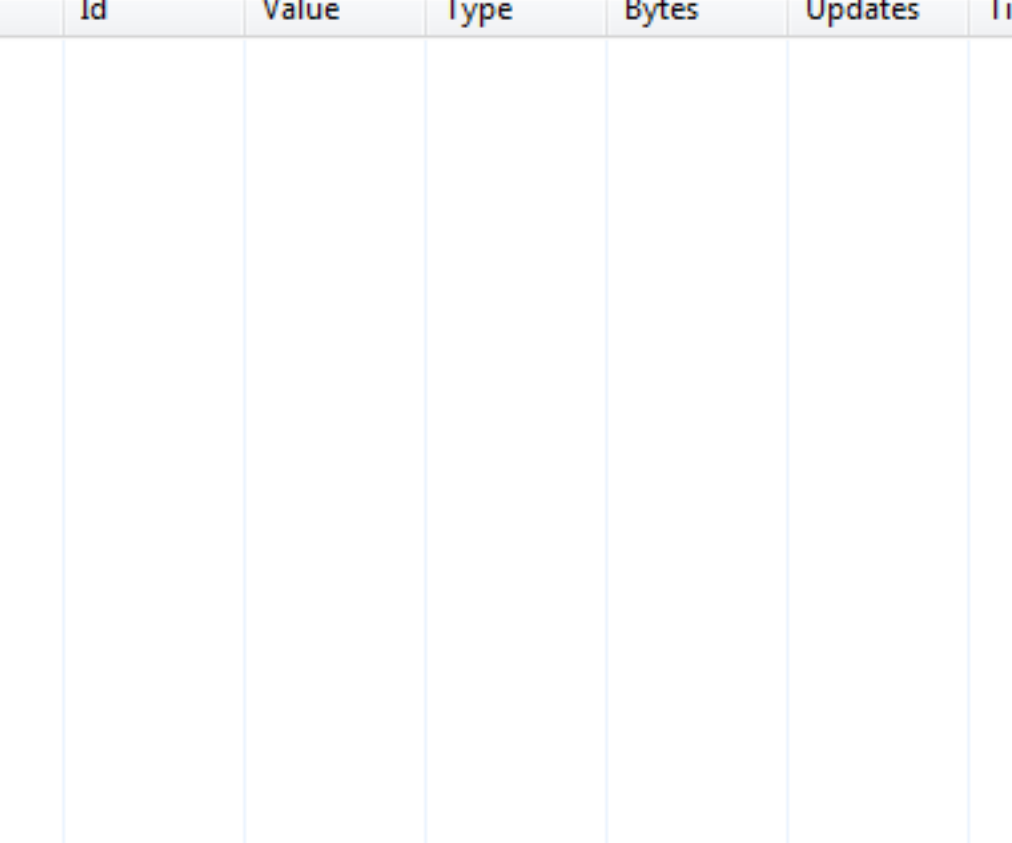
   **Disconnect visualization** (current state is connected): Pressing the active icon will disconnect the current visualization. Clicking on the down arrow between the icons allows any available target to be selected.

 **Remove all signals:** Pressing this icon will remove all signals from the table display

 **Reset Counter:** Pressing this icon will reset the "Updates" count to 0

Control Properties

The newly created display will be empty indicating that the control has not been connected to a signal.







The screenshot shows the Visual Studio Code interface. The 'Console' tab is selected, and the 'Visualization' tab is also visible. The 'Visualization' tab displays a table with the following columns: Name, Id, Value, Type, Bytes, Updates, and TimeSta... The table is currently empty.

Name	Id	Value	Type	Bytes	Updates	TimeSta...
------	----	-------	------	-------	---------	------------

Signals can be attached to each control element by drag-and-drop. An example of a ***Table Viewer*** visualization with several signals attached is shown below.

Console

*Visualization

Name	Id	Value	Type	Bytes	Updates	TimeSta...
 sensor...	1	0.0	double	00 00 00 ...	0	0
 speed	2	0.0	double	00 00 00 ...	0	0
 air_te...	3	0.0	double	00 00 00 ...	0	0
 ice_w...	4	false	bool	00	0	0

Columns:

- **ID:** The corresponding signal ID. This is the same ID as in the signalpool.
- **Value:** The current value of the signal.
- **Type:** The signal data type.
- **Bytes:** A byte array display of the signal value.
- **Updates:** A count of how often the signal has been updated.
- **Timestamp:** Time stamp associated with the last update.

Modifying a Table View Signal Value

The value of a given signal can be modified directly by double clicking on the signal name. The following dialog is called:

Signal Value

Set New Signal Value

Name: speed

Old Value: 0.0

New Value: 20

OK

Cancel

Modify the value by entering the new value and pressing **OK**. The **Table View** is displayed with the new signal value.

Test Results

Test Reports

One requirement of any test system is always a **Test Report**. Somebody somewhere will always want something printable that can be reviewed or archived if required. MESSINA handles this with a built-in report generator.

Test Reports

In order to view a report, a style sheet must be assigned. This must be set in the **Main Menu** → **Window** → **Preferences** section. See the **Preferences** section of this documentation for a detailed description of how to set the **Preferences** required for creating and viewing a **Test Report**.

Test Reports are created from **Test Results** that have been saved. Saved **Test Results** are listed in the **Test Results** folder as displayed in the [Project Explorer](#) view. To create a **Test Report**, double click on an existing result. A report will be generated automatically. The following test report was generated by running the TempComp example 5 times.

Note: Only a portion of the report is displayed here. No logging information is displayed.

Test Report

File generated by Tutotorial_Autor at 01.01.17 12:00:00

Project: MyFirstProject

Configuration: XiL

	TempComp_Windows	(Matlab Simulink)	4.1.0.10986 (1.229)
--	------------------	-------------------	---------------------

Target:

	Windows (local)	(Windows)	4.1.0.10986 / Windows 6.1 (1.0) 1	VM	4.1.0.10986
--	-----------------	-----------	-----------------------------------	----	-------------

Comments:

Statistics

Successful	1	50,00 %
Failures	1	50,00 %
Errors	0	0,00 %
Inconclusive	0	0,00 %
Tests	2	

Successful Tests

Test name	Test group	Time of test	Duration	Comments
MyTestCase.java (<default>)	/MyTestCampaign	30.06.2011 21:11:55	0:00:41.579	

Go to top

Failed tests

Test name	Test group	Time of test	Duration	Comments
MyTestCase.java (<default>)	/MyTestCampaign	30.06.2011 21:12:37	0:00:57.095	Assertion failed: Temperature not reached (MyTestCase.java:14)

Go to top

Error tests
No error tests.
Go to top

Test Results Info
1 MyTestCampaign

Result:	FAILED
Start:	30.06.2011 21:11:55
Duration:	0:01:38.674

Comments:	Total 2; Successful 1
-----------	-----------------------

TestResultsInfo

[Go to top]

1.1 MyTestCase.java (<default>)

Result:	PASSED
ReturnValue:	0
Start:	30.06.2011 21:11:55
Duration:	0:00:41.579
Comments:	

MyTestCampaign	Test Log
----------------	----------

Parameters:

<u>Name</u>	<u>Type</u>	<u>Value</u>	<u>Path</u>
air_temp	int	5	/
timeout	int	30000	/
speed	int	130	/

[Go to top]

1.2 MyTestCase.java (<default>)

Result:	FAILED
ReturnValue:	-2000
Start:	30.06.2011 21:12:37
Duration:	0:00:57.095
Comments:	Assertion failed: Temperature not reached (MyTestCase.java:14)

MyTestCampaign	Test Log
----------------	----------

Parameters:

<u>Name</u>	<u>Type</u>	<u>Value</u>	<u>Path</u>
air_temp	int	1	/
timeout	int	30000	/
speed	int	130	/

[Go to top]

OS	x86-Windows XP-5.1
----	--------------------

Go to top

The format of the report is standardized so it will always have the same layout which is based on the style sheet.

Note: It is possible to create a user defined report. An XSLT script file (style sheet) must be created/available and selected in the ***Preferences***.

Signal Traces and Data Logging

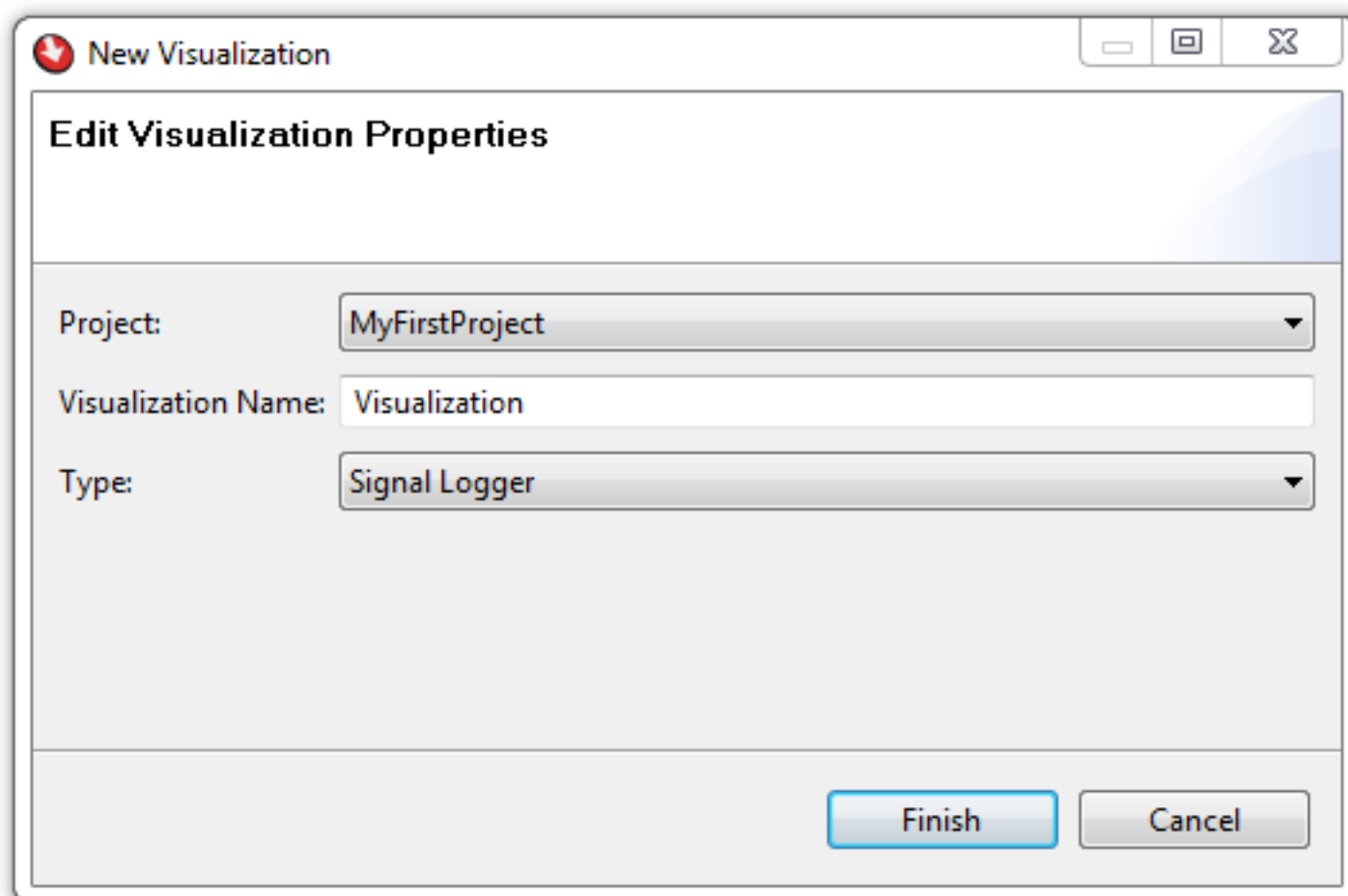
The MESSINA **Logger** visualizations offer the ability to log and store a list of all/any required signals available in the **Signalpool**. Signals can be connected to the data logger visualizations by drag-and-drop from the **Signalpool**. The sections below describe the header icons and the logger visualization displays.

Adding a Logger Visualization

MESSINA **Visualizations** are created and edited using the **Executor** perspective which can be selected using the tabs at the top right of the main window. From the main menu, open the dialog **File** → **New** → **Visualization**. It is also possible to open the **File** → **New** → **Other** dialog and select the **Visualization wizard** in the MESSINA folder.

Alternatively, the **Visualization Wizard** can be called using the **Context Menu** → **New Wizards** → **Visualization**.

The following dialog box appears when the **Visualization Wizard** is called:




Select the **Signal Logger** or **MDF Logger** item from the **Type** pull-down list and press **Finish**. The empty visualization is displayed.

Header Icons

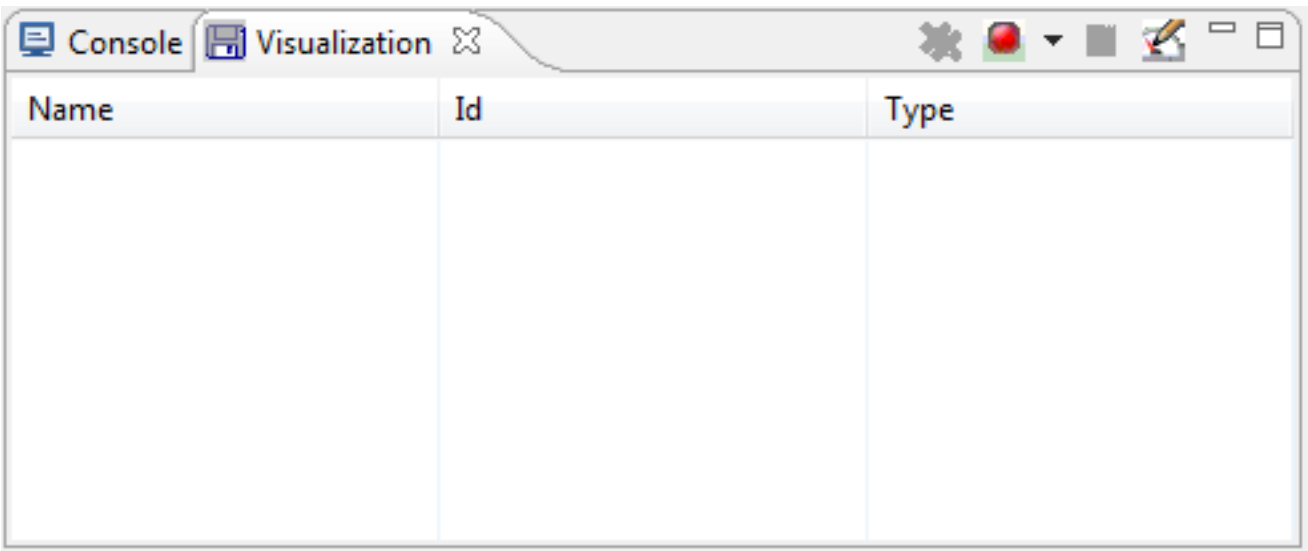
 **Stop Logging** (logging is currently ON)

 **Start Logging** (logging is currently OFF)

 **Remove all signals:** Pressing this icon will remove all signals from the table display

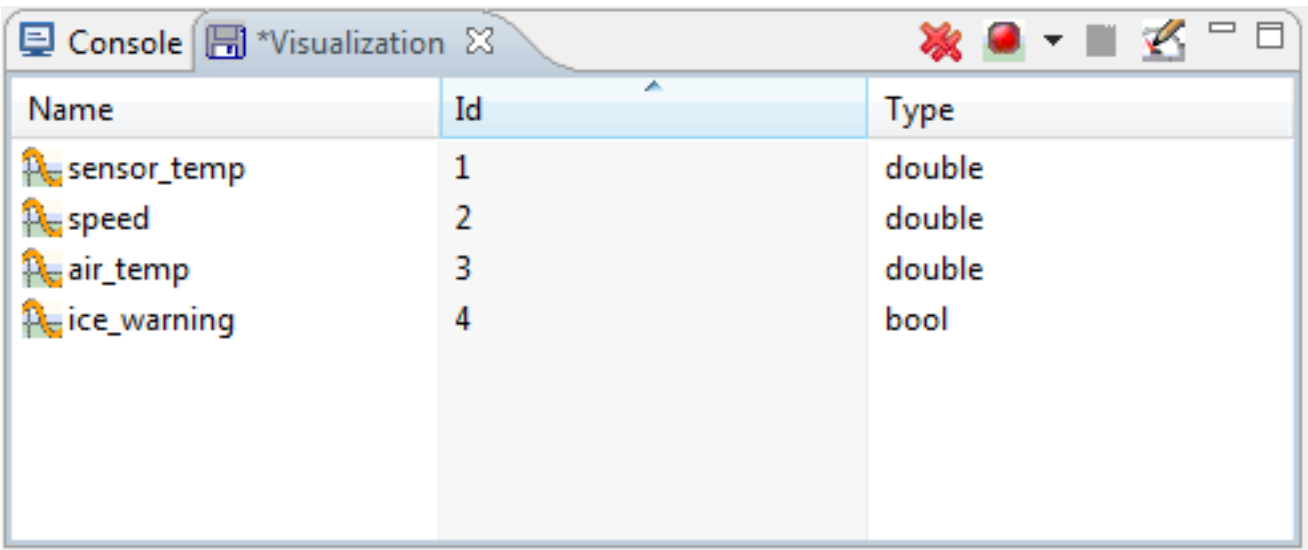
Control Properties





The newly created display will be empty indicating that the control has not been connected to a signal.



Name	Id	Type

Signals can be attached to each control element by drag-and-drop. An example of a **Logger** visualization with several signals attached is shown below.





Name	Id	Type
 sensor_temp	1	double
 speed	2	double
 air_temp	3	double
 ice_warning	4	bool

Columns:

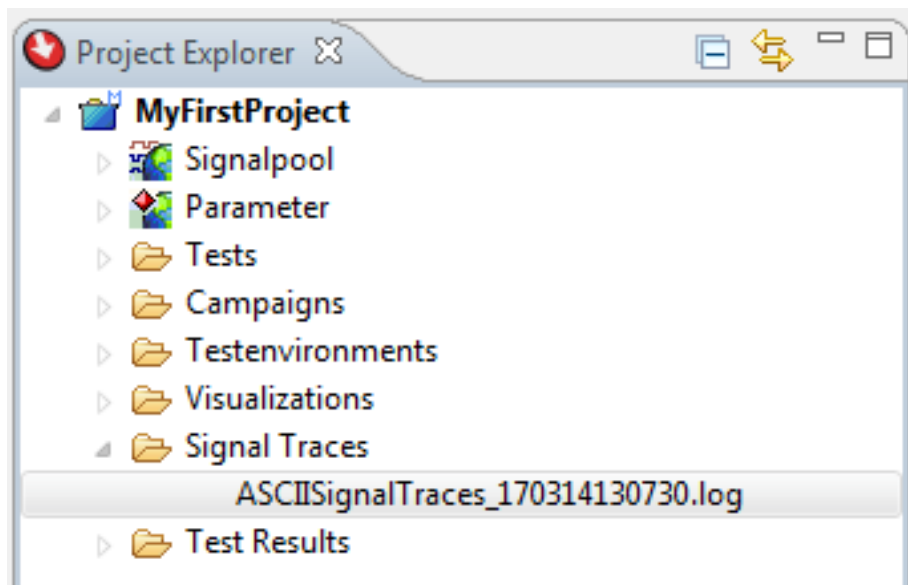
- **Name:** The corresponding signal name. This is the same name as in the signalpool.
- **ID:** The corresponding signal ID. This is the same ID as in the signalpool.
- **Type:** The signal data type.

Logging Signals

To start logging a selected signal(s), the target must be connected and the configuration must be running. At least one signal must be added to the logger visualization. Pressing the start logging icon

 will start the logging process manually. It will continue until the stop logging icon  is pressed. Additional to this the signal logging can be started automatically out of test cases. To learn more about it, see: [Signal Capture](#)

Once the logging process is stopped, the data is saved to the file. An example of the [Project Explorer](#) view after logging is completed is shown below:



Double clicking a Signal Trace item will call the standard viewer for that type of log file. The standard viewer can be set in the **Preferences**.

ASCII Signal Trace Files

The **ASCIISignalTraces** files are stored in a standard text format that can be viewed with any text editor program (e.g. Notepad.exe). An example of an ASCII signal trace file is show below (note: the <tab> was added here for clarity because the items are separated by a tab):

```
1 <tab> 2027701 <tab> 4.0
2 <tab> 2027701 <tab> 103.0
3 <tab> 2027701 <tab> 4.9
3 <tab> 2027901 <tab> 4.8
3 <tab> 2028201 <tab> 4.7
3 <tab> 2028401 <tab> 4.6
3 <tab> 2028701 <tab> 4.5
3 <tab> 2028901 <tab> 4.4
3 <tab> 2029201 <tab> 4.3
3 <tab> 2029401 <tab> 4.2
3 <tab> 2029701 <tab> 4.1
```

The data lines contain the signal ID (index), then a tab, then the time stamp, then another tab, then the value. A data line is created each time the value of the signal changes.

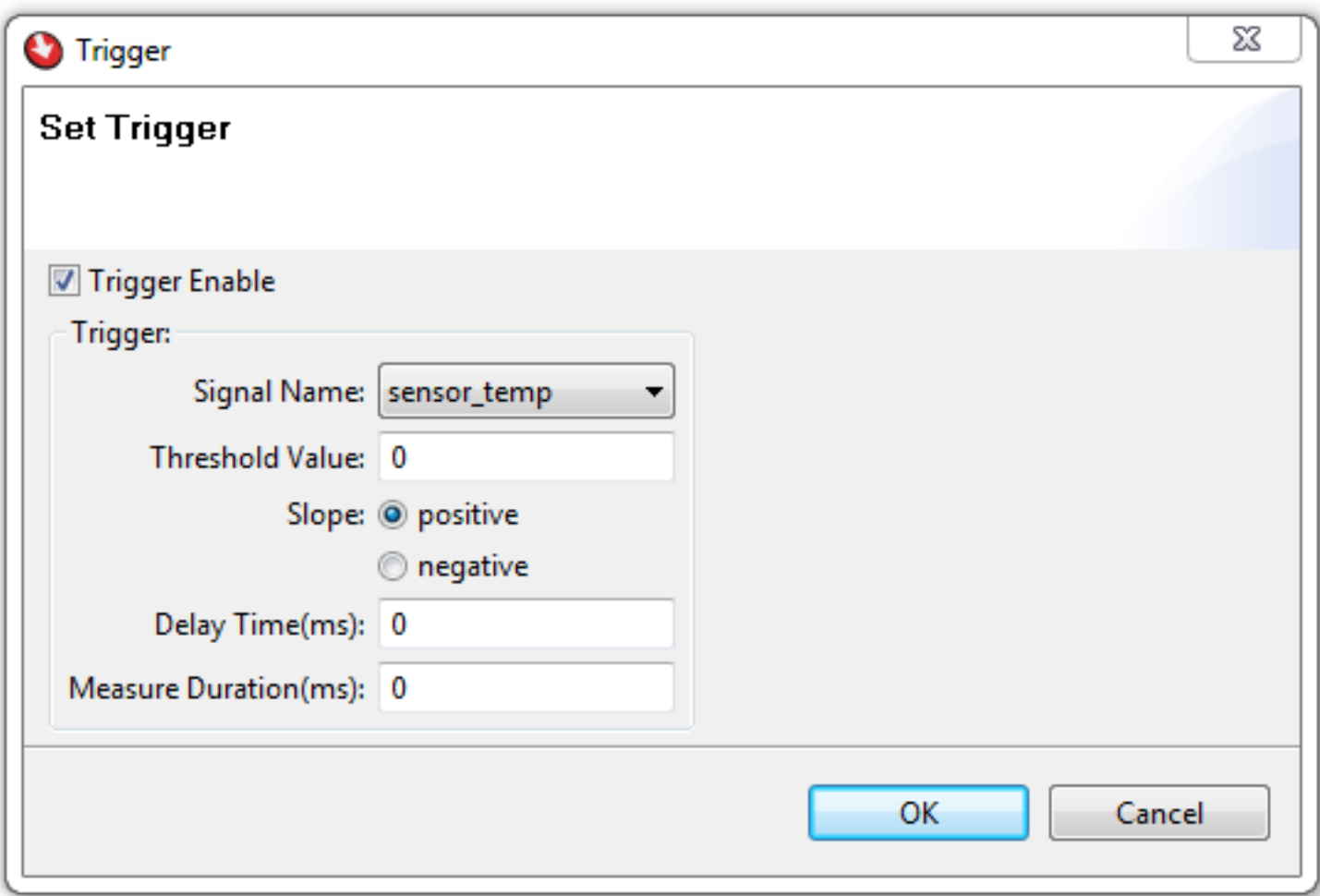
MDF Signal Traces

The **MDFSignalTraces** files are stored in a special (standarized) format and can not be viewed as text. A special viewing tool is required (e.g. canape32.EXE) which is not part of the MESSINA installation. The standard viewer can be set in the **Preferences**.

The MDF visualization offers the possibility to edit a trigger. This can be done by pressing the **Edit**

Trigger icon: 

The following dialog is called:



When the **Trigger Enable** check box is activated, the trigger settings can be modified. The **Signal Name** is selected from a pull-down list of all available signals. The **Threshold value** (which is the trigger value) can be entered directly. The **slope** (direction), **delay time** (how long to wait before triggering) and **measure duration** (the min. time the signal must be at the threshold level before a trigger occurs) can also be set. Pressing **OK** will accept the new trigger settings.

Log View

Refer to the [Log View](#) section of this documentation for a detailed description.

FAQ

Question:

What should I take into account when using MESSINA with Subclipse?

Answer:

Don't use the lock function. This can cause problems with linking and overwriting of sequences.

Non-versioned projects should be added to a repository by using the **Team** → **Share Project...** context menu entry. Since the **Library** project (imported CAN catalogues, models, etc) is not visible in the MESSINA **Project Explorer** you have to use the standard eclipse **Package Explorer** or **Navigator** views in order to synchronize with a subversion repository. These views can be opened via **Window** → **Show View** → **Other...**

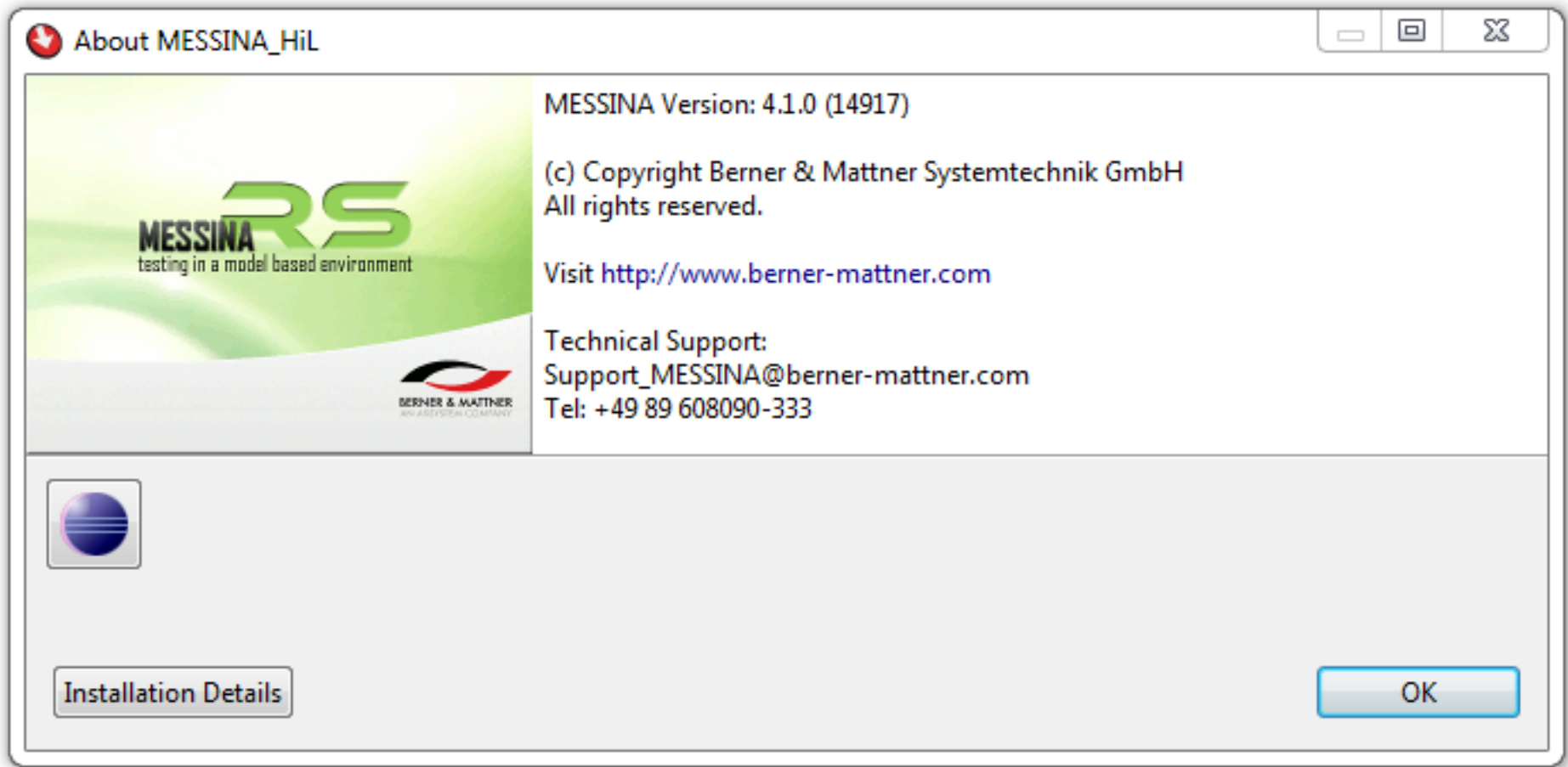
Please refrain from committing the whole MESSINA workspace as it could cause severe complications.

Question:

How can I determine the MESSINA version (e.g. for support enquiries)?

Answer:

Use the **Main Menu** → **Help** → **About MESSINA** option. The main version is displayed. It is also possible to view the **Installation Details** by selecting the option from the dialog.



Question:

How can I reset the perspective to the original format?

Answer:

Use the **Main Menu** → **Window** → **Reset Perspective** option. Only the currently displayed perspective is reset to its default configuration.

Question:

My visualization is not reacting during a running process. What could the problem be?

Answer:

1. Confirm that the visualization being shown is that from the currently active project (it is possible to display visualizations from a project that is not currently active).
2. Confirm that the visualization is connected.
3. Confirm that a process is executing (this is visible in the [Result Manager](#) view)

Question:

How can I generate a test report?

Answer:

The [Test Report](#) section of this documentation describes the process of creating a test report.

Question:

How can I update my current MESSINA version?

Answer:

MESSINA has a built-in mechanism for doing updates.

Question:

Is it possible to rename a project?

Answer:

Renaming is currently not supported.

Question:

When I execute a **Campaign** or **Test Case**, the active display always goes to the Target view. Why does this happen?

Answer:

The target view is actually the **Log View**. This is always activated (gets focus) automatically when a **Test Case** or **Campaign** is executed. The **Log View** displays information about the current process if a logging state is activated. Refer to the [Log View](#) section of this documentation for detailed information.

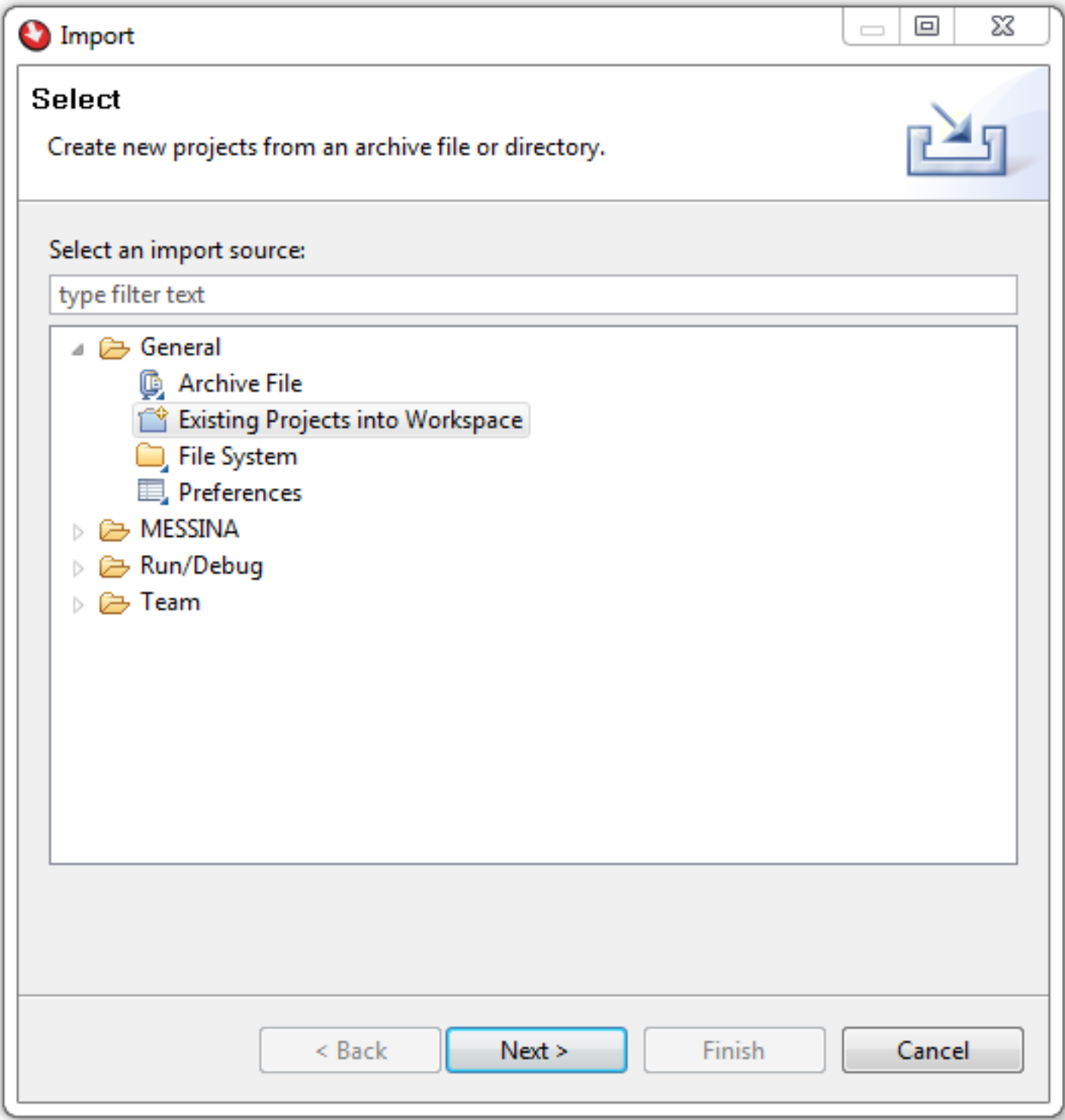
Question:

How can I add an existing project to the current workspace?

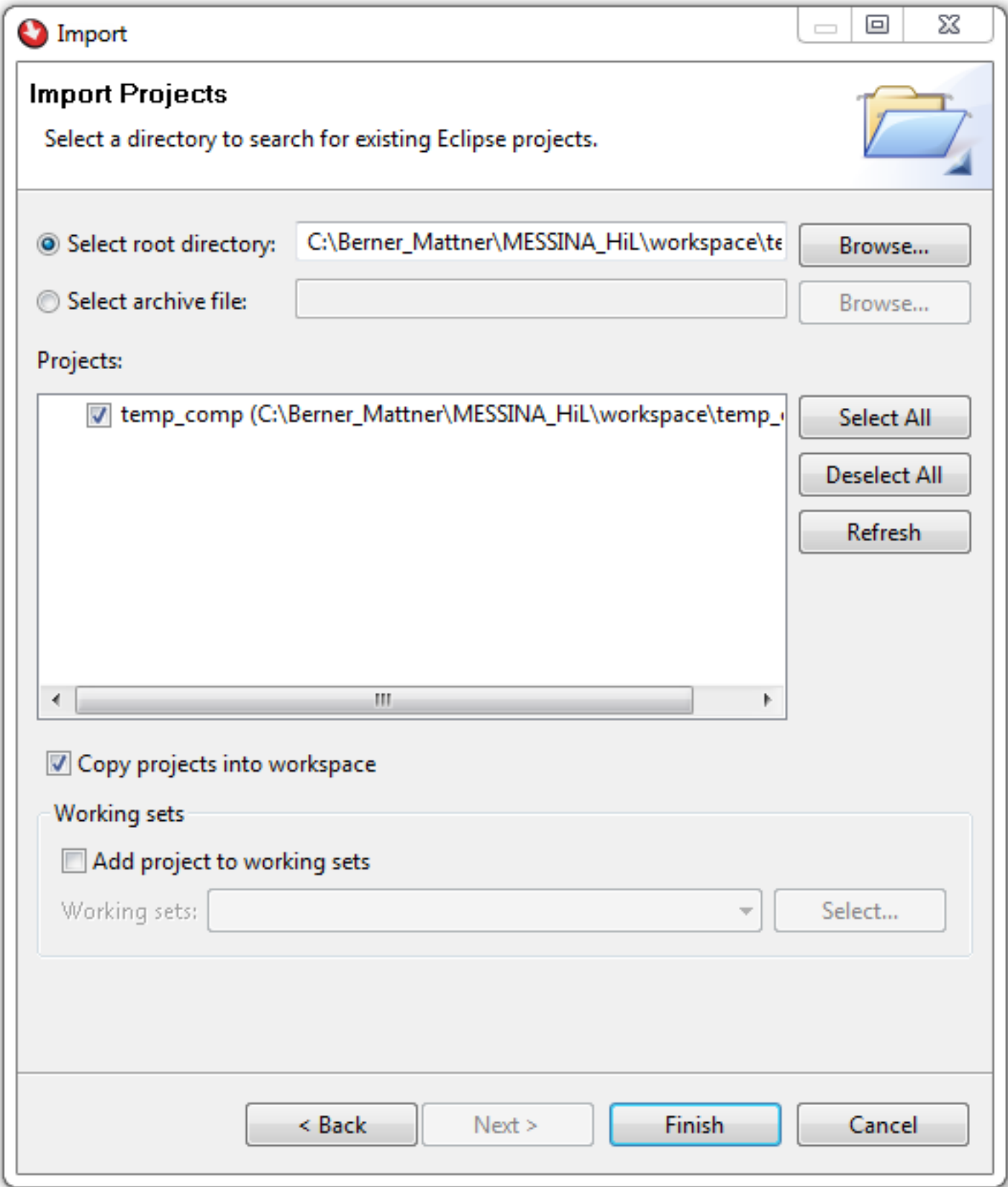
Answer:

Do the following steps in order:

1. Use **Main Menu** → **File** → **Import** and select **General** → **Existing Projects into Workspace** (see picture below)



2. Press the Browse button to navigate to the workspace from which you wish to copy a project. Select the desired project from the list displayed. Make sure the Copy projects into workspace checkbox is active (see picture below)



3. Press Finish