VORTEX
OPENSPLICE

Test Tool User Guide
Release 6.x

Contents

1

Preface

1.1 About The Vortex OpenSplice Tester User Guide

The *OpenSplice Automated Testing and Debugging Tool User Guide* is intended to provide a complete reference on how to configure the tool and use it to test applications generated with the Vortex OpenSplice software.

This *User Guide* is intended to be used after the Vortex OpenSplice software has been installed and configured according to the instructions in the OpenSplice *Getting Started Guide*.

1.2 Intended Audience

This *OpenSplice Automated Testing and Debugging Tool User Guide* is for everyone using the tool (which is usually referred to as the *Tester*) to assist in developing and debugging their DDS applications with Vortex OpenSplice software.

1.3 Organisation

The Introduction provides general information about the Automated Testing and Debugging Tool.

Getting Started describes how to use the Tester's main features.

Next, *some familiarization exercises* show how to perform some typical tasks with step-by-step instructions.

The *Command Reference section* has a complete list of all of the commands available, with fully-detailed descriptions of their use.

The *section on Scripting* describes how to automate repetitive testing procedures with scripts and macros, provides a list of all of the built-in script instructions, and shows how different scripting languages can be installed and used with the Tester.

Message Interfaces has information about testing applications with non-DDS interfaces.

Google Protocol Buffers describes the Tester's support for Google Protocol Buffers.

An *Appendix* contains the complete *formal description of the Tester Scripting language* for reference.

1.4 Conventions

The icons shown below are used in PrismTech product documentation to help readers to quickly identify information relevant to their specific use of Vortex OpenSplice.

Icon	Meaning
⚠	Item of special significance or where caution needs to be taken.
𝑖	Item contains helpful hint or special information.
Windows	Information applies to Windows (*e.g.* XP, 2003, Windows 7) only.
Unix	Information applies to Unix-based systems (*e.g.* Solaris) only.
Linux	Information applies to Linux-based systems (*e.g.* Ubuntu) only.
C	C language specific.
C++	C++ language specific.
C#	C# language specific.
Java	Java language specific.

2

Introduction

This section provides a brief introduction to the Vortex OpenSplice Tester.

2.1 Features

The Vortex OpenSplice Automated Testing and Debugging Tool provides an easy way of displaying messages produced in Vortex OpenSplice and also provides means to publish messages manually or with a script.

(The OpenSplice Automated Testing and Debugging Tool is usually referred to as *Tester*; the name `ospltest` is used when referring to the executable program.)

This tool is made with the software tester, and the way he performs his job, in mind. A pre-defined list of topics of interest can be provided. For all topics a reader is created in the correct partition. Once started, the tool receives all instances of the topics of interest and will display them in the sample list in the order they were produced (using the source time stamp). This makes it very easy to see when topics are produced and in what order. It also provides feedback about unexpected updates.

Other features of Tester include the ability to:

- dump a selection of topic(s) to a file
- dump all logged topic instances to a file
- filter the sample list based on key
- filter the sample list based on key and topic name
- filter the sample list based on key
- create a script with a selection of previously sent or received topics
- compare topic samples
- edit topic samples and then write them or dispose the topic instance
- create new topic samples and write them or dispose the topic instances

2.2 Location of Tester in the OpenSplice architecture

Tester is complementary to OpenSplice Tuner (`ospltun`). Tuner supports 'white box' application monitoring and tuning, and Tester supports 'black box' system testing, debugging, data capture, analysis, and visualization.

2.3 Things to Know

⚠ **NOTE**: Tester uses the internal Control & Monitoring (C&M) API for access to OpenSplice. At this time Tester only supports Vortex OpenSplice systems.

Tester can be used both locally (*via* shared memory or single process) and/or remotely (*via* a SOAP connection to the SOAP service).

Tester uses a periodic poll to read data (the default poll period is 250 ms). The normal restrictions for storage scope apply (only keys defined with the topic separate topics for reading, if topics with the same key are produced within a polling period, then only the last topic is read).

Tester uses the default QoS for writing (as provided by the first application which registers the topic) and the weakest QoS for reading. However when specifying the topic in `add topic` (or `add topics`) or in the topic file the QoS can be given, this QoS must be compatible with the topic QoS as defined when the topic was registered.

⚠ **NOTE**: In order for the Tester system browser to correctly show the complete system, OpenSplice Durability services have to be properly configured so that the transient 'built-in-topics' are properly aligned when new nodes join the system. Monitoring the built-in topic sample set on different nodes will quickly reveal any failure in correct lining-up of transient data (in which case there will be different sets visible on different nodes). Monitoring the `DCPSHeartbeat` built-in topic will reveal fundamental connectivity issues in your system (you should see as many unique heartbeats as there are interconnected nodes in the system).

2.4 Prerequisites

Tester is included in the standard OpenSplice installation.

Tester's minimum system requirements are the same as for OpenSplice itself; please refer to the *Release Notes* for both Tester and OpenSplice for details. The Vortex OpenSplice *Getting Started Guide* contains additional information about installation and configuration on various systems.

Note that to compile plugins you will also need to have `ant` and JDK1.6 installed (see *Getting Started with a Message Interface*). Tester has been implemented in the Java Language, and it uses the Vortex OpenSplice Command and Management (C&M) API.

Although Tester uses the C&M API, it doesn't depend on a locally installed or running instantiation of OpenSplice. It can operate either 'co-located' with a running DDS target system, or it can operate in 'remote-connection' mode (like the Tuner).

When Tester is run on the same platform as Vortex OpenSplice, it uses the `OSPL_HOME` environment variable to find the necessary OpenSplice library files. It also uses `OSPL_URI` as its default OpenSplice configuration.

When Tester connects to a remote 'target' platform using SOAP it doesn't use any local environment variables, it just needs to be installed on the machine where you run it.

(Note that the Vortex OpenSplice Tuner can be started with a `-uri` command-line parameter (see how to *start a remote connection*). This is a new feature that is actually used by the Tester in the system browser, where you can spawn a Tuner (see how to *spawn a tuner from the system browser*) that then connects to the node/application that the browser is pointing to.)

3

Getting Started

This section describes how to use the Tester's main features.

3.1 Starting and Stopping Tester

Tester may be started by running the OpenSplice Tester application or from a command prompt and oslptest with the following command line arguments:

-? or -help Display the command line options

-ns No splash screen

-uri <uri> URI to connect

-nac No auto-connect upon application start

-s <path to script> Script to run

-b <path to batch file> Batch script to run

-noplugins Do not process plugins

-plugindir <dir> Extra plugin directory to search

-headless Run script or batch script without the GUI

-rc <port> Enable remote control (via <port>)

-dsl <language> Default script language

-l <path to topic file> Load topics from file

(These preferences can also be set from the menu *File > Preferences*.)

3.2 Starting - Local Connection

Windows On Windows, use either of the shortcuts created by the installer (on the desktop and in *Start > Programs*) to start Tester.

Linux On Linux go to the installation directory and execute the command:

```
%  ospltest
```

This will start Tester with separate windows.

3.3 Starting - Remote Connection

To connect to a remote platform, execute the command:

```
%  ospltest -uri http://perf1.perfnet.ptnl:50000
```

(Port number 50000 is the default port in a standard DDS shared-memory deployment.)

3.4 Stopping

Stop Tester either by using the menu option *File > Exit* or by clicking on the main window *'close'* button ⊠.

3.5 Remotely Controlling Tester

Starting Tester in remote control mode *e.g.* `"ospltest -rc <port> -headless"` allows Tester to be controlled from another application, shell script, *etc.*.

Use cases for remote control include:

- Using Tester in combination with a commercial or proprietary test system;
- Within a continuous build and test environment this would provide more options to control DDS testing in combination with other application-specific testing;
- In an integrated development environment like Eclipse using Junit for testing.

A Tester instance is controlled *via* a TCP/IP connection. Text-based commands are sent over this connection.

The remote control application can be used by executing the command:

```
ospltestrc [-p <port>] [-h <host>] <command>
```

where

<host> is the host name of the machine that the Tester you wish to control is running on

<port> is the port that Tester is listening on (specified by the -rc option when Tester was started)

<command> is the command to send to Tester.

The remote control commands are:

stop Terminate the Tester instance

batch <batch name> Execute the batch with <batch name>

script <script name> Execute the script with <script name>

scenario <scenario> Execute a scenario which is provided in full text on a single line (new lines `"\n"` are replaced by `" "`)

connect <optional uri> Connect to a specified or the default Domain uri

disconnect

When a command is completed the following is reported on a single line:

```
Done
```

When a batch is executed, for each scenario two lines are returned to the test controller:

```
Scenario: <index> of <count> execute: <scenario name>   Scenario:
<index> result: <result>
```

3.6 Trying out Tester

Once you have started Tester, you can get a feeling for how to use it with a few simple exercises:

- Create a default reader for some of the registered topics.

- Double-click one of the samples and see all the fields of the topic.

- Browse through the list of samples using the arrow keys.

- Select a topic in the sample list and press *[F9]*, then select a field for display in the sample list.

- Select another topic (it need not be of the same type as the one displayed in the topic instance window) and press *[F2]* for a comparison between the two topics.

- Select a topic in the sample list or in the topic list and press *[F4]*, then in the `Write` topic window set the fields to the desired value and write or dispose the topic.

- Choose *File > Dump* on the sample list and save the information of the topic samples in the sample list to a file.

- Have a try with the scripting, it can make your life a lot easier (especially with recurring tasks).

The rest of this section describes the features that you will use when you try these exercises.

3.7 Tester Windows

3.7.1 Main Window

Once started, Tester presents the user with the following main window.

Tester main window
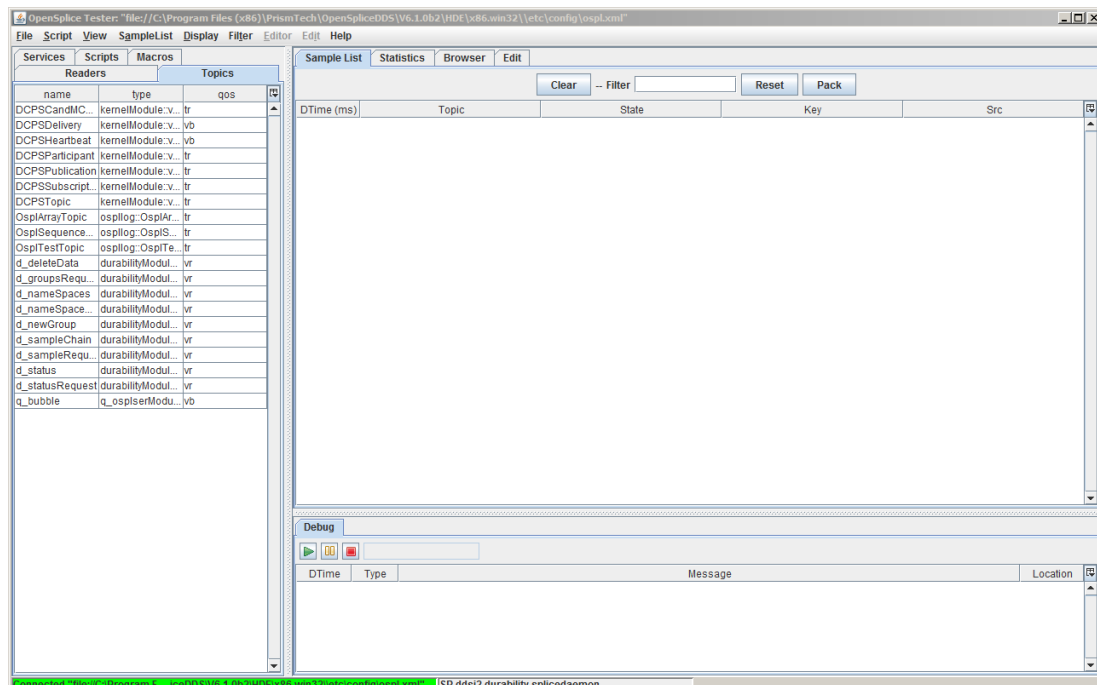
The Command Menu (below) provides direct access to most of the Tester capabilities.

Tester command menu

The Tester main window has three sub-frames:

1. *Main* tabbed frame for selecting items from a list, such as topics, scenarios, and readers or writers.

2. *Working area* frame where you will do most of your work such as editing scenarios, investigate samples, and capturing statistics

3. *Debug* frame used to debug scripts and macros.

3.7.2 Overview Windows

The user can select the type of resource to work with by selecting tabs. These can be the Services and Topics in the system, the Scripts and Macros they have installed, or the Readers for the current Tester timeline.

Tester resource tabs

Services

Lists the installed services. This is a read-only list.

Tester Services list

Scripts

The script list provides a convenient way of selecting an existing script for editing or execution. The list is filled at startup or when clicking the *Refresh* button. All files in the specified script directory are added to the list. The script directory (or directories) are specified in the preference page.

A script can be selected in the script editor by single-clicking the entry in the table. When the entry is double-clicked the script is loaded in the script editor and executed.

Tester scripts tab
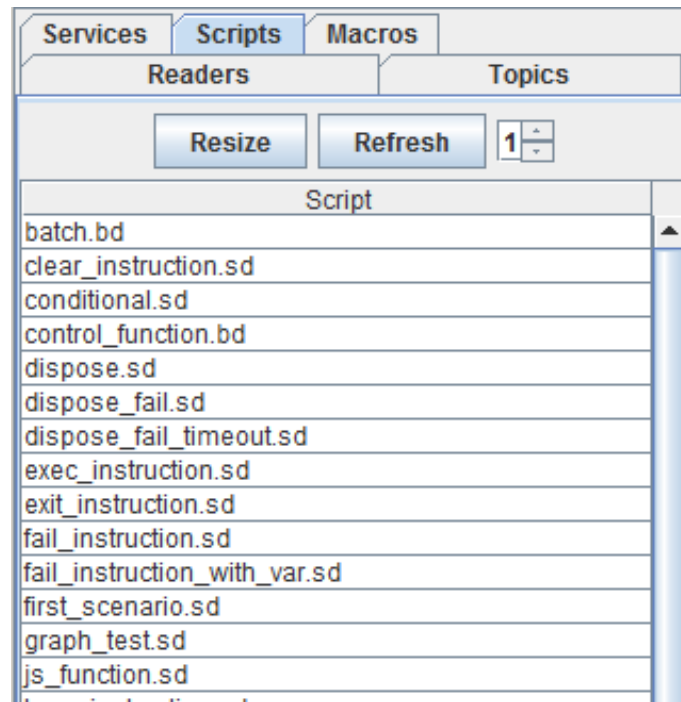
Macros

The Macros List is similar to the Scripts List.

Tester macros tab
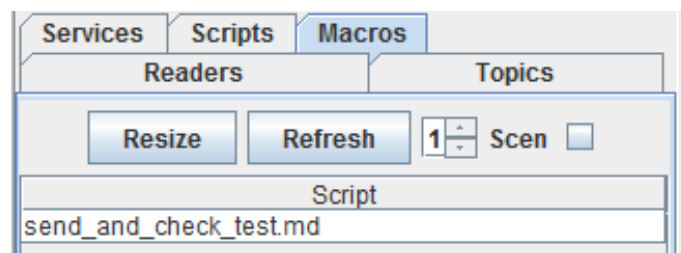
Topics

The topics list displays the list of registered topics.

Tester topics tab

Readers

The readers list displays the readers (and implicit topic writers) for the current Tester timeline. The default name for a reader is the same as the name of the topic it is subscribed to. For each reader the count of received samples (as available in the sample list) is displayed. A check box is provided for changing the read state or the show state. When *Read* is unchecked the reader stops reading topics. When *Show* is unchecked the topic of that topic will not be displayed in the sample list.

Tester readers tab

3.7.3 Working Windows

These windows support testing activities.

Sample List Window

Used to view and generate samples for the current timeline (Readers).

Sample list window
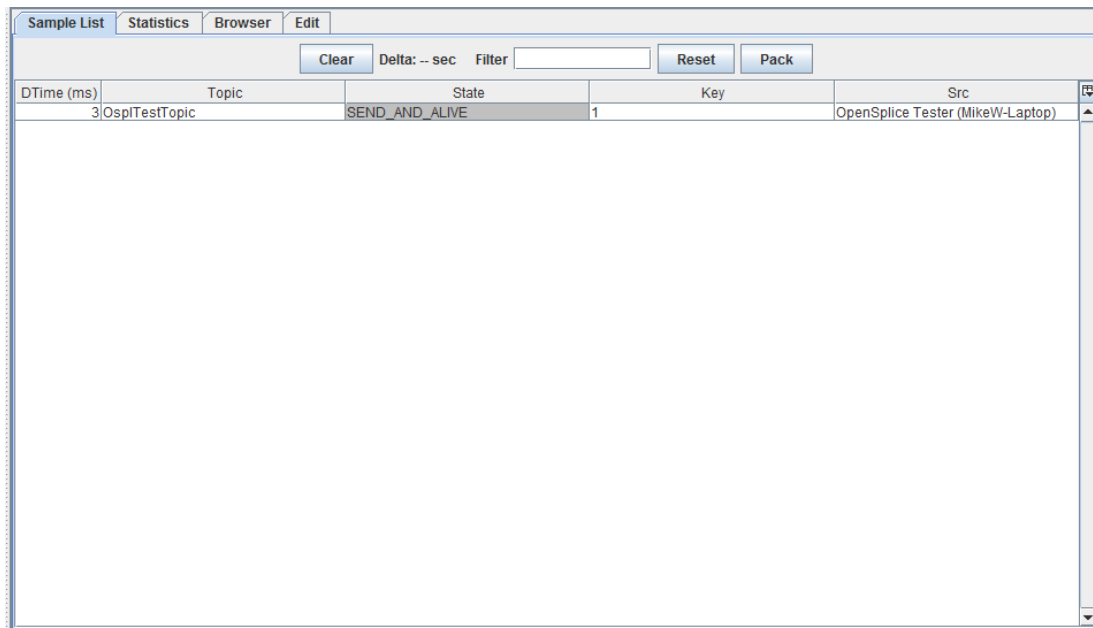
Statistics Window

The statistics window provides statistics for the topics in use, like write count, number of alive topics, *etc.*. Statistics are gathered from the local copy of OpenSplice. To gather statistics from remote nodes, use OpenSplice Tuner.

Statistics window
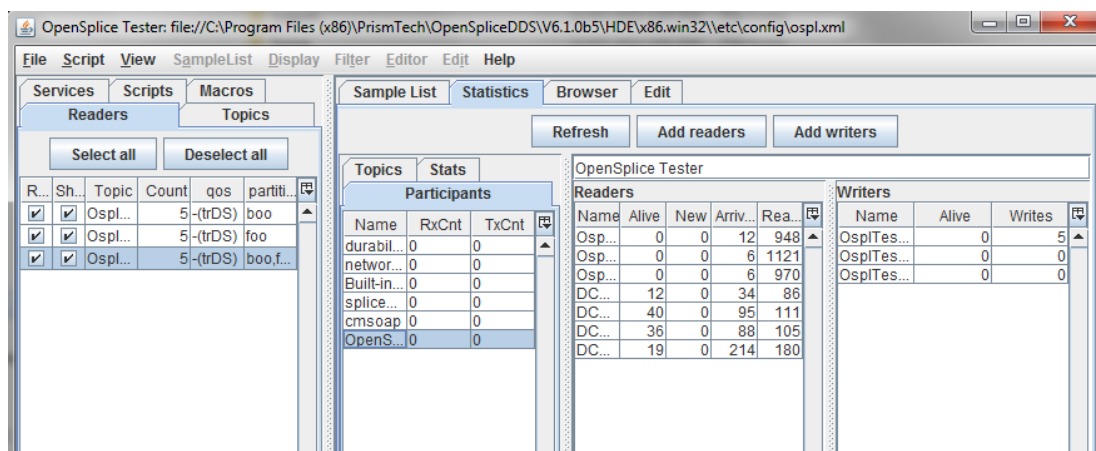
Browser Window

The browser window provides information about nodes, executables, participants (applications), readers, writers and topics. Information can be browsed by selecting a node/executable participant or a topic. When an executable or participant is selected the reader and writer lists (subscribed and published topics) for that executable/participant are shown. Together with the topic name concise information about the QoS and partition is shown. When the mouse cursor is hovered over the QoS value the hint will show detailed information about the QoS.

When a topic is selected the list of participant readers (subscribe) and writers (publish) are shown, together with concise information about the QoS and partition. By selecting a row in either the reader of writer list the compatible readers/writers will be shown in green and non-compatible (by QoS/partition) readers/writers will be shown in red.

Tester browser window
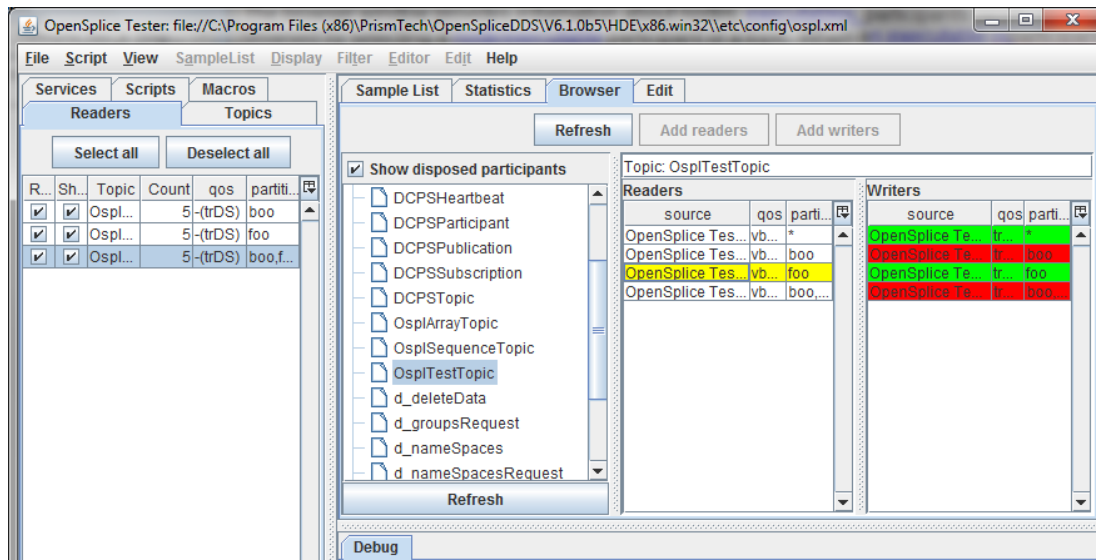
3.7.4 Scripting Windows

Edit Window

The script window is used for editing scripts. The editor supports syntax highlighting, auto-completion, and more.

Script editing window
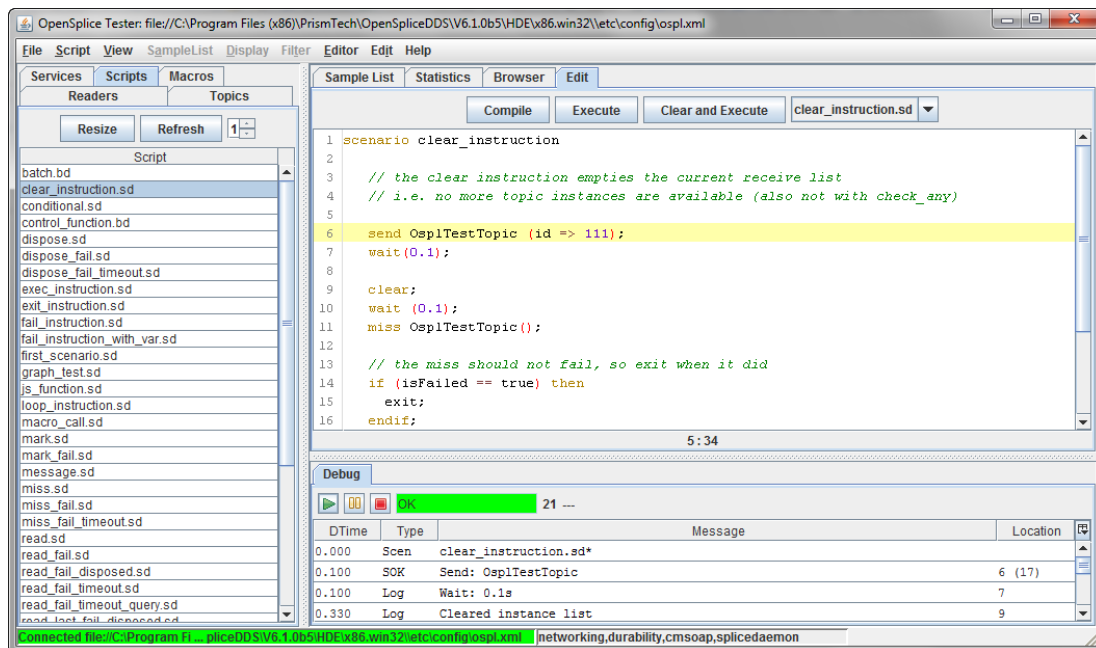
Debug Window

The debug window displays compile and execution results. Details can be filtered. Positive results are highlighted with green, negative results are highlighted with red.

Debug window

DTime	Type	Message	Location
0.430	Log	Evaluate false == true = false	
0.430	Log	If is false: false	14
0.430	CFNK	Expected message not received: OsplTestTopic with key 111	19
0.430	Log	Fail reversed	16
0.430	EOK	Execution finished	

3.7.5 Other Windows
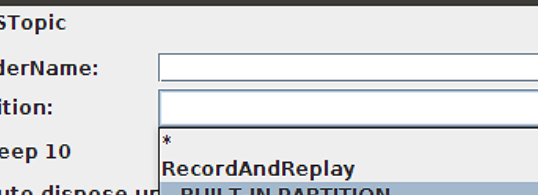
The following dialog windows will be used.

Add Reader Window

Used to create/define a new Reader.

The dialog provides a drop-down list of existing partitions to choose to create the new Reader in.

Add Reader dialog
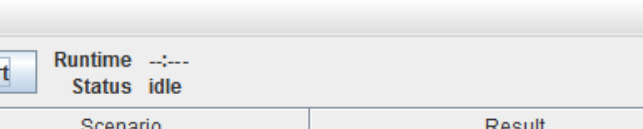
Batch Window

Used to Start a batch scenario and display the test results.

Batch Execute Scenarios window

Batch Results Window
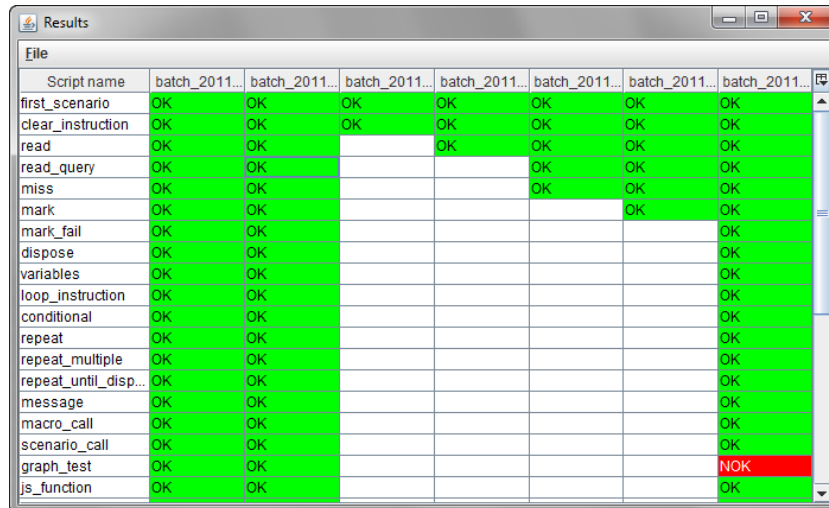
Displays the detailed results of a batch of scripts. Detailed individual test result can be viewed by double-clicking on a test result.

Batch results window

Detailed Batch results log

Chart Window

Used to plot topic field values.

Topic field values graph
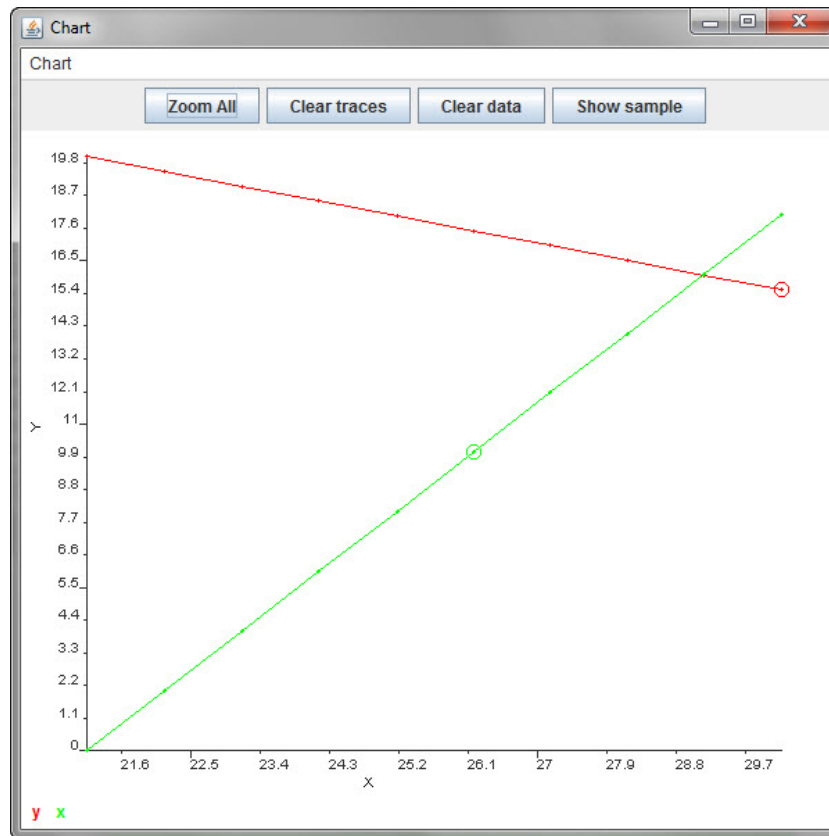
Edit Sample Window

Used to create samples for a selected topic.

Edit sample window

Topic Instance Window

The topic sample window is used for displaying field values of a topic. It can be opened by double-clicking a sample in the sample list or by pressing *[F3]* (additional) or *[F2]* (additional with compare) in the sample list while a sample is selected. Special fields are highlighted with colors:

> *Key field* (Green)

> *Foreign key* (Yellow)

Different (compare only) (Red)

Not existing (compare only) (Orange)

When a field is selected, *[Ctrl+H]* will toggle between normal and hexadecimal representation, and *[Ctrl+D]* will toggle between normal and degrees/radians representation.

4

Familiarization Exercises

This section gives step-by-step instructions for using the Vortex OpenSplice Tester to perform many typical tasks to help you become familiar with the way it operates.

The exercises in this section assume that OpenSplice and the Tester have been succesfully installed. These illustrations make use of the example data supplied with the product.

4.1 Starting the Tester

OpenSplice must already be running before you start the Tester.

Step 1: Start Vortex OpenSplice

Step 2: Start the Tester:

- On Linux, run `ospltest`.

- On Windows, choose Vortex OpenSplice Tester from the *Start* menu or run `ospltest` from the Vortex OpenSplice command prompt.

Starting Tester
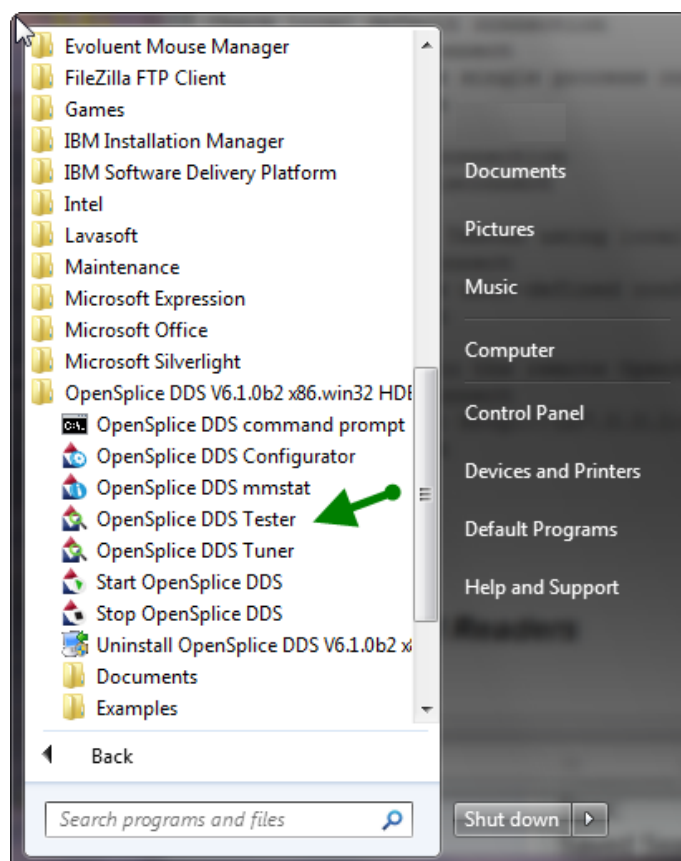
4.2 Connection management

When it starts, the Tester will automatically try to establish a connection to a running instance of OpenSplice using the default URI. You can also make or break connections from the main window by following the steps given below.

The command line option `-nac` stops Tester from making a connection at startup, and with the `-uri` command line option a connection to an alternative URI can be made at startup.

4.2.1 To Connect to a local OpenSplice instance

Step 1: Choose *File > Connect*.

Step 2: Set the path or Browse to the configuration file (*e.g.*, `file://<OpenSplice install dir>/etc/config/ospl.xml`).

Step 3: Click the *OK* button.

Connecting to a local OpenSplice instance
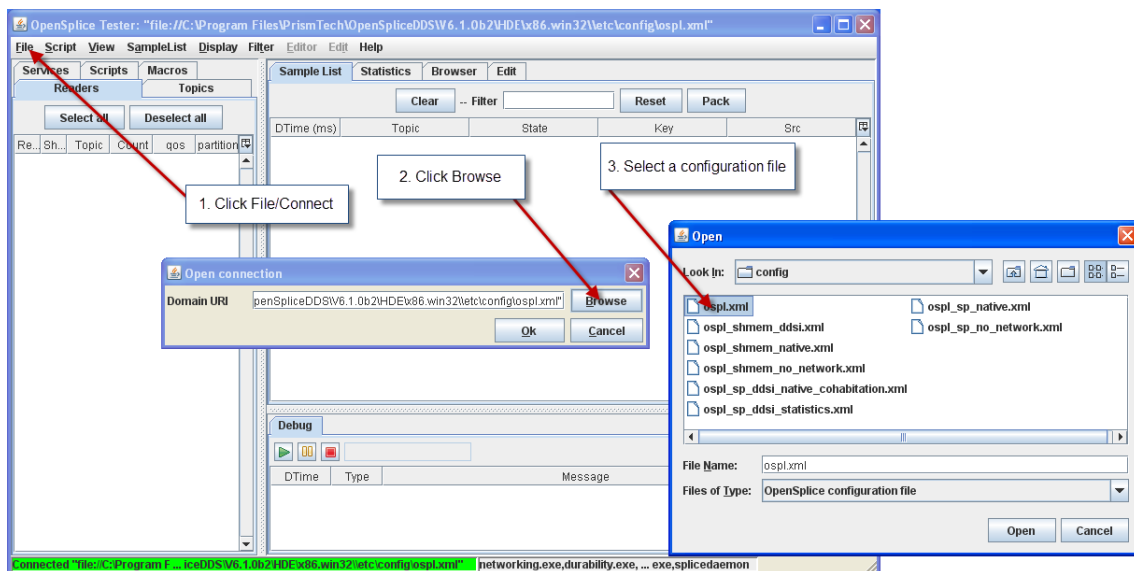
4.2.2 To Connect to a remote OpenSplice instance

Step 1: Choose *File > Connect*.

Step 2: Enter the URI for the remote OpenSplice system (*e.g.*, `http://127.0.0.1:8000`).

Note: The port number must be set to the port number as configured for the SOAP service of the remote OpenSplice instance.

Step 3: Click the *OK* button.

Connecting to a remote OpenSplice instance
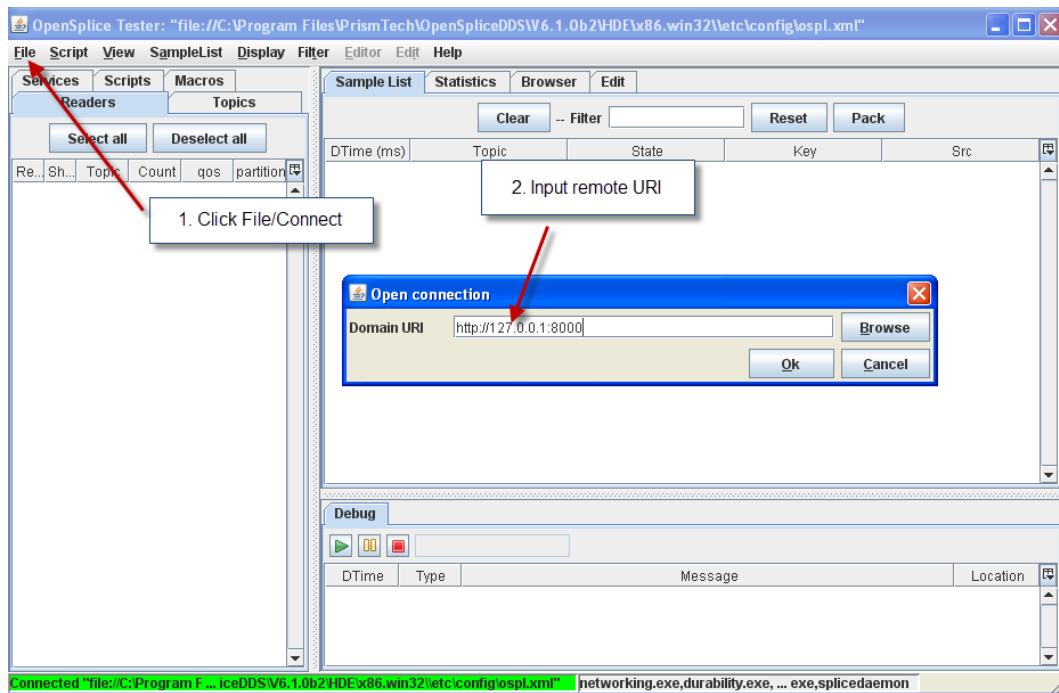
4.2.3 To Disconnect

Step 1: Choose *File* > *Disconnect*.

4.2.4 To Exit Tester

Step 1: Choose *File* > *Exit* or click the *Close* button ☒ on the Tester main window.

4.3 Topics and Readers

Tester can subscribe to multiple topics. These Readers will comprise a timeline for testing. Samples of those topics are automatically read and displayed in the Sample List. Tester readers can also be used to write or edit samples.

4.3.1 The Topic list

Check the Topic list. Make sure that the Tester is connected to the default URI.

4.3.2 To View a Topic's Type definition

Step 1: Select the *Topics* tab.

Step 2: Right-click *OsplTestTopic*.

Step 3: Choose *View Topic Type* from the pop-up menu. The *View Topic Type* window will appear, displaying the type name and type description for the chosen topic.

Viewing a topic's type
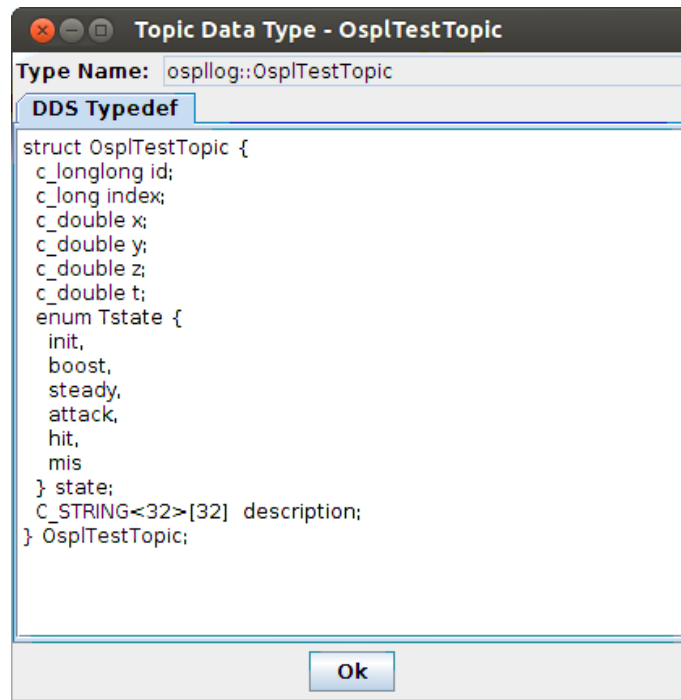
4.3.3 To Add a Reader from the Topic list

Step 1: Select the *Topics* tab.

Step 2: Right-click *OsplTestTopic*.

Create Readers from the Topics list
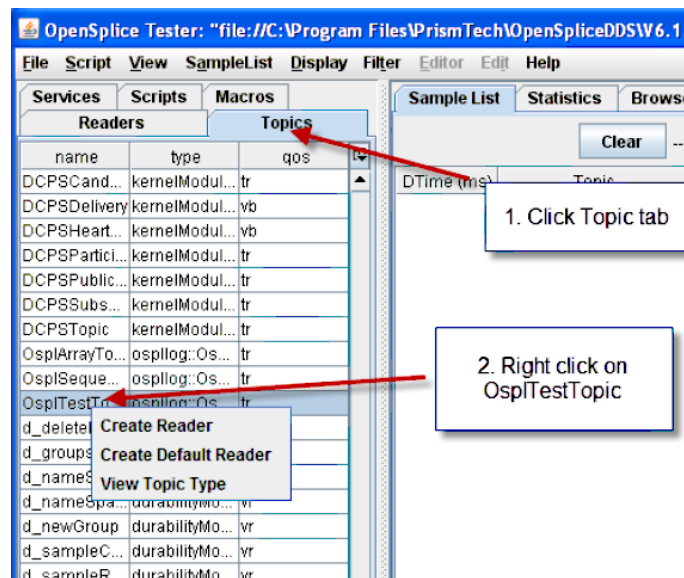
Step 3: Choose *Create Default Reader* from the pop-up menu. The reader will automatically be named the same as the topic.

Step 4: Choose *Create Reader* and modify the (writer) QoS or reader name if desired.

Step 5: Click *Add*.

Create myReader
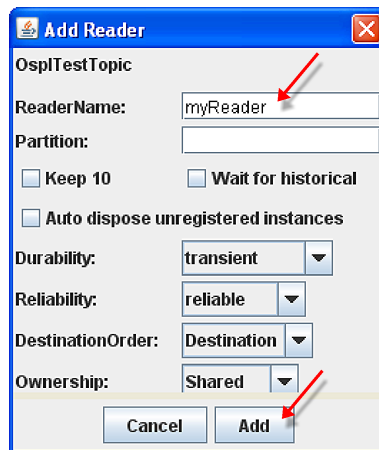
Step 6: Open the *Readers* tab and you will see the readers you just created.

4.3.4 To Add a Reader from the File menu

Step 1: Select the *Readers* tab.

Step 2: Choose *File > Add Reader*.

Step 3: Select *OsplTestTopic* from the drop-down list.

Step 4: Click *Add*.

Adding a Reader from the File menu
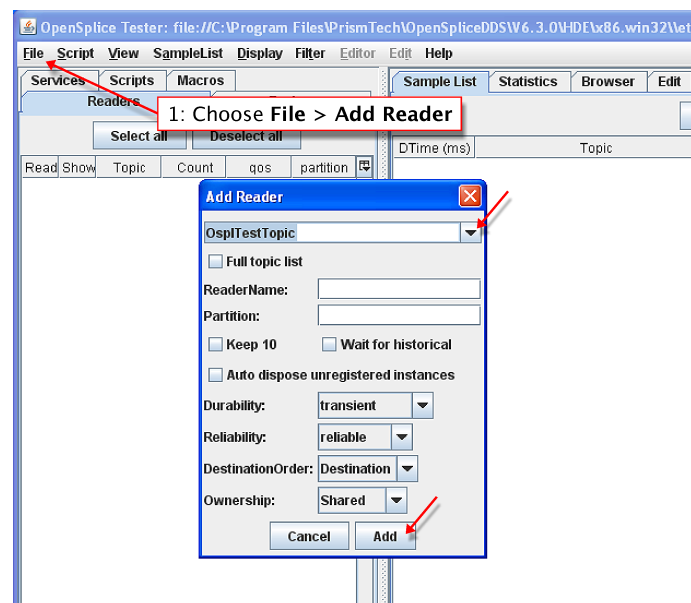
4.3.5 To Add multiple Readers to the Tester timeline

Step 1: Choose *File > Add Readers*.

Step 2: Type `ospl` in the filter field to limit the list of topics. Select *OsplArrayTopic* and *OsplSequenceTopic*.

Step 3: Click *Add*.

Adding multiple Readers
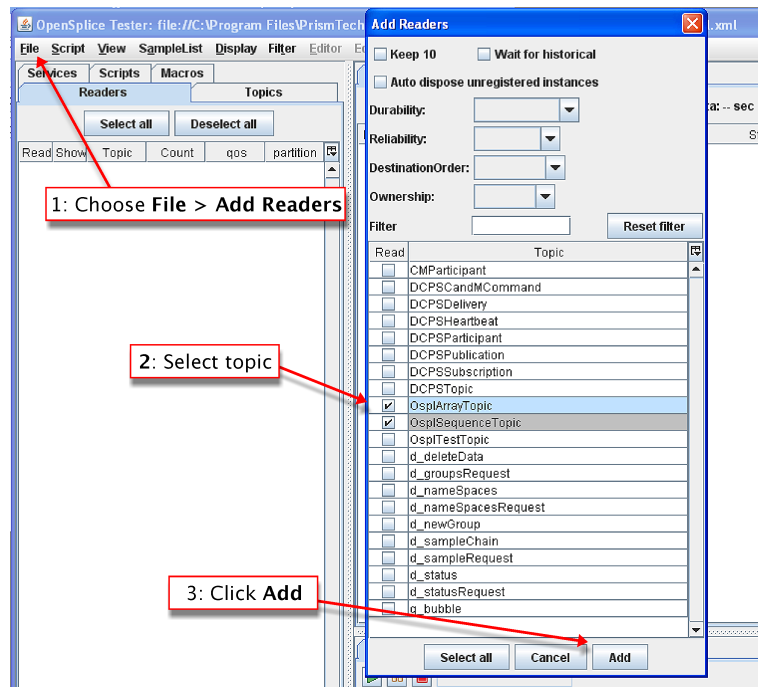
4.3.6 To Save the current Readers to a file

If you need to preserve the Readers for a timeline, you can save the current Readers list.

Step 1: Choose *File* > *Save Readers List*.

Step 2: Enter a name for the new file.

Step 3: Click *Save*.

4.3.7 To Remove all Readers

Step 1: Choose *File* > *Remove all Readers*.

4.3.8 To Load Readers from a saved file

Step 1: Choose *File* > *Load Readers List*.

Step 2: Select the name of the saved file.

Step 3: Click *Load*.

4.3.9 To Delete a Reader

Step 1: Select *OsplTestTopic* reader from the list.

Step 2: Press the *[Delete]* key or right-click on *OsplTestTopic* and choose *Delete Reader* from the pop-up menu.

4.4 Samples

4.4.1 Writing and Editing Samples

To Write Sample Topic data
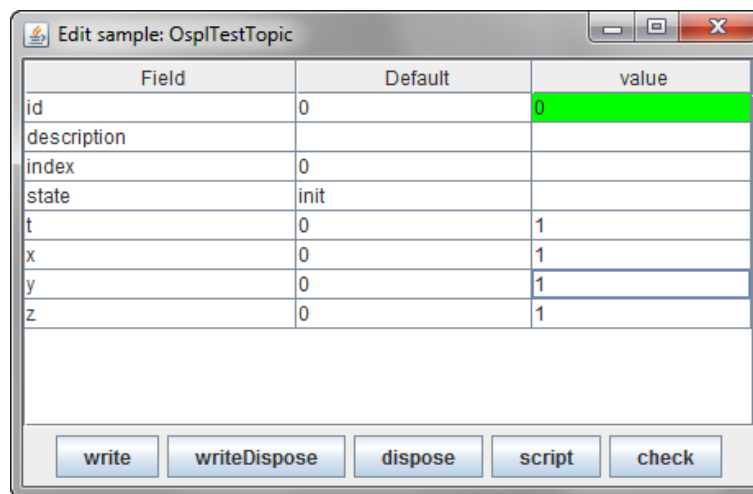
Step 1: Select *OsplTestTopic* reader from the list.

Step 2: Press *[F4]* or choose *Edit Sample* from the pop-up menu.

Step 3: Enter following values for the fields in the list:

   id: 0, *t*: 1, *x*: 1, *y*: 1, *z*: 1

Entering sample topic data

Step 4: Click the *write* button.

Step 5: Close the *Edit Sample* window.

To display detailed information on sample data

Step 1: Double-click on the first *OsplTestTopic* sample in the *Sample List* window.

Display detailed sample data information

Sample: OsplTestTopic	
source_timestamp	1315320828s.810113666ns
daytime	10:53:48.810 (53 06)
insert_latency	18249806ns
relative_time	-51.138
view_state	new
sample_state	not_read
instance_state	alive
valid_data	valid_data
state	NEW, NOT_READ
partition	*
source	OpenSplice Tester (simon-3b1569dcf)
qos	2ba1636a.0000012b

Field	value
id	0
index	0
x	1
y	1
z	1
t	1
state	init
description	

Sample	<<	>>
Same topic	<<	>>
Same instance	<<	>>

More information on sample info

The detailed sample data table displays sample info data of a given sample. Some fields are derived from middleware sample info data, while others are not. What follows is a description of some of those fields.

- The *insert_latency* is a calculated value representing the difference between the sample's insert timestamp, and its write timestamp, as it is received in the sample info.

- The *relative_time* is a Tester specific time measurement, in seconds. It does not represent any actual timestamps from the middleware. Its main use is determining the time elapsed between a Tester scenario script execution start and the sample receipt. It's mainly meant as a loose measurement of time tracking since start of sample reading or start of a script scenario, and not as a strict real-time middleware timing.

To Display extra fields

By default the Sample List displays topic-independent fields. You can add topic-specific fields as follows:

Step 1: Select any sample.

Step 2: Press *[F9]* or right-click and choose *Select Extra Fields* from the pop-up menu.

Step 3: Click to select ('check') *x*, *y*, *z* and *t*.

Selecting extra fields to display
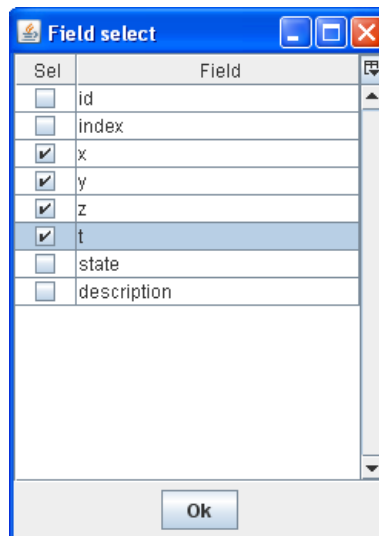
Step 4: Click *OK*.

The selected fields will be added to the Sample List.

New fields added
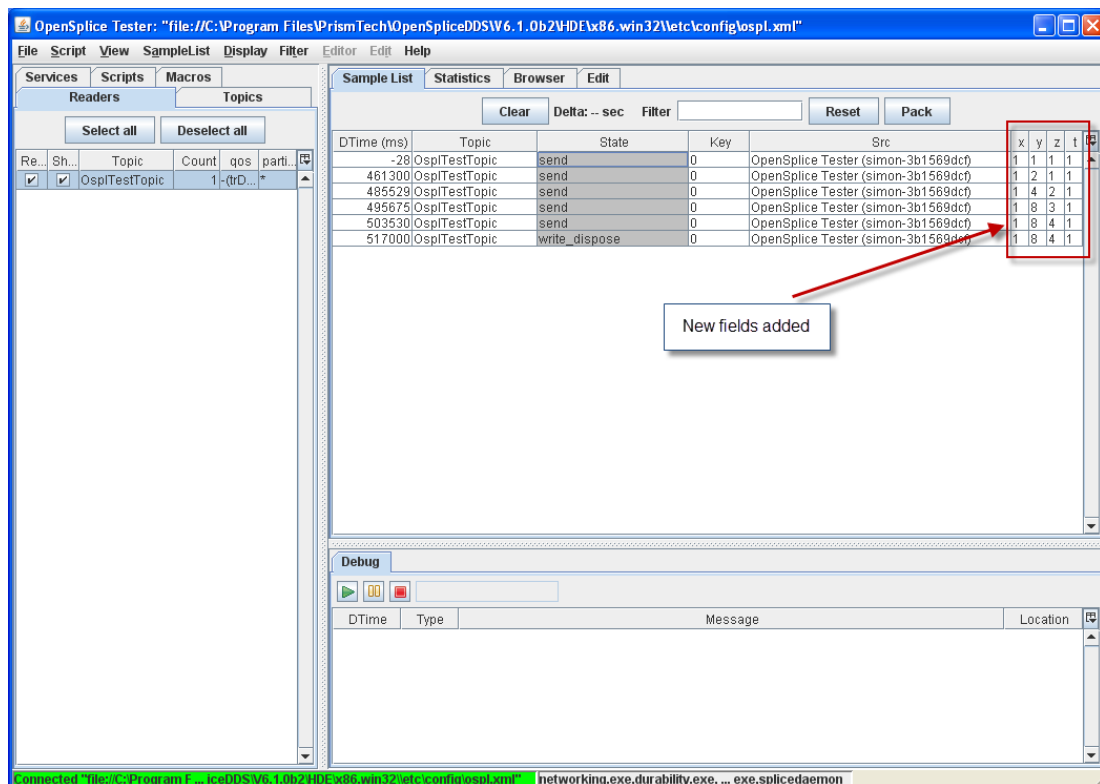
To Edit a sample

Step 1: Select the first sample.

Step 2: Press *[F4]* or choose *Edit Sample* from the pop-up menu.

Step 3: Enter following values in the fields:

> *id*: 0, *x*: 1, *y*: 2, *z*: 1, *t*: 1

Step 4: Click *Write*.

Step 5: Enter following values in the fields:

   id: 0, *x*: 1, *y*: 4, *z*: 2, *t*: 1

Step 6: Click *Write*.

Step 7: Enter following values in the fields:

   id: 0, *x*: 1, *y*: 8, *z*: 3, *t*: 1

Step 8: Click *Write*.

Step 9: Enter following values in the fields: *id*: 1, *x*: 1, *y*: 8, *z*: 4, *t*: 1

Step 10: Click *WriteDispose*.

To Compare two samples

Step 1: Double-click the sample with the values *id*: 0, *x*: 1, *y*: 4, *z*: 2, *t*: 1.

Step 2: Select the sample with the values *id*: 0, *x*: 1, *y*: 8, *z*: 3, *t*: 1.

Step 3: Press *[F2]* or choose *Compare Samples* from the pop-up menu.

Comparing samples
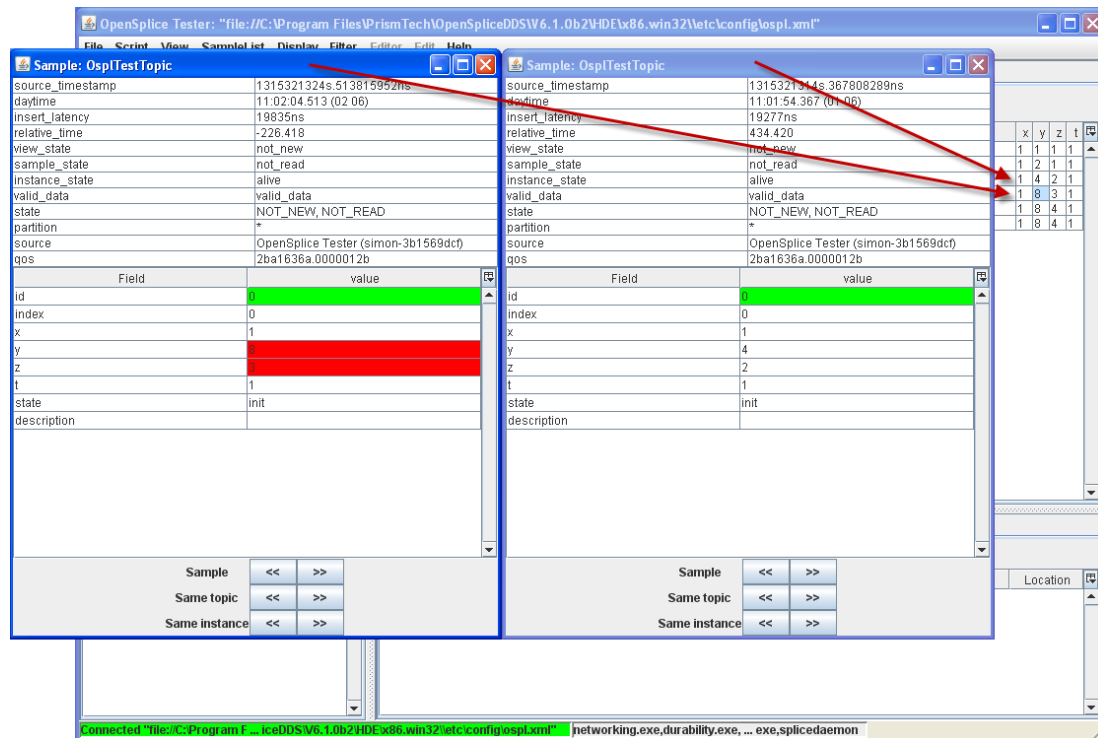
4.5 Filtering

Filtering: un-filtered Topic list
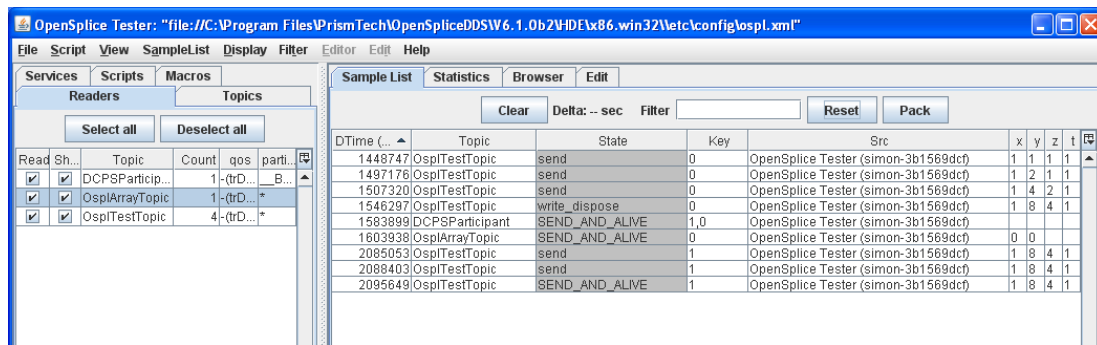
4.5.1 To Filter the Sample List on a Topic

Step 1: Select the *OsplTestTopic* sample.

Step 2: Press *[F5]* or choose *Filter on Topic* from the pop-up menu.

Sample List filtered by Topic
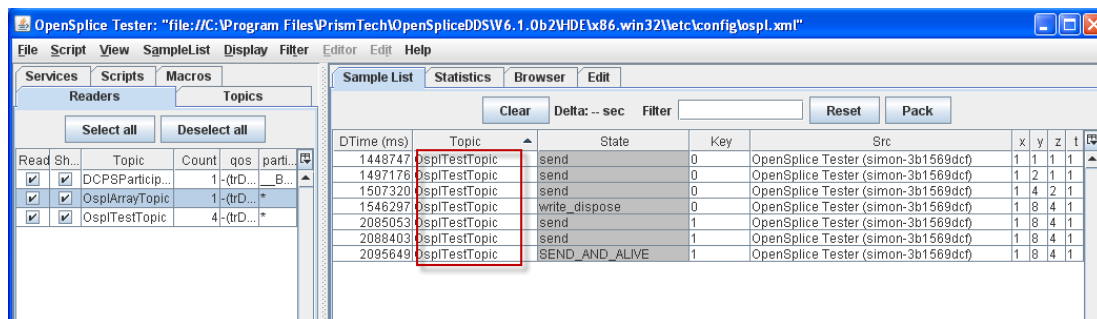
4.5.2 To Reset Filters and display all samples

Step 1: Press *[F7]* or choose *Reset filter* from the pop-up menu or click the *Reset* button on the *Sample List* window.

4.5.3 To Filter on both Topic and Key

Step 1: Select *OsplTestTopic* with *id(key)*: `1`.

Step 2: Press *[F5]* or choose *Filter on topic and key* from the pop-up menu.

Sample List filtered by Topic and Key
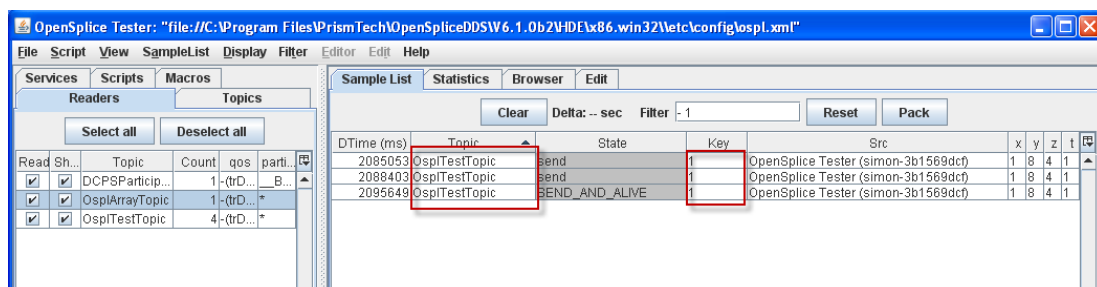
4.5.4 Filter samples on State

Step 1: Select a sample with a *State* of `SEND_AND_ALIVE`.

Step 2: Choose *Filter on State* from the pop-up menu.

Sample List filtered by State
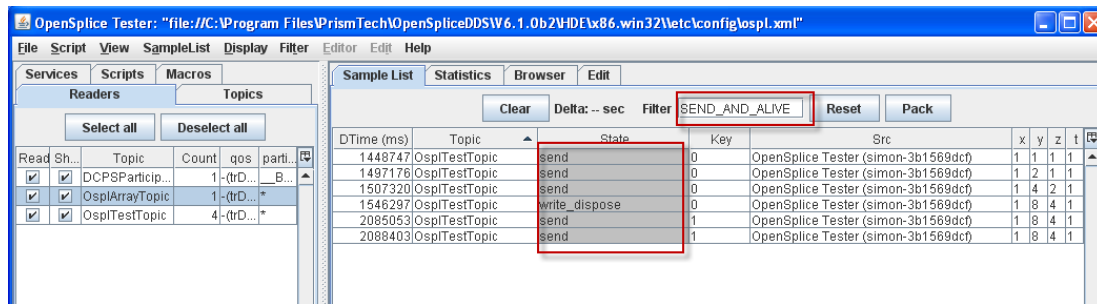
4.5.5 To Filter Samples on Key value

Step 1: Select *OsplTestTopic* with *id(key)*: `0`.

Step 2: Choose *Filter on key* from the pop-up menu.

Sample List filtered by Key value
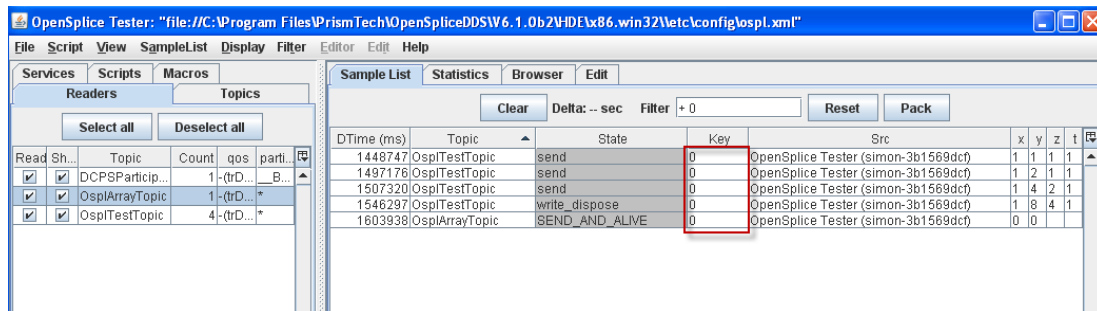
4.5.6 Filter on column text

Step 1: Select the *State* column of any sample.

Step 2: Choose *Filter on column text* from the pop-up menu.

Step 3: Type in `send`.

Step 4: Press the *[Enter]* key.

Sample List filtered by column text
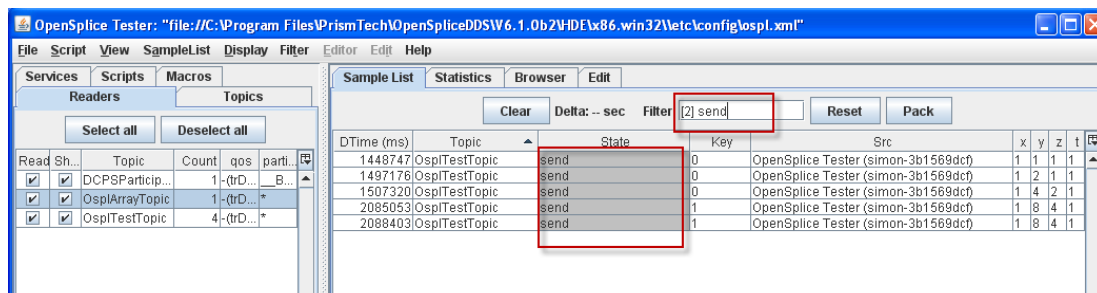
4.5.7 Find specific text

Step 1: Press *[Ctrl+F]* to open the *Find* dialog.

The Find dialog
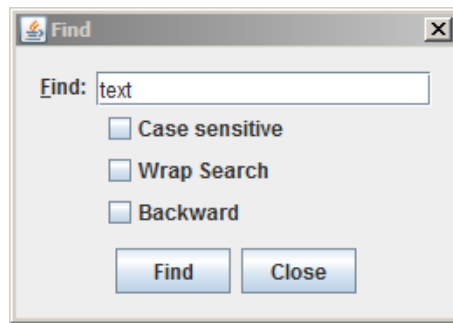
Step 2: Type in the text to search for, and select any of the options if required.

Step 3: Click *Find*. The first occurrence of the search text is highlighted.

Step 4: Click *Find* again to find the next occurrence of the search text.

4.5.8 Global Topic filters

Topic filters
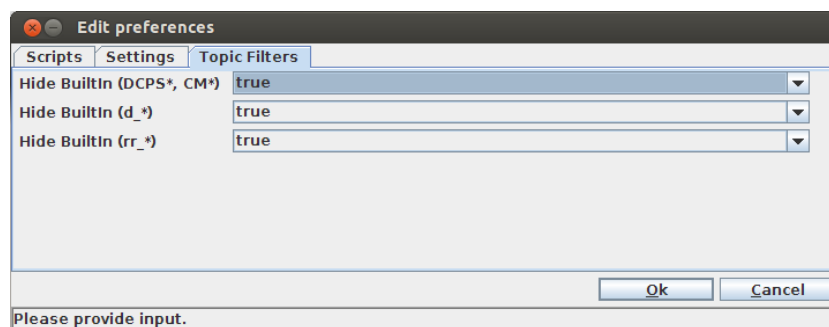
It is possible to completely hide certain topics from all views in the tool. These are global topic filter preferences, which can be enabled (`true`) or disabled (`false`). They are accessed from the preferences window in *File > Preferences > Topic Filters*. Enabling a given filter will hide the matching topics from the Topics table and from the system browser view.

4.6 Working with Samples

4.6.1 To Delete a column from the Sample List table

Step 1: Select the *x* column of any sample.

Step 2: Press the *[Delete]* key.

Column deleted from Sample List display
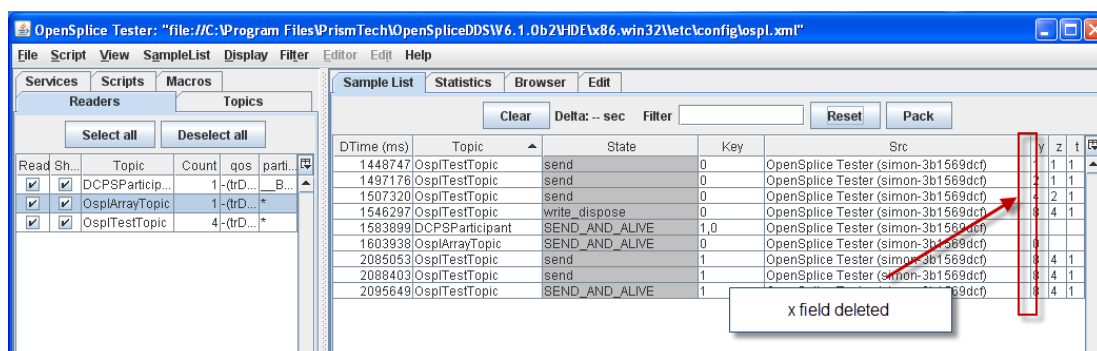
4.6.2 To Chart Sample Data

Using any list of samples:

Step 1: Select the *z* column of any sample and press the *[X]* key.

Step 2: Select the *y* column of any sample and press the *[Y]* key.

Step 3: Choose *SampleList > Show Chart* or press *[Alt+Shift+C]* to display the chart.

Chart of Sample data
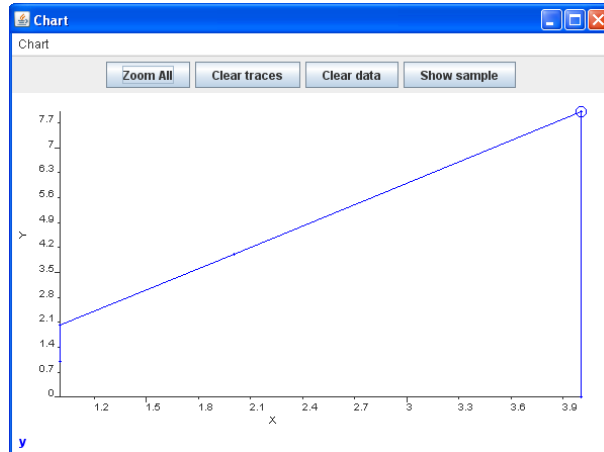
4.6.3 To Dump a sample list to a file

Step 1: Choose *SampleList > Dump*.

Step 2: Enter a name for the file to save.

Step 3: Click *Save*.

4.6.4 To Dump selected Samples only

Step 1: Select *OsplTestTopic* with *key*: 1.

Step 2: Choose *SampleList > Dump Selection*.

Step 3: Enter a name for the file to save.

Step 4: Click *Save*.

4.6.5 To Dump to a CSV format file

Step 1: Choose *SampleList > Dump to CSV*.

Step 2: Enter a name for the file to save.

Step 3: Click *Save*.

4.6.6 To Dispose data with Alive state

Step 1: Choose *SampleList > Dispose Alive*.

Disposing data with 'Alive' state
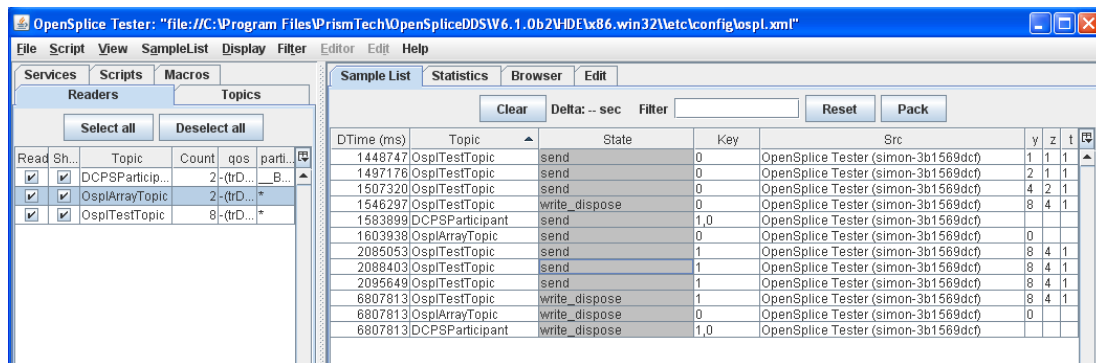
4.6.7 To Translate Sample data to test script

Step 1: Choose *SampleList > Diff Script*.

The Scripting commands to replicate all of the sample data will be inserted into the current scenario in the *Edit* window.

4.6.8 Translate selected sample to test script

Step 1: Select a set of samples.

Step 2: Choose *SampleList > DiffScript Selection*.

The Scripting commands to replicate this subset of the sample data will be inserted into the current scenario in the *Edit* window.

4.6.9 To display samples with not_alive_no_writers state

There is a setting to control whether Tester's active data readers ignore samples whose state is `NOT_ALIVE_NO_WRITERS`.

If an application data writer has a QoS of `autodispose_unregistered_instances` set to `false` and then `unregister_instance` is called on a data writer for some instance, a sample reaches matching data readers with the `no_writers` state.

The default setting is `true`, which means that these samples are ignored and not displayed.

'Max samples' and 'not_alive' options
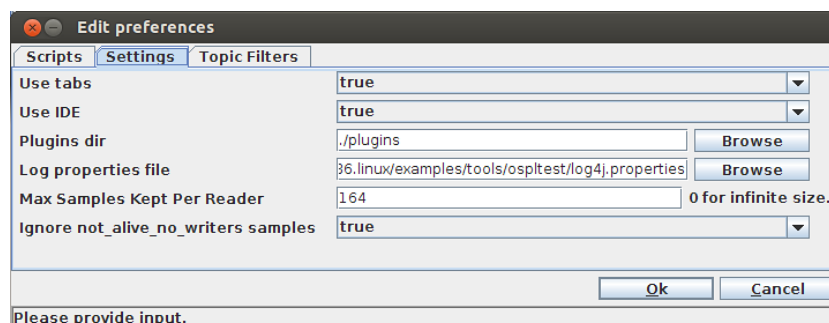
4.6.10 To control the number of samples kept per reader

It is possible to limit the number of samples kept per reader. This setting, *Max Samples kept per reader*, is accessed by choosing *File > Preferences > Settings*.

This setting accepts integer values. The default value is 0, which means that there is no limit imposed on the number of samples that will be stored.

4.7 Groups

4.7.1 Definition of a Group in Tester

Vortex OpenSplice supports setting of the Presentation policy of publishers and subscribers. As a result, Tester can set these policies on its own created publishers and subscribers and be able to write/read coherent sets of data into/from the system.

Just as in Tester where data reader and data writer are aggregated into a *Reader* and listed in the Readers list, a custom created publisher and subscriber pair are aggregated into a *Group* and are listed in the Groups list. A Group has a defined Partition and Presentation QoS policy that is set on creation and is set to its contained publisher and subscriber.

4.7.2 To Add a Reader under a Group

Step 1: Choose *File > Add Reader*.

Step 2: Select *OsplTestTopic* from the drop-down topics list.

Step 3: Check the box labeled *Create as group*.

Step 4a: Choose a name for the Group (Group name must be unique and non empty).

Step 4b: Fill in desired *Partition*, and Presentation policy settings (*Access scope*, *Coherent access* and *Ordered access*).

OR

Step 4c: Select an already existing Group from the drop-down Groups list. Partition and Presentation form elements will populate to existing values for the selected Group and become disabled.

Step 5: Click *Add*.

Create a Reader under a Group

Completing this action will create the *Groups* tab on the main panel if this is the first Group that has been created in the current session.

⚠ If there is already a reader existing on the selected topic, then the new reader must have a unique name assigned to it, else the new reader will not be created and assigned to the Group.

4.7.3 To Add multiple Readers under a Group

Step 1: Choose *File > Add Readers*.

Step 2: Select *OsplTestTopic*, *OsplArrayTopic*, and *OsplSequenceTopic* from table.

Step 3: Check the box labeled *Create as group*.

Step 4a: Choose a name for the Group (Group name must be unique and non empty).

Step 4b: Fill in desired *Partition*, and Presentation policy settings (*Access scope*, *Coherent access* and *Ordered access*).

OR

Step 4c: Select an already existing Group from the drop-down Groups list. Partition and Presentation form elements will populate to existing values for the selected Group and become disabled.

Step 5: Click *Add*.

Create multiple Readers under a Group

Completing this action will create the *Groups* tab on the main panel if this is the first Group that has been created in the current session.

⚠ If there is already a reader existing on a selected topic, then the new reader must have a unique name assigned to it, else the new reader will not be created and assigned to the Group. In this case, one should use the method in To Add a Reader under a Group.

4.7.4 To Publish coherent sets

Once at least one Group has been created, the *Groups* tab will become visible. It contains a list of currently active groups, their QoS policies, and current number of Tester readers (readers under the subscriber, writers under the publisher).

There are two actions available to rows of this table: *Delete Group*, and *Publish Coherent Data*.

Deleting a Group will also delete the readers that it owns.

If *Publish Coherent Data* is selected, then a new window will appear.

Publish coherent sets menu
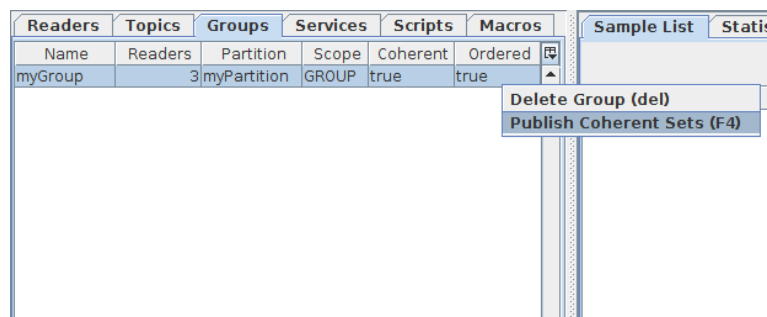
Coherent Publisher Window
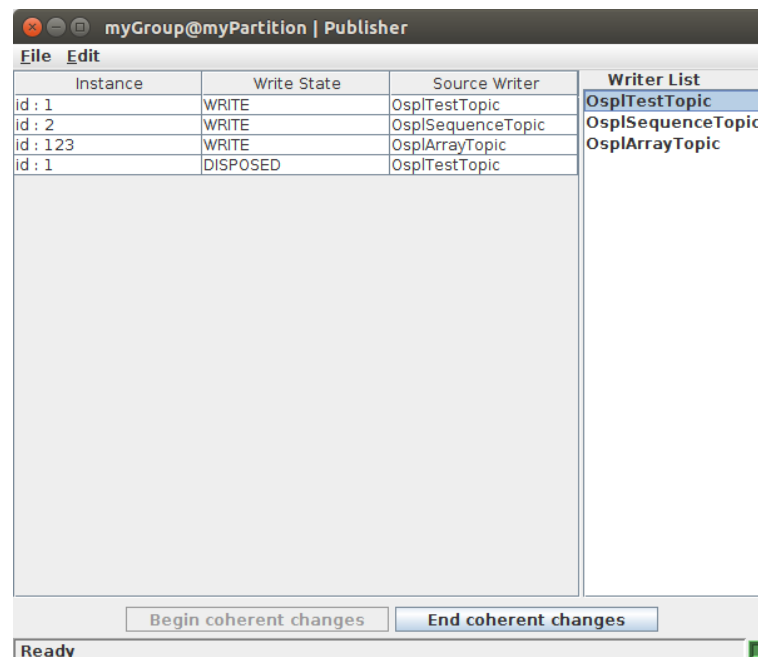
This new view contains two parts: the list of writers currently under this Publisher, and the user data table containing data about the outgoing samples in a coherent set.

The actions available to the user in this view are:

- *Begin coherent changes*
- *End coherent changes*
- *Refresh writer list*

Begin coherent changes will set the publisher in coherent mode. In this state, samples written by this publisher's writers will not be made visible to remote readers (under a subscriber with matching Presentation policy) until the *End coherent changes* action is made. Otherwise, samples written while the publisher is not in coherent mode are published normally.

Samples can be constructed and written from this view by either double clicking or right clicking on a writer in the writers list and selecting *Write data*. This brings up a writer window identical to the one in Writing and Editing Samples, only this one will update the Coherent publish window with written data.

When a sample is constructed in the writer window and then written, disposed, or any other writer action made, and if the publisher is in coherent mode, then the sample will appear in the Coherent publish window's data table.

The data table displays the written sample's instance key, the outgoing instance state, and the originating writer's name. Samples in the data table can be double clicked to bring up a view of the sample edit window populated with the selected sample's fields and field values.

⚠ Once a sample has been written from the writer window, regardless if the publisher is in coherent mode or not, it is live in the system and cannot be edited.

Once editing a set of coherent data is complete, clicking *End coherent changes* button will notify the publisher that the set is complete, and remote coherent subscribers will allow access to the published data.

The *Refresh writer list* action (accessible from the *Edit* menu or **F5** keystroke) refreshes the current list writers that the publisher owns, if any writers were created or deleted since the creation of the window.

4.7.5 To Subscribe coherent sets

The Group's readers behave in the same way that normal Tester readers behave. All readers are periodically polled for available data and added to the *Sample list*. A Group's reader is polled in such a way that it maintains coherent and ordered access.

4.8 System Browser (Browser window)

4.8.1 Browse tree

The System Browser is used to examine the Nodes, Participants, and Topics in your system using a tree paradigm.

Step 1: Choose *View > Browser* or click the *Browser* tab of the main window.

Step 2: Expand the *all* tree.

Step 3: Select *Tester participant* from the *Browser* tree. Note that your own Tester is highlighted in yellow in the tree.

```
All participants
- OpenSplice Tester
```

Step 4: Select *Built-in participant* from

```
Nodes
  + <your machine name>
    + java.exe
      - Built-in participant
```

Step 5: Select Build-in participant from

```
Nodes
  + <your machine name>
    + java.exe
      - ddsi2
```

Step 6: View readers and writers of durability service. Select *Build-in participant* from

```
Nodes
  + <your machine name>
    + java.exe
      - durability
```

Step 7: View readers and writers of `splicedaemon`. Select *Build-in participant* from

```
Nodes
  + <your machine name>
    + java.exe
      - splicedaemon
```

Browser window
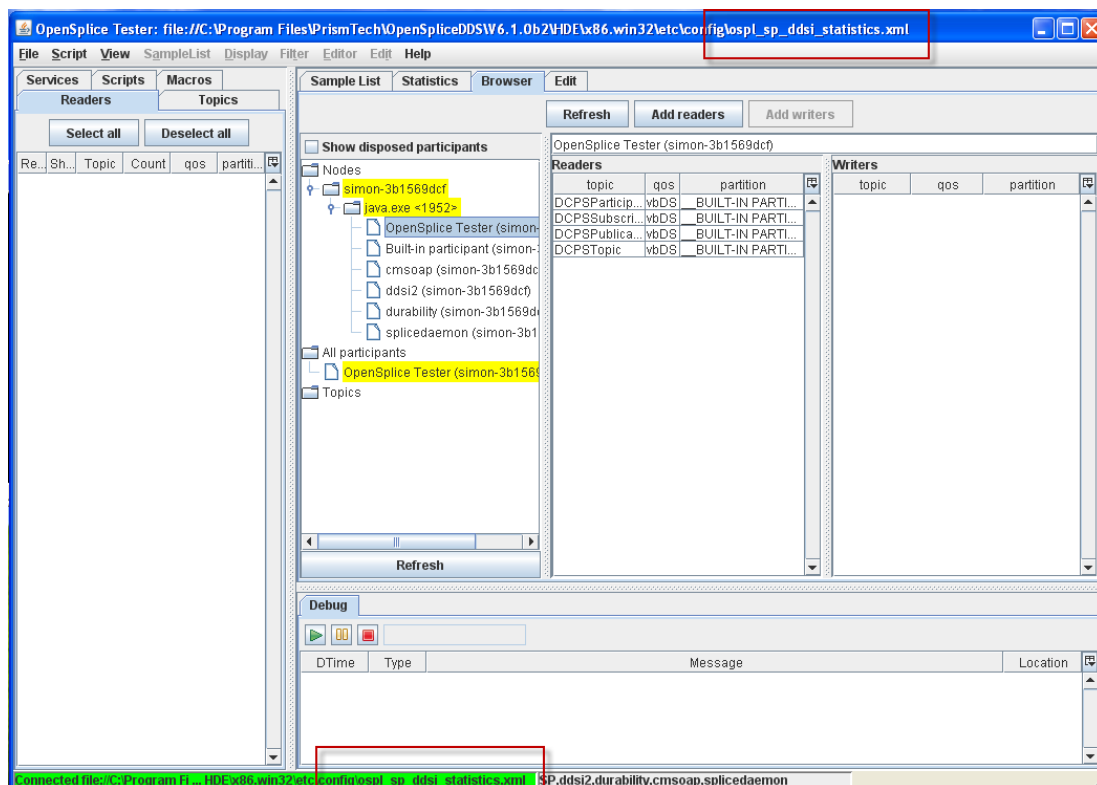
(The red boxes in the illustration indicate the current Open Connection.)

4.8.2 Readers and Writers tables are updated when a new Reader is created

Step 1: Open the *Browser* window.

Step 2: Select *OpenSplice Tester participant* from the *All participants* tree.

Step 3: Create a new `OsplTestTopic` reader (see To Add a Reader from the Topic list for instructions).

Step 4: The Readers and Writers table will be updated.

Readers and Writers table updated
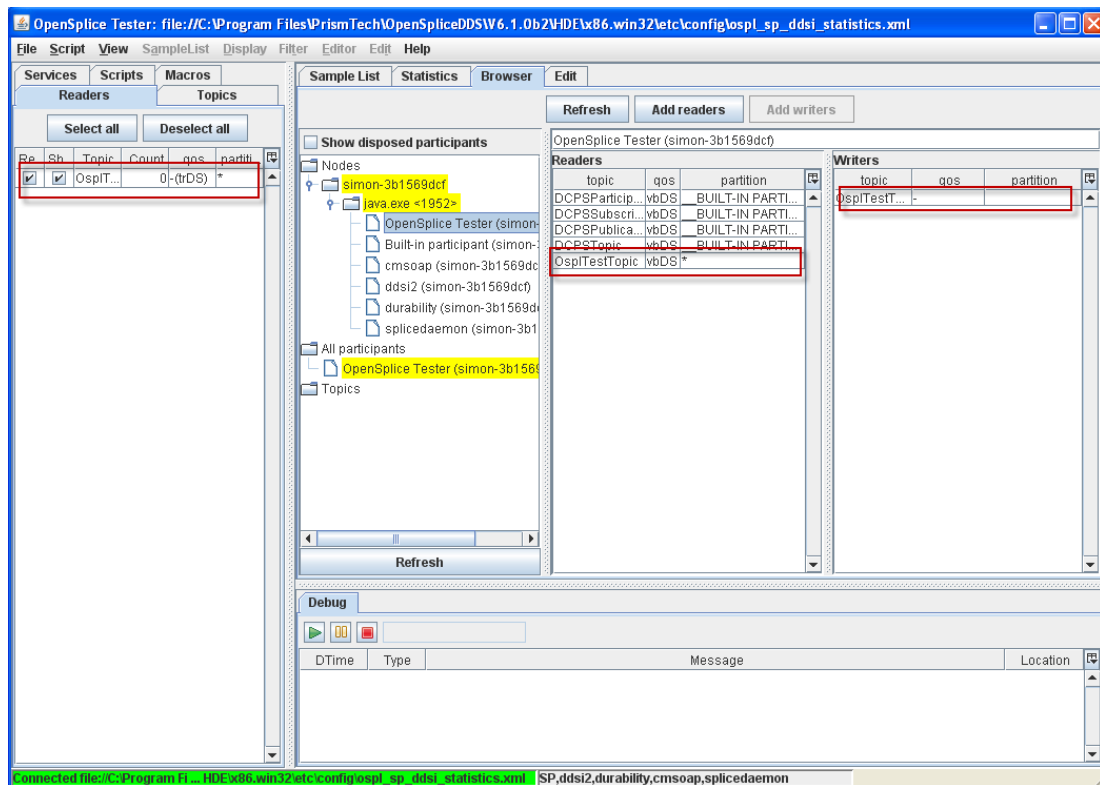
4.8.3 Readers and Writers tables are updated when a new Reader is deleted

Step 1: Open the *Browser* window.

Step 2: Select the *OpenSplice Tester participant* from the *All participants* tree.

Step 3: Delete the existing `OsplTestTopic` reader.

Step 4: The deleted reader will be highlighted with orange to indicate that the reader is disposed.

Reader deleted
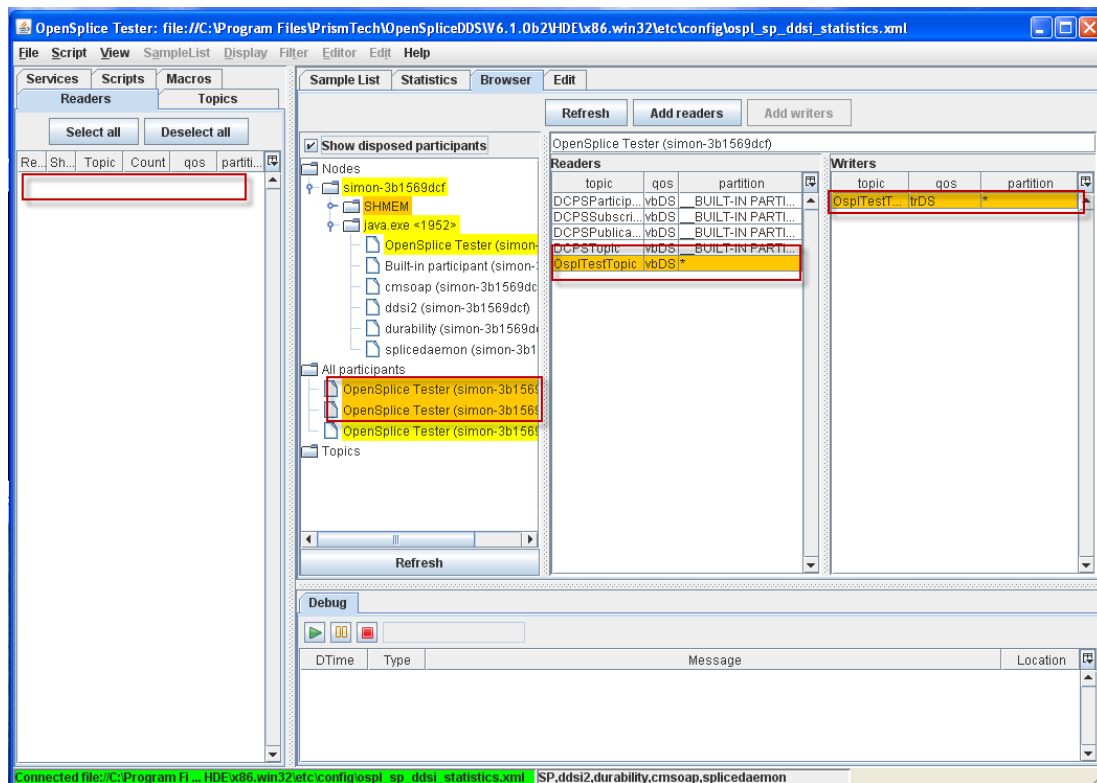
4.8.4 To Check Reader and Writer compatibility

Step 1: Choose *Create Reader* from the pop-up menu from `OsplTestTopic`.

Step 2: Enter `boo` for the name and `boo_partition` for the partition.

Step 3: Create another reader with `hoo` for the name and `hoo_partition` for the partition.

Step 4: Choose *Create Default Reader* to create a default reader.

Step 5: Open the *Browser* window and select *Topics/OsplTestTopic* from the browser tree.

Step 6: Select a Reader with `*` partition from the Readers table.

Reader with '*' partition selected
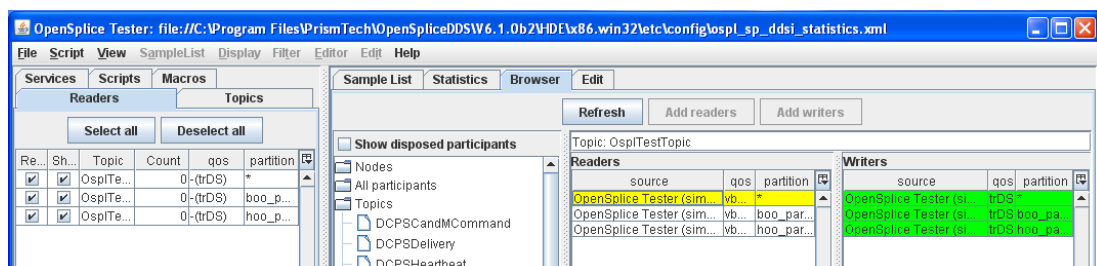
Step 7: Select a Reader with `boo` partition from the Readers table.

Reader with 'boo' partition selected
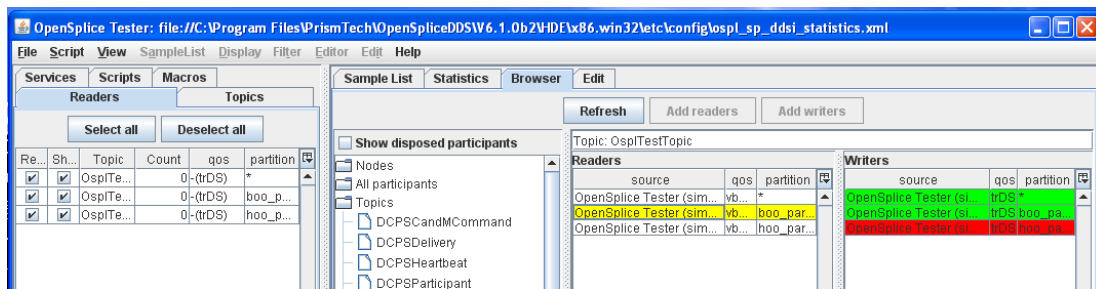
Step 8: Select a Reader with `hoo` partition from the Readers table.

Reader with 'hoo' partition selected
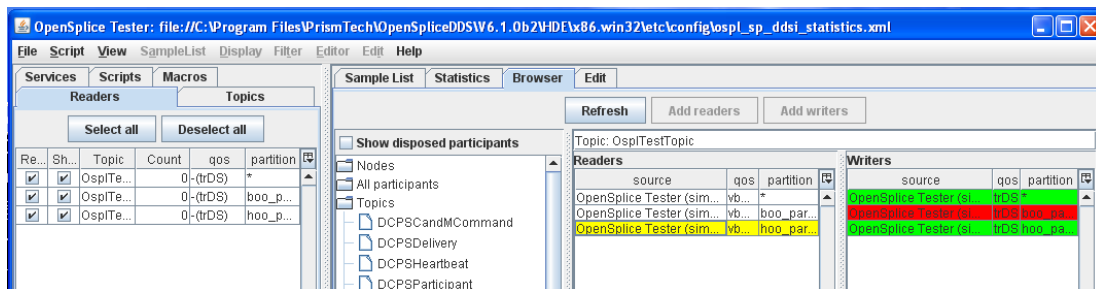
In the *Browser* window, Readers/Writers are highlighted with red to indicate incompatibility with the selected Writer/Reader (yellow).

4.8.5 To Show Disposed Participants from the Browser tree

Step 1: Open the *Browser* window.

Step 2: Select (check) *Show disposed participants*.

Step 3: Expand the *Nodes* tree.

Step 4: Expand the *All participants* tree.

Step 5: De-select (un-check) *Show disposed participants*.

Disposed participants
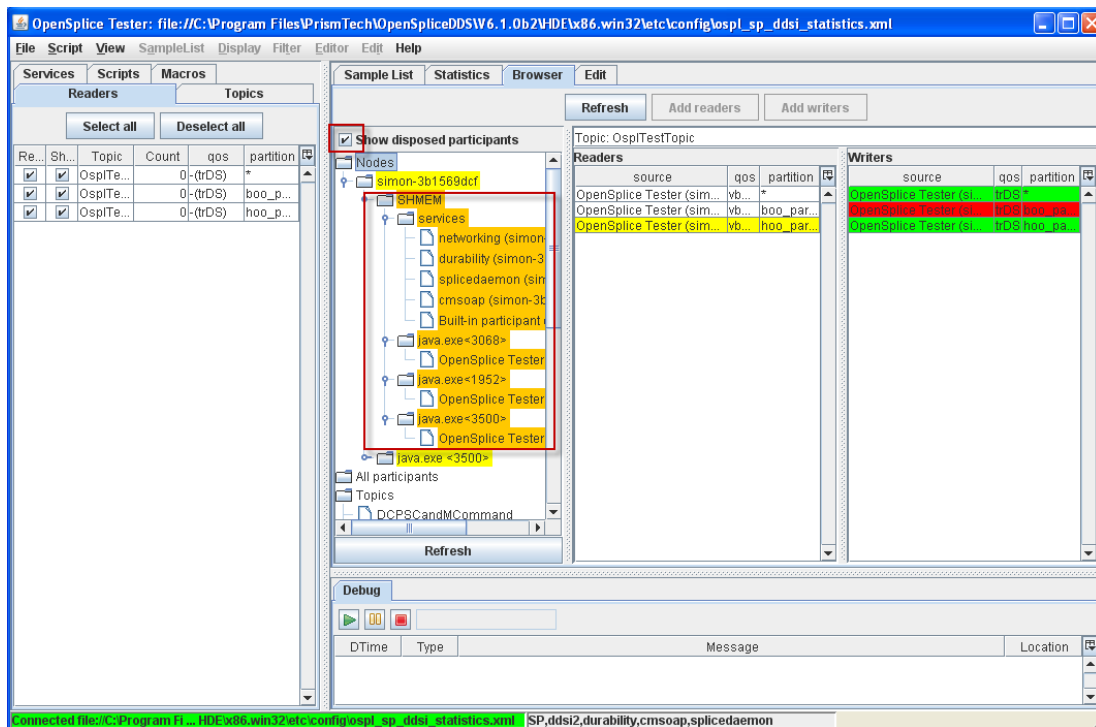
4.8.6 To Spawn a Tuner from the System Browser

Any domain participant that is part of a configuration that includes a SOAP service should have the *Start Tuner* pop-up menu.

Step 1: Connect Tester using the `ospl_sp_ddsi_statistics.xml` configuration file in the `etc/config` directory.

Step 2: Open the *Browser* window.

Step 3: Expand the *Nodes* tree.

Step 4: Right-click on the *OpenSplice Tester* participant.

Step 5: Choose *Start Tuner* from the pop-up menu.

4.8.7 Statistics

First, connect Tester using the `ospl_sp_ddsi_statistics.xml` configuration file in the `etc/config` directory.

i Note that statistics can only be gathered from the Tester process.

The Statistics window
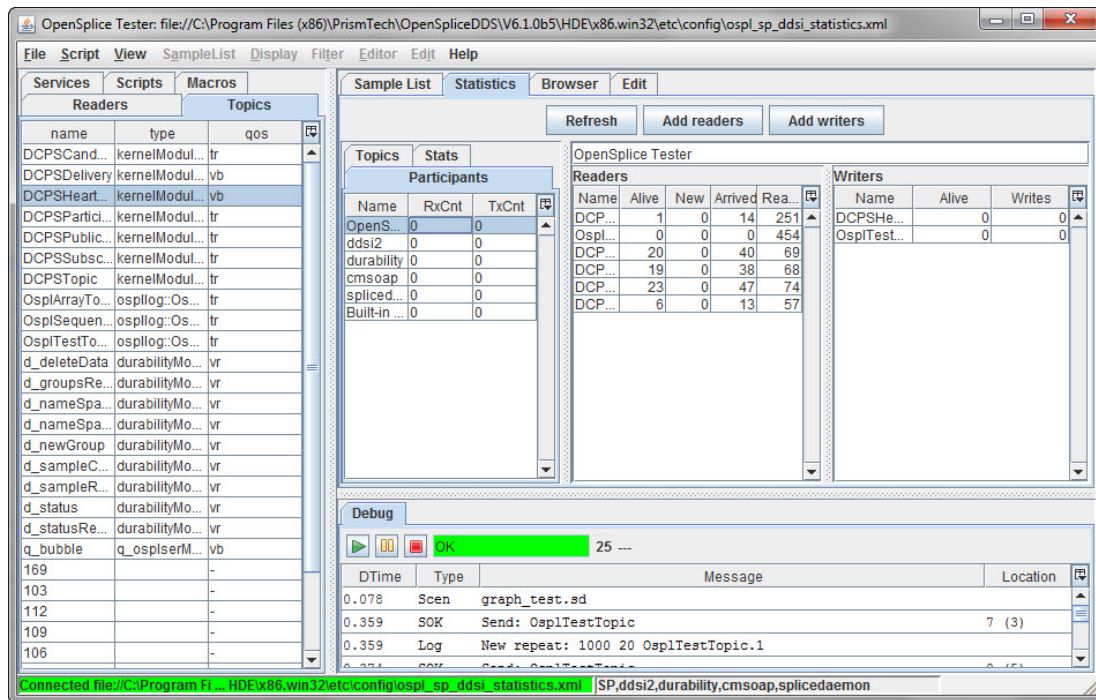
Statistics - participants

Write sample topics and check statistics window content

Step 1: Create a default reader for *OsplTestTopic*.

Step 2: Write four samples.

Step 3: Open the *Participant* tab of the *Statistics* window.

Step 4: Select the *OpenSplice Tester* participant from the list.

Statistics - topics

Write sample topics and check statistics window content

Step 1: Create a default reader for *OsplTestTopic*.

Step 2: Write four samples.

Step 3: Open the *Topics* tab of the *Statistics* window.

Step 4: Select *OsplTestTopic* from the list.

4.9 Scripting

4.9.1 To Create a New Scenario

Note that you can only have *one* scenario open at a time. To avoid losing changes in the current scenario you must save it before creating a new scenario or selecting a different one from the drop-down list of recently-used scenarios (next to the *Clear and Execute* button).

Step 1: Choose *Editor > New Scenario* to create a new scenario and open it in the editor, or if the Editor window is already open, press *[Ctrl+N]* to create and open a new scenario. A warning is displayed if there are unsaved changes in the current scenario.

Step 2: In the *File Save* dialog that appears, specify the location of the new scenario and give it a name.

4.9.2 To Create a New Macro

Step 1: Choose *Editor > New Macro* to create a new macro and open it in the editor, or if the Editor window is already open, press *[Ctrl+M]* to create and open a new macro.

You can have multiple macros open at the same time. Use the drop-down list next to the *Clear and Execute* button to see or select them.

Step 2: In the *File Save* dialog that appears, specify the location of the new macro and give it a name.

4.9.3 To Edit an Existing Scenario or Macro

Step 1: Choose *Editor > Open* from the top menu.

Step 2: In the dialog that appears, type in or browse to the location of the macro or scenario you wish to open, then click *Open*.

4.9.4 To Save an open Scenario or Macro

Save the current scenario or macro.

Step 1: Choose *File > Save* from the top menu or press *[Ctrl+S]*.

Step 2: If the scenario or macro has been saved before, then it is immediately saved, over-writing the previous version.

Step 3: If the scenario or macro has *not* been saved before, a *Save As...* dialog appears; type in or browse to an appropriate location and enter a name for the scenario or macro, then click *Save*.

4.9.5 To Complete and Compile a Scenario

This function 'wraps' the current text in the Edit window with `"scenario"` and `"end scenario"`.

Complete is only used when a new scenario is created without a template, from *DiffScript* or the *Write* button in a sample editor. *Compile* is only needed when you do not want to execute, but just check the syntax.

Step 1: Choose *Edit > Complete* from the top menu.

Step 2: Click the *Compile* button.

Step 3: Click the *Execute* button.

Step 4: Click the *Clear and Execute* button.

4.9.6 Script selection

Step 1: Expand the *Script Selection* drop-down list of recently-used scripts near the *Clear and Execute* button.

4.9.7 Code completion

The Tester has a 'code completion' function which reduces the amount of typing that you have to do and reduces the chances of errors. For example, you can press the *[Ctrl+Space]* keys after you have typed the first few characters of a reader name and the Tester will display a list of the names of the readers which start with the same characters and you can choose the one you want.

Assuming that the *OsplTestTopic* reader already exists, and that a new script is open in the *Edit* tab:

Step 1: Complete the current scenario by choosing *Editor > Complete* from the top menu. (Note that it is generally preferable to start from a template.)

Step 2: Type `send Ospl` then press *[Ctrl+Space]*.

Step 3: 'OsplTestTopic' appears; press *[Enter]* to accept it, and the instruction is completed.

This also pops up the sample editor, enabling you to set the arguments. The sample editor can also be activated by *[Ctrl+Space]* when the cursor is in the instruction, or *[Ctrl+left-click]* on the instruction.

Code completion (send)
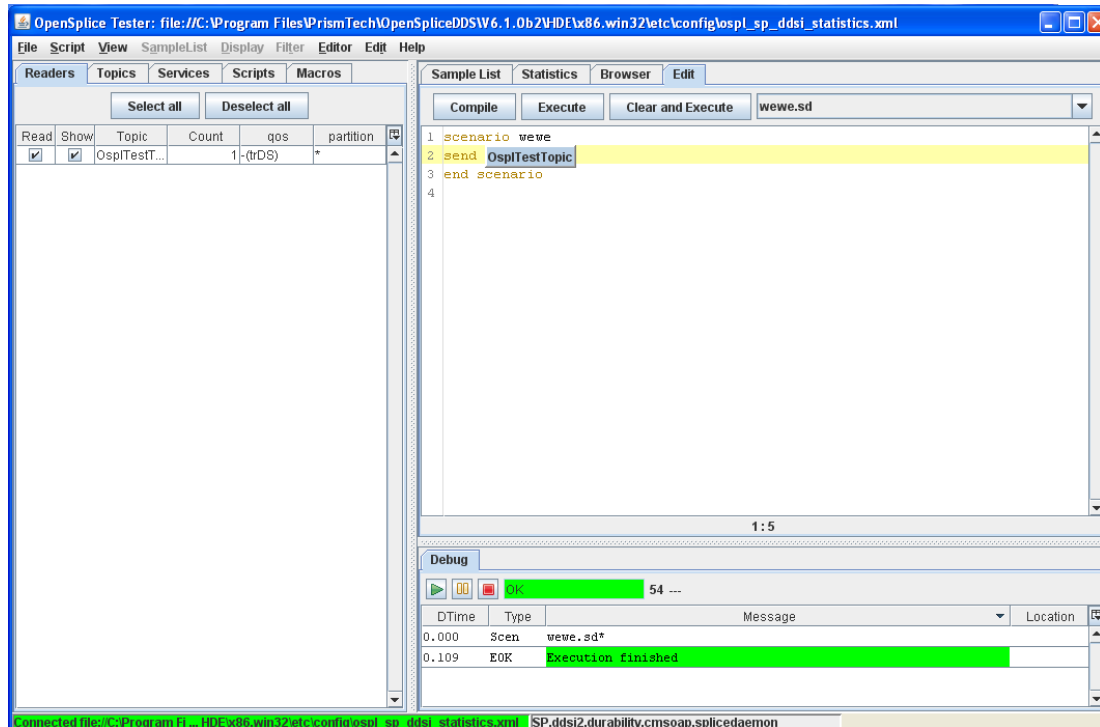
Step 4: Type `check Ospl` then press *[Ctrl+Space]*.

Step 5: 'OsplTestTopic' appears; press *[Enter]* to accept it, and the instruction is completed.

Code completion (check)
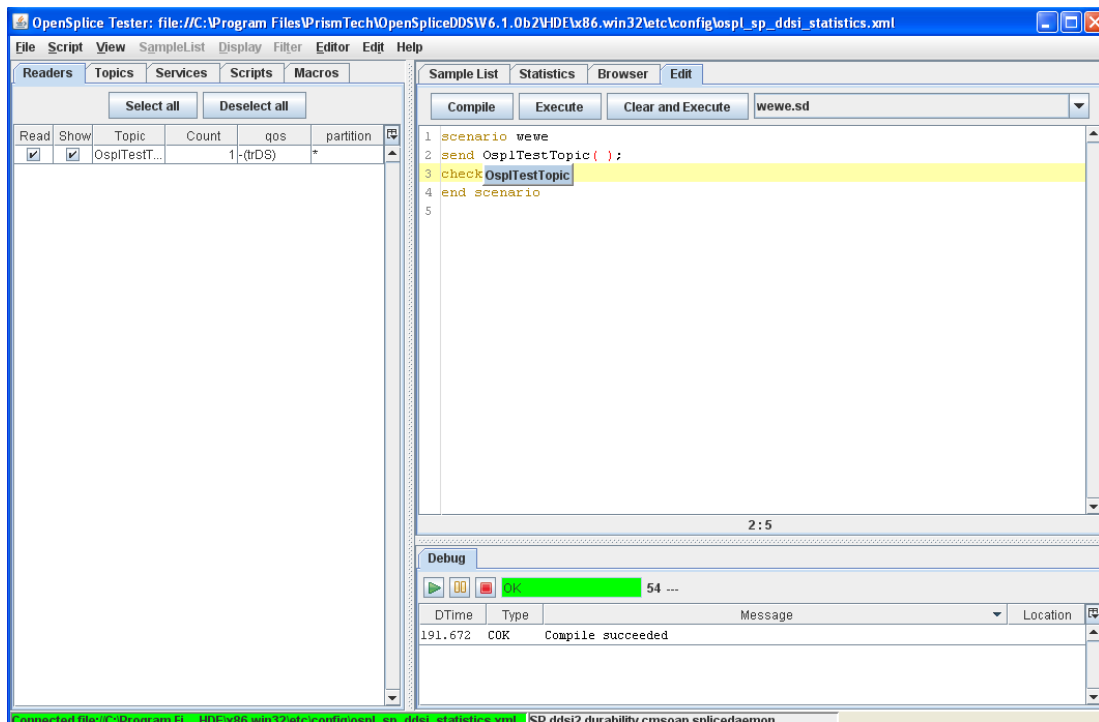
4.10 Execute and Debug

4.10.1 To Run the Current Script

Step 1: Click the *Execute* ('Play' ▶) button in the *Debug* window to run the current script.

Step 2: While the script is still executing, click the 'Pause' ⏸ button in the *Debug* window.

Step 3: While the script is still executing, click the 'Stop' ■ button in the *Debug* window.

Step 4: In the *Debug* window, double-click the entry where the column *Location* has a value of 6. Double-clicking on an entry in the *Debug* window highlights the relevant line in the *Editor* window.

Debugging a script
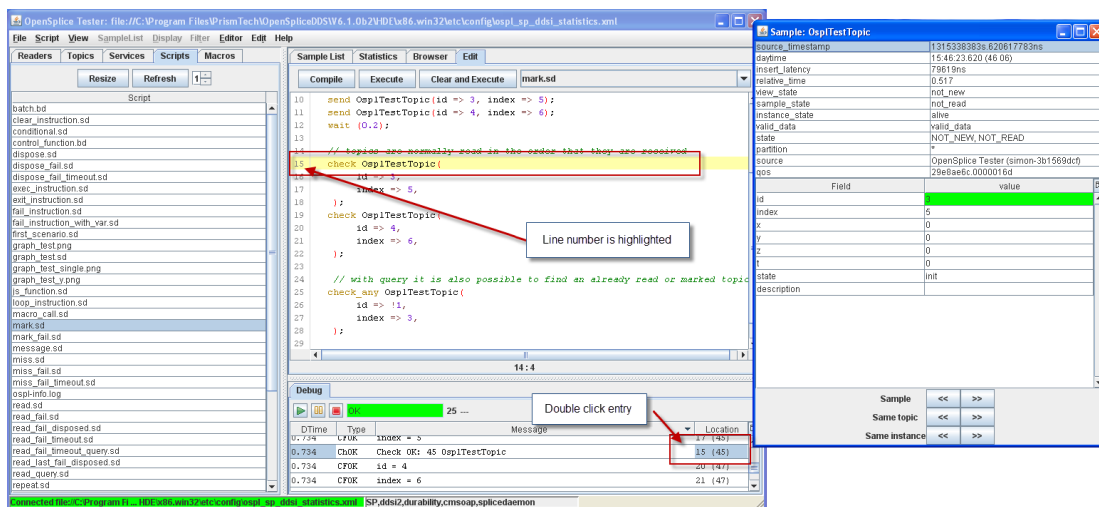
4.10.2 Batch execution (Batch window)

Load and run batch scenario.

Step 1: Choose *Script > Batch* from the top menu.

Step 2: In the *Batch* window, choose *File > Load batch*.

Step 3: Select *batch.bd* in the example script directory.

Step 4: Click the *Start* button.

Batch execution

4.10.3 To Run a Batch Script from the Command Line

Step 1: Change directory to the example scripts directory where the `batch.bd` is found (`<OSPL_HOME>/examples/ ...`).

Step 2: Run `ospltest -e -b batch.bd`.

4.10.4 Batch results

Load batch result

Step 1: Choose *Scripts > Batch results* from the top menu.

Step 2: With the *Batch results* window open, choose *File > Load result* from the top menu.

Step 3: Select the batch result file from the batch run.

Batch results

Script name	batch_20110907	batch_20110907	batch_20110906	batch_20110902	batch_20110902
dispose	OK	OK	OK	OK	OK
variables	OK	OK	OK	OK	OK
loop_instruction	OK	OK	OK	OK	OK
conditional	OK	OK	OK	OK	OK
repeat	OK	OK	OK	OK	OK
repeat_multiple	OK	OK	OK	OK	OK
repeat_until_dispose	OK	OK	OK	OK	OK
message	OK	OK	OK	OK	OK
macro_call	NOK	NOK	NOK	OK	OK
scenario_call	OK	OK	OK	OK	OK
graph_test	NOK	OK	NOK	NOK	OK
js_function	OK	OK		OK	NOK
set_instruction	NOK	NOK		OK	OK
fail_instruction	OK	OK		OK	OK
exec_instruction	OK	OK		OK	OK
fail_instruction_with_var	OK	OK		OK	OK
read_fail	OK	OK		OK	OK
read_fail_timeout	OK	OK		OK	OK

Scan regression folder for batch results

Step 1: Choose *File > Scan Regression* from the top menu.

Step 2: Double-click the test result column of any test.

The results displayed will appear similar to the example in Figure 56.

Scan regression for specified directory

Step 1: Choose *File > Scan Regression dir* from the top menu.

Step 2: Select the directory (folder) that contains batch results.

The results displayed will appear similar to the example in Figure 56.

4.11 Adding virtual fields

'Virtual fields' are fields with calculated values. For example, a translation from radians to degrees, or from cartesian to polar coordinates. The virtual field can be provided in Java (inside a plugin, see Plugins later in this section) or a script language (see the section on *scripting*, as well as the following example).

4.11.1 Add virtual fields to the topic

Step 1: Choose *File > Add fields* from the top menu.

Step 2: Browse to the example directory and select `fields.txt`.

Step 3: Open the *SampleList* window.

Step 4: Select the *OsplTestTopic* sample.

Step 5: Add extra fields from the pop-up menu.

Adding extra fields to a sample
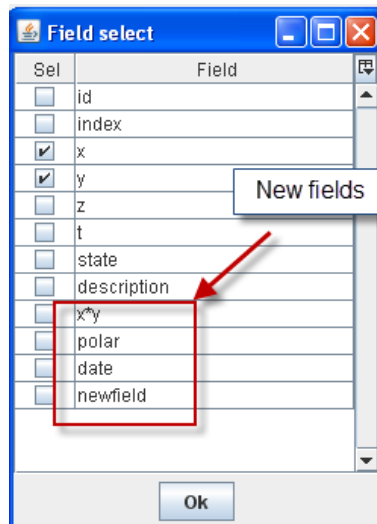
4.12 Plugins

Plugins can extend the functionality of Tester by providing virtual fields (see Adding virtual fields), or additional interfaces. Plugins are automatically loaded upon startup from the specified plugin directory. Two sample plugins are provided with Tester: *SimplePlugin* adds virtual fields, and *TestInterface* adds a UDP/IP message interface (see *Message Interfaces*).

4.12.1 Install / Uninstall plugins

Step 1: Go to the `examples/tools/ospltest/SimplePlugin` directory.

Step 2: Run `ant` from the command console to build the *SimplePlugin* example.

Step 3: Run Tester and choose *File > Preference* from the top menu.

Step 4: In the *Settings* tab, set the correct value for *Plugins dir* and click *OK*.

Setting the path to the Plugins directory
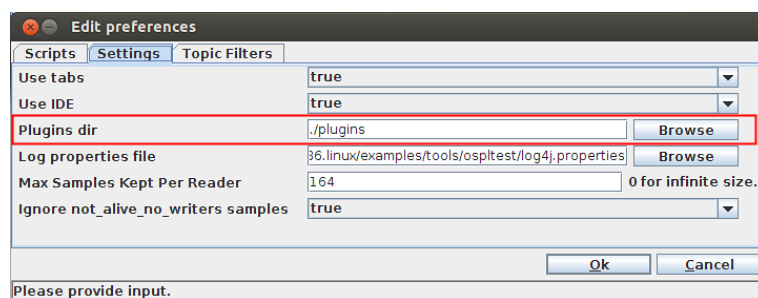
Step 5: Choose *File > Plugins* from the top menu.

Step 6: Click *SimplePlugin* to select it.

The SimplePlugin example

Step 7: Double-click any *OsplTestTopic* data in the *SampleList* window. New fields are added.

Fields added to OsplTestTopic sample
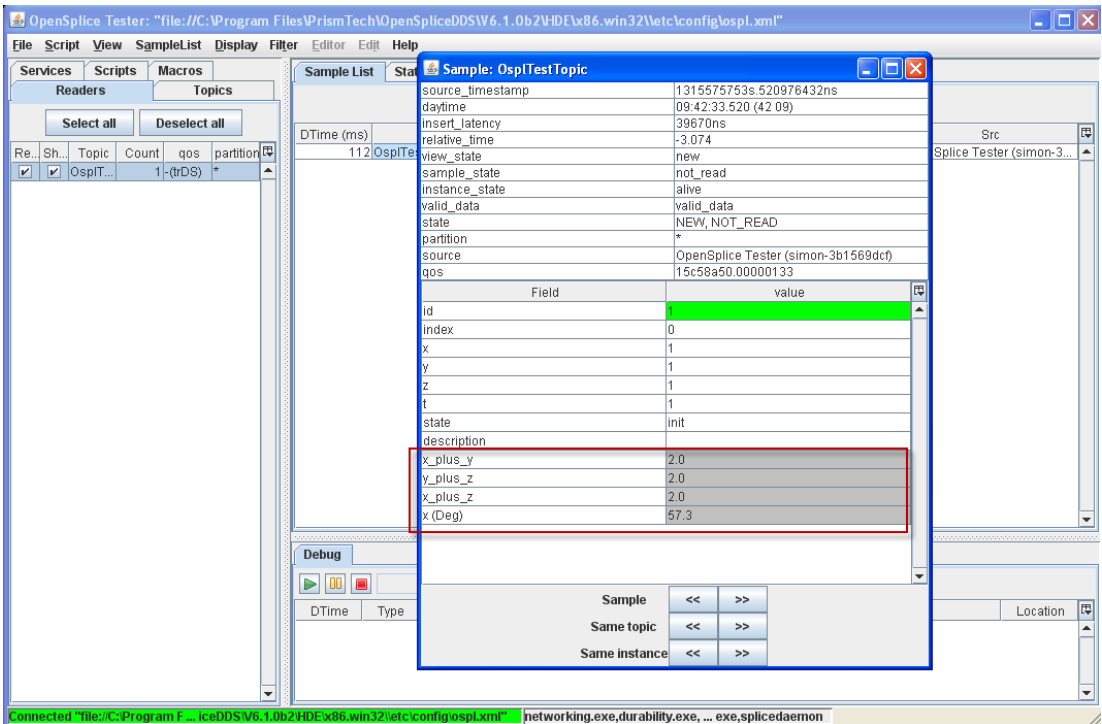
Step 8: In the *Select extra field* dialog (*[F9]*), one more field is added.

Extra field added
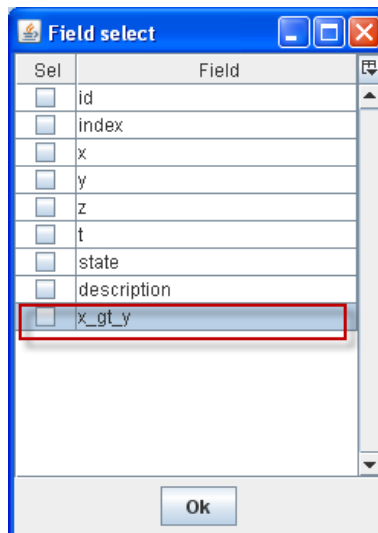
4.13 More on Virtual fields

Additional virtual fields can be provided *via* a plugin or *via* a script.

4.13.1 Adding Virtual Fields *via* plugin

Override the class:

```
ExtraTopicField
```

Compile this class in a plugin and in the 'install' function register the extra fields with:

```
connection.registerExtraField(<instance of extra topic field
class>);
```

An example of a plugin with an extra field is provided in examples:

```
<OSPL_HOME>/examples/tools/ospltest/plugins/SimplePlugin
```

4.13.2 Adding Virtual Fields *via* script

A script file can be loaded using the top menu: *File > Add Fields*.

The script file has the following syntax:

```
[#!<language>]
<name of the field>
<name of the applicable topic>
<script which returns a value and can have multiple lines>
next_field
<name of the field>
<name of the applicable topic>
<script which returns a value and can have multiple lines>
```

The language description is optional. 1 to *n* fields can be described in a single file.

The data of the sample is available in an object variable which is pushed to the script engine before the execution of the script. The object sample provides the following functions:

```
String getDayTime();
long getTime();
long getId();
```

```
String getMsgName();
String getKey();
String getInstanceState();
boolean isALive();
String getSource();
String getFieldValue(String fieldname);
```

These functions can be used to retrieve data from the current sample and determine the value for the extra field. An example of a script file is provided in `examples/tools/ospltest/fields.txt`.

5

Command Reference

This section lists all of the Tester's commands and describes their operation.

5.1 Introduction

The commands are described below in the order in which they appear in the menus (starting at the top left).

Where a menu option also has a keyboard shortcut, it is shown in *[italics in square brackets]* (for example, *[Ctrl+C]*).

Some menu options can also be invoked by clicking on buttons in appropriate tabs or windows.

5.2 Menus

Tester main menu

5.2.1 File

File > Connect, *[Ctrl+Shift+C]* Open a connection to a Domain.

File > Disconnect, *[Ctrl+Shift+D]* Disconnect from a Domain.

File > Remove All Readers Remove all previously-added Readers.

File > Add Reader Add a single topic Reader.

Add Reader dialog
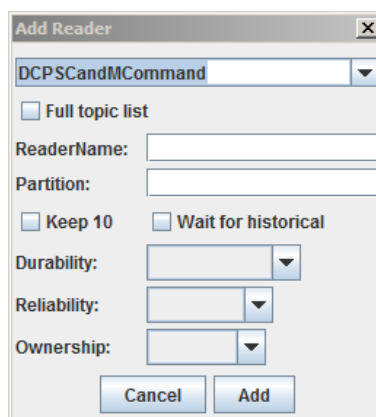
File > Add Readers Add multiple Readers by selecting from the Topic List.

Add Readers from Topic list
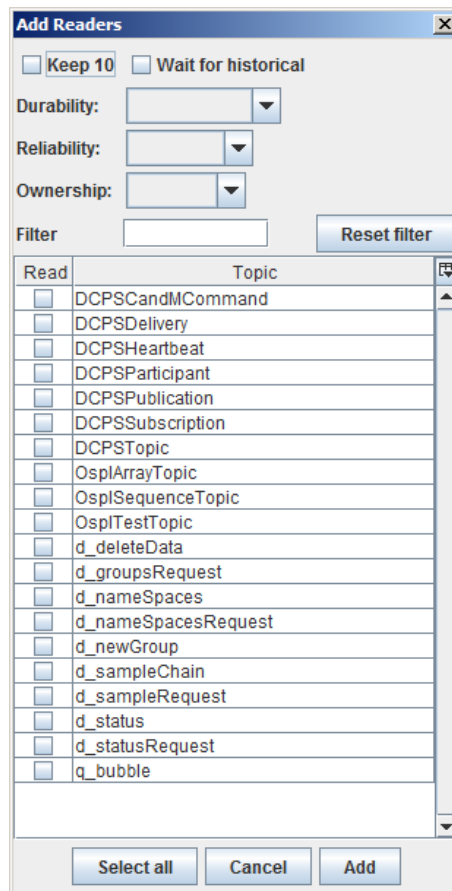
File > Save Readers List

> Save the current list of topics to a file. The keys, QoS, wait for historical info will be preserved.

The format of the readers list file (and the add reader specification) is:

```
<!>[#QOS#]topic_name[|readername][\[partitionname\]]    <optional_key>
<optional_foreign_key1> <optional_foreign_key2> <optional_foreign_key3>
```

File > Load Readers List Load a topics file. Topics already in the list will not be recreated.

File > Add Fields Load new fields. Example `field.txt` is located in the `example` directory.

Load Extra Fields dialog
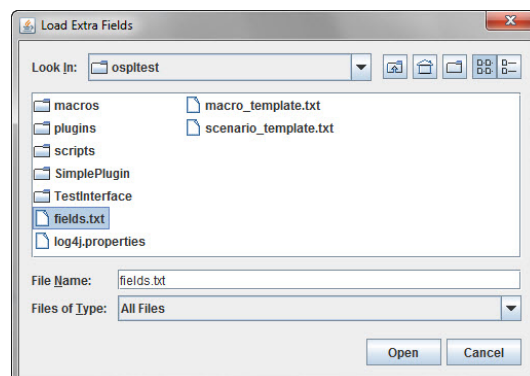
File > Plugins Install/Uninstall Plugins. The example `SimplePlugin` plugin is located in the `example` directory. It must be compiled and put in to the `plugins` directory specified in *Preference* page.

Plugins dialog
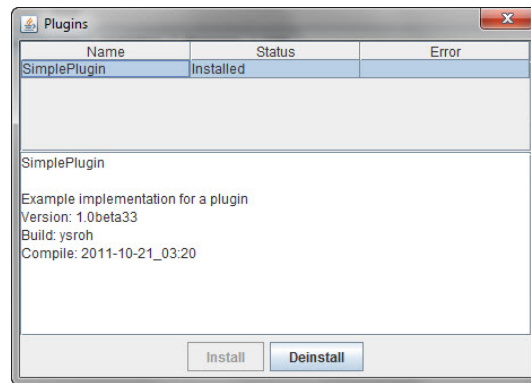
File > Save Layout Save the current layout of the windows in a file; this can later be used to organize the windows in the same way. *Save Layout* is only applicable to non-IDE mode.

File > Load Layout Load a specific layout of the windows as previously saved (select by file on the disk) with *Save Layout*. *Load Layout* is only applicable to non-IDE mode.

File > Preferences Can be used to change the locations of the macros and scripts directories. (See also the instructions for *starting and stopping Tester*).

File > Exit Quit the application.

5.2.2 Script

Script > Script Editor, *[Alt+Shift+S]* Open the script *Edit* window.

Script > Debug Window Open the script *Debug* window.

Script > Scripts Open the scripts window which allows for quick access to scripts found on the script path (as defined in the ospltest.properties). (See also File > Preferences and *Starting and Stopping Tester*.)

Script > Macros Open the macros window which allows for quick access to the macros found in the macro path (as defined in the `ospltest.properties`). (See also File > Preferences and *Starting and Stopping Tester*.)

Script > Batch, *[Alt+Shift+B]* Open the *Batch Execute* window for the batch execution of several scripts.

Script > Batch Results Display the results of the batch run.

5.2.3 View

View > Samples, *[Alt+1]* Open the *Sample List* window.

View > Statistics, *[Alt+2]* Open the *Statistics* window.

View > Browser, *[Alt+3]* Open the *Browser* window.

5.2.4 SampleList

The *Sample List* displays the current list of read samples. The list is sorted on source time (timestamp) of the topic samples. Topics Samples are only displayed when the *Show* checkbox in the Reader list is checked (note that un-checking *Show* does not delete the topics Samples). A double-click in the list results in the topic being displayed in the *Sample* window.

The state displayed with the topic is the Sample state of the sample. When the state of the topic is `alive` then if this is the last Sample with that key it is displayed as `ALIVE_AND_KICKING` for received samples and `ALIVE_AND_SEND` for samples sent by Tester. This makes it very easy to spot topics which are not disposed.

When exactly *two* topics are selected, the difference between the source timestamps is displayed.

The following menus are only active when the *Sample List* tab is selected showing samples. (If you are in the *Browser* tab, for example, then the menus will not be active (they will be 'greyed out')).

SampleList > Clear, *Clear* **button** Clear the Sample List.

SampleList > Dump Dump the contents of the current (filtered) Sample List to a file.

SampleList > Dump Selection, *[P]* **(also** *[Ctrl+P]* **and** *[Alt+P]*) Write the current selection content to a file.

SampleList > Dump to CSV Write the contents in CSV format.

SampleList > Dispose Alive Dispose all topics in the Sample list with a state `alive and kicking` (*i.e.* all last Samples of a topic with a given key which are still alive), this function can be used to clean up (dispose left alive samples) a list after a test.

SampleList > Diff Script Create a list of instructions in the current scenario which reproduces the list of samples in the Sample list. The *diff* means that only fields which do not have the default value or are a key/switch field are used in the script.

SampleList > Diff Script Selection Create a diff script for the current selection of samples.

SampleList > Show Chart, *[Alt+Shift+C]* Display the chart window. To fill the chart with data select a column with numeric values and press *[Y]*. This will add a trace with the values of the column, using the time received on the X axis. Multiple traces can be added. Select a filter to limit to the appropriate values. To display a scatter plot, clear the traces and select the column to use on the x-axis, then press *[X]*. After this select the column with values for the Y axis and press *[Y]*. It is also possible to automatically create multiple traces based on a key value. First select the column to be used as key and press *[K]* before the Y column is selected.

[F2] Compare two topic Samples. Select the first topic Sample in the *Sample* window (by double-clicking), then select the second topic Sample and press *[F2]*. The samples will be displayed side by side with the differences marked in the window of the second topic (normally the left window). A field marked in red is different, a field marked in orange was not found in the first topic Sample. If not different then (foreign) key fields will be marked in green and yellow. (See also *Topic Instance Window*.)

[F3] Display a topic Sample in a separate *Sample* window.

[F4] Open the topic edit window with the values of the selected topic.

[F9] Fields of the current selected topic sample can be added for display in the Sample list. Fields are displayed based on name. Any topic Sample with a field of that name will provide the value of the field. A field column can be deleted by selecting a cell in the column and then pressing *[Delete]*.

5.2.5 Display

When the Sample List is open these commands allow the user to adjust the window display attributes to their needs.

Display > Font Smaller, *[Ctrl+minus]* Decreases the font size of the *Sample List* window.

Display > Font Larger, *[Ctrl+plus]* Increases the font size of the *Sample List* window.

Display > Day Time Toggles the Dtime column format between number of milliseconds (ms) and time-of-day (hh:mm:ss.ms).

Display > Colors Toggles the display of colors (on or off).

Display > Refresh Refreshes the *Sample List* window.

Display > Only Show Alive Filters the samples to display samples in the 'alive' state.

5.2.6 Filter

When the *Sample List* is open these commands enable you to filter the displayed samples based on the Topic and Key attributes of the current sample.

The filter can also be applied by typing the key directly in the filter window. Add a **+** (plus) sign in front of the key value to filter including foreign key relations (it is not possible to filter on key and topic name when entering the key manually). The filter can also be reset by clicking the *Reset* button.

Filter > Topic, *[Ctrl+F5]* Filter on topic name.

Filter > Topic and Key, *[F5]* Filter on key and topic name.

Filter > Key, *[F6]* Filter on key only (so all topics with the same value for key are displayed).

Filter > Resets, *[F7]* Clears the filter.

[F8] Filter on the key value and also allow forward foreign key relations (*i.e.* find topics which have a key which matches a foreign key of an already-displayed topic.

[F12] Filter all messages with the same sample state.

[F] Filter based on text in a column, the column is listed in the filter box (*i.e.* `[<column>]`) add the text on which to filter and then press *[Enter]*.

5.2.7 Editor

When the *Edit* window is open these commands allow the user to create and manage Scenarios and Macros.

Editor > New Scenario, *[Ctrl+N]* Create a new scenario. A *File Save* dialog will be displayed to provide the filename of the scenario. The initial scenario will be created using the template `scenario_template.txt` which is found in the installation directory.

Editor > New Macro, *[Ctrl+M]* Create a new macro. A *File Save* dialog will be displayed to provide the filename of the macro. The initial macro will be created using the template `macro_template.txt` which is found in the installation directory.

Editor > Open, *[Ctrl+O]* Opens the *File Open* dialog, the selected Script or Macro file will be loaded in the editor.

Editor > Save, *[Ctrl+S]* Save the current script to disk (to the same file as it was loaded/created).

Editor > Save As, *[Ctrl+Shift+S]* Opens the *Save* dialog for entering a filename to which the current script will be saved.

Editor > Complete, *[Ctrl+Shift+C]*, *[Ctrl+T]* Completes the Scenario by inserting `"start scenario"` and `"end scenario"` text at the beginning and end of the current file.

5.2.8 Edit

When the *Edit* window is open these commands provide basic text editing capabilities.

Edit > Cut, Edit > Copy, Edit > Paste, Edit > Find/Replace Traditional text editing commands. The standard key combinations (such as *[Ctrl+X]* and *[Ctrl+C]*) are also recognized.

Edit > Format, *[Ctrl+Shift+F]*, *[Ctrl+I]* Automatically formats the text in the current edit window. Formatting removes extra blank lines and normalizes the indentation.

Keyboard-only commands

Some functions are not accessible from the menu bar; these are mostly common editing commands that are invoked with standard ('traditional') key combinations ('shortcuts').

[Crl+A] Select all text in the current field or editor window.

[Ctrl+E] Execute the current scenario.

[Ctrl+Space] Complete the scenario at the current location. If the cursor is on an empty line, the list of possible commands is shown; on a complete command, the appropriate editor for that command is opened (if available).

[Ctrl+Z] Undo the last command.

Macro Recorder

The Tester has a simple macro recorder, intended for ad hoc use, controlled by keyboard commands only. It can record and store a single un-named macro which is only retained for the current session (until the Tester is closed).

[Ctrl+Shift+R] Start recording a new macro. Any previously-recorded macro is deleted.

[Ctrl+Shift+S] Stop recording.

[Ctrl+Shift+M] Play the recorded macro.

5.3 Lists

5.3.1 Services

Displays a list of the Services running on this node. A display-only window.

5.3.2 Scripts

Displays a list of the installed Scripts (`.sd` files) and Batch Scripts (`.bd` files).

Refresh Refreshes the list.

<select> a Script Displays the Script in the *Edit* window

5.3.3 Macros

Displays a list of the installed Macros (`.md` files).

Refresh Refreshes the list.

Scen Checking this option displays Scripts as well as Macros.

<select> a Macro Displays the Macro in the *Edit* window

5.3.4 Readers

For each reader the count of received samples is displayed as well as the QoS and partition. A check box is provided for changing the *read* state or the *show* state. When *Read* is unchecked the reader stops reading samples. When *Show* is unchecked the topic samples of that topic will not be displayed in the sample list.

Select all Checks the *show* state for all topic samples.

Deselect all Unchecks the *show* state for all topic samples.

<select> a Topic Instance Enables you to check/uncheck the *Read* and *Show* state.

<right-click> Delete Reader, *[Delete]* Deletes the selected reader.

<right-click> Recreate Reader, *[Ctrl+R]* Recreates the selected reader and as such re-reads any persistent/transient data available.

<right-click> Show First Sample, *[F3]*, **or double-click on the reader** Shows the first sample for the selected reader.

<right-click> Edit Sample, *[F4]* Opens an *Edit Sample* window for the selected topic.

[F9] Opens the field selection window for the display of fields of the selected topic.

Edit Sample Window

The *Edit Sample* window is used for editing field values of a topic and then writing the sample or dispose the instance. It is also used to insert the topic values as a 'send' or 'check' entry in the current script (at the cursor position in the script window).

The *Edit Sample* window can be filled with a topic from both the *Topics* window and the *Sample List* window with the *[F4]* key. If the topic write window is filled with a topic from the topics list window then the values are all empty (except for union discriminators, which get a default value). If the window is filled from the sample list window then the fields get the values of the selected topic sample in the sample list. The key fields are marked in green and the foreign keys are marked in yellow.

Fields can be edited by selecting the edit field (right-most column). If the field is of an enumerated type then a combo box is displayed which provides all possible values. The topmost value is empty for reset to the default value (not set).

The keyboard can be used to navigate the edit fields. The cursor *[Up]* and *[Down]* (arrow) keys move between fields; any other key starts editing the value in the current field.

Edit sample window

(There is a second form of this window, used when opened from the script with *[Ctrl+Space]*, *[Ctrl+Left-click]*, or as part of completion. It only has two buttons: *OK* and *Cancel*. Pressing *[Ctrl+Enter]* or *[Ctrl+Return]* is the same as clicking *OK*.)

write Write the sample.

writeDispose Write the sample and Dispose the instance.

dispose Dispose the instance.

script Instead of writing the sample this creates the script commands to write the sample. These commands are inserted into the scenario currently being edited and the user will be taken to this text.

check Similar to *script* but creates the script command to check the sample values.

[F4] Copy the current selected field from the topic in the instance window.

[F5] Copy all fields based on an equal name from the topic in the instance window.

[F6] Fill all fields with .sec in the name with the current time seconds and fields with .nanosec in the name with the current time in nanoseconds.

[Ctrl+T] Fills a field of type int with the seconds part of the current time.

[Ctrl+U] Fills a field of type long with a unique key.

[Ctrl+V] Paste a value.

[Alt+Down] Opens the *enum* editor.

[Enter], *[Return]* Commits the current edited value.

[Esc] Discards the current edited value.

Once the desired values have been entered the topic can be written by clicking the *Write* button, disposed by clicking the *Dispose* button, or write disposed by clicking the *WriteDispose* button.

5.3.5 Topics

The topics list displays the list of topics as known in the system.

<select> a Topic Selects a Topic.

<right-click> Create Reader Create a Reader for the selected Topic.

<right-click> Create Default Reader Makes the selected Reader the default reader to be displayed in the *Samples List*.

[F2] The key list definition window will open which allows to change the (foreign) keys. The syntax is the same as in the add topic window or topic file. To support the selection of the keys the primary fields of the topic are displayed and will be inserted at the cursor position in the edit field when clicked.

5.3.6 Groups

The Groups list displays the list of groups created Tester and currently active. The Groups tab is only visible in the main window when at least one Group has been created in the current Tester session.

<select> a Group Selects a Group.

<right-click> Delete Group, *[Delete]* Delete the selected Group.

<right-click> Publish Coherent Sets, *[F4]* Create a Coherent publisher window from which coherent sets of data can be created and written.

5.4 Windows

5.4.1 Sample List Window

The *Sample List* window is used to display samples. By default the delta time, topic name, state, key, and source are displayed. Additional columns can be added and filters defined.

<div align="center">

Sample List window

</div>

Clear Clears the list.

Filter <value> The current filter value.

Reset Resets the filter value.

Pack Adjusts the displayed column widths.

<select> a Sample Selects a sample to use with *<right-click>* commands. *[Ctrl+Left-click]* selects another sample. If exactly *two* samples are selected, the difference in source time will be displayed in the top bar of the *Sample List* window.

<right-click>Select Extra Fields, *[F9]* Opens a dialog box allowing selection of extra fields to display.

<right-click> Display Sample, <double-click> Displays sample details.

<right-click> Display Sample New Window, *[F3]* Displays sample details in new window.

<right-click> Compare Sample, *[F2]* Compares two samples with each other and shows differences in red color.

<right-click> Edit Sample, *[F4]* Allows Tester to edit the selected sample values.

<right-click> Filter on topic, *[Ctrl+F5]* Filters on the selected topic value.

<right-click> Filter on topic and key, *[F5]* Filters on both the selected topic and key values.

<right-click> Filter on State, *[F12]* Filters on the State of the selected sample.

<right-click> Filter of Key, *[F6]* Filters on the Key value of the selected sample.

<right-click> Filter on Column Text, *[F]* Sets the filter to be the value of the current column.

<right-click> Filter Reset, *[F7]* Resets the filter value.

<right-click> Delete extra column, *[Del]* Removes the selected extra column from the list.

<right-click> Add Column as Key to Chart, *[K]* Assigns the selected column as the key field for the chart.

<right-click> Add Column as X to Chart, *[X]* Assigns the selected column as the x-axis for the chart.

<right-click> Add Column as Y to Chart, *[Y]* Assigns the selected column as the y-axis for the chart.

[Ctrl+F] Finds the next sample containing the search text in any column.

5.4.2 Statistics Window

The *Statistics* window provides statistics for the topics in use, such as write count, number of alive topics, *etc.*. The following values are displayed for each topic:

Count The number of samples currently in the OpenSplice database

Arrived The number of arrived samples

Takes The number of takes by the reader

Reads The number of reads by the reader

Alive The number of alive topics (instances not disposed)

Writes The number of written samples

The left table shows either the participants, the topics, or the statistics of the currently-selected reader/writer as indicated by the selected tab.

When the list of participants is shown, a participant can be selected. The second table shows the list of readers with their statistics, the third table show the list of writers with their statistics.

When the list of topics is shown, a topic can be selected. The second table shows the list of participants reading the topics with their statistics, the third table shows the list of participants writing the topic with their statistics.

If a value of -1 or -2 is shown then an error occurred during the retrieval of the statistics for the reader/writer.

By selecting a row in the reader or writer list all statistics for that reader or writer will be shown in *Stats* tab of the left window.

Refresh Will refresh the content.

Add readers Will add the topics in the reader list to the list of monitored topics.

Add writers Will add the topics in the writer list to the list of monitored topics.

[CTRL+F] Finds the next reader/writer containing the search text in any column.

5.4.3 Browser Window

The *Browser* window enables you to view the Readers and Writers in the system. You may browse by Node, Participant, or Topic.

Browser window
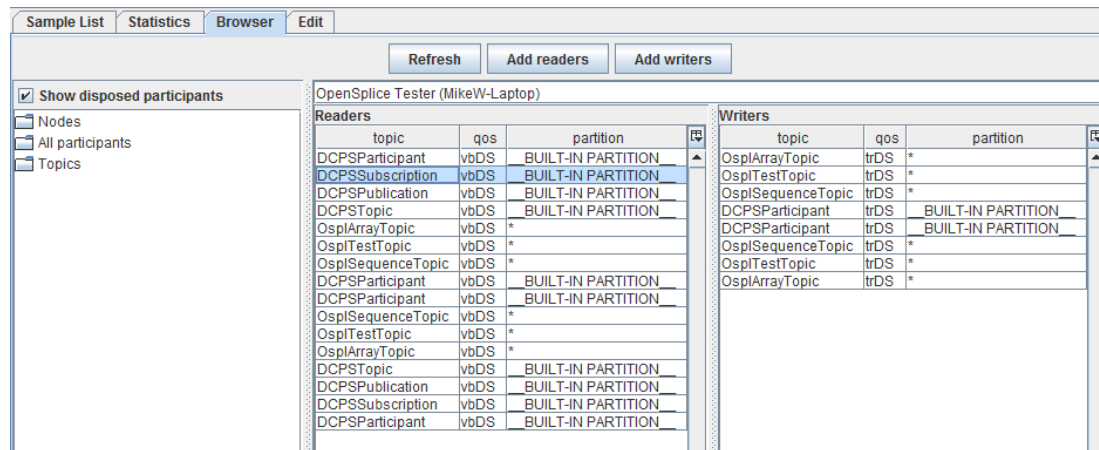
Refresh Will refresh the browser content.

Add readers Will create a Tester reader from the list of readers for the selected read-topic. The QoS of the discovered reader will be used to ensure that data read by that reader will be captured in the timeline.

Add writers Will create a Tester reader from the list of writers for the selected written-topic. The QoS of the discovered writer will be used to ensure that data written by that writer will be captured in the timeline.

Show disposed participants Used to toggle the display of disposed participants.

[CTRL+F] Finds the next reader/writer containing the search text in any column.

5.4.4 Edit Window

The *Edit* window is used to create and modify Scripts and Macros. Please refer to Chapter 5, Scripting, on page 73, for more details.

Edit window
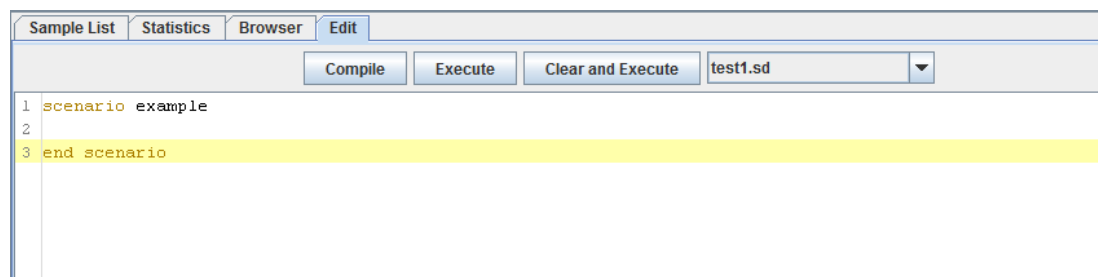
Traditional text editing commands and standard key combinations (such as *[Ctrl+X]* and *[Ctrl+C]*) are recognized. Menu commands and keyboard shortcuts for editing scripts and macros are described in sections 4.2.7, Editor, 4.2.8, Edit, 4.3.2, Scripts, and 4.3.3, Macros.

When editing macros, instruction-specific editing dialogs may open; for example, the `send`, `check` and `execute` macro instructions have their own editing dialogs which help to make your entries conform to their syntax.

Editor for execute instruction
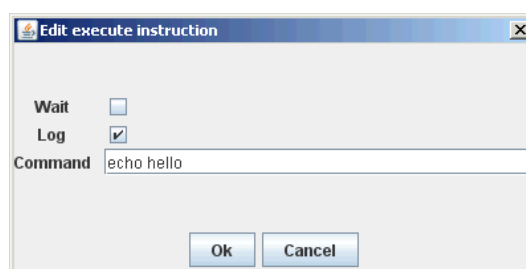
Compile Compile the current content.

Execute Run the current script or macro without clearing the sample list.
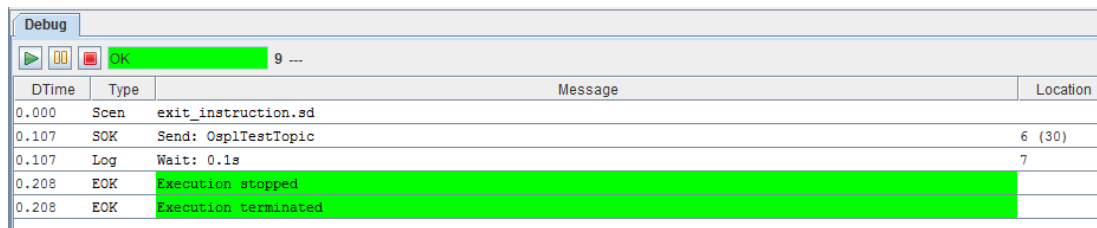
Clear and Execute Clears the sample list and then runs the current script/macro and returns the user to the *Sample List* window.

<drop down> Allows for quick selection of recently-edited scripts/macros.

5.4.5 Debug Window

The *Debug* window is used for tracing/debugging Script compilation and execution. For each step, the day/time, type of message, and message text is displayed along with the location (line number) in the scenario.

Debug window

Control execution of the scenario with the buttons at the top left of the window:

Start (Play) Start or resume execution

Pause Pause execution

Stop Stop (halt) execution

[CTRL+F] Finds the next message containing the search text in any column.

6

Scripting

The Vortex OpenSplice Tester provides automatic testing capabilities by means of scripting. This section describes the features of Tester's built-in scripting instructions, and how to install additional script engines.

6.1 The Script Language

The script language as used by the Tester is specifically designed to create readable and easily maintainable scripts.

Instructions are simple, with named parameters which enable the Tester to limit the testing to the fields applicable to the test. For example, the `send` instruction is an instruction which sends a topic. The basic syntax is the keyword `send` followed by the `topicname` and a list of named parameters between parentheses ('round brackets'), terminated with a semicolon.

Illustrating send keyword syntax

```
send OsplTestTopic(
    id => 1,
    x => 2,
    y => 3,
    z => 4,
    t => 1,
    state => boost,
    description => "hello",
);
```

The `check` instruction is similar to the `send` instruction; it has options to find a specific instance using key fields or a query.

Illustrating check keyword syntax

```
check OsplTestTopic(
    timeout => 0.2,
    id => 1,
    index => 0,
    x => 2,
    y => 3,
    z => 4,
    t => 1,
    state => boost,
    description => "hello",
);
```

In this example a `timeout` is set, which will allow a wait of up to 0.2 seconds for the topic sample for the correct instance to arrive.

6.1.1 A script file

A `scenario` has the following format:

Illustrating scenario keyword syntax

```
--Project      : Project www.opensplice.org

scenario <name>

    <instructions>

end scenario
```

The name is for information only, and is not used further.

6.1.2 Variables

The script language allows the use of variables. Variables can be used to store values that can then be used at a later time. A variable is indicated by either a << or a >> prefix. Variables may be declared implicitly, or explicitly using the `var` instruction.

Example variable

```
// a variable can be declared
var myvar => 5;

// and then used in an instruction
send OsplTestTopic(id => 1, index => <<myvar);
```

In this example the variable `myvar` is declared and initialized with the value 5. Within the `send` instruction the variable is used to provide the value for the field index. The << prefix indicates the direction of the assignment from the variable to the field.

Variable with >> prefix

```
check OsplTestTopic(
      id => !4,
      index => >>index_of_4,
   );
```

Here the variable `index_of_4` is declared implicitly and the value of the field index is copied to the variable (the prefix >> points to the variable).

All environment variables and java virtual machine (JVM) properties are also available as variables, and they can be used as shown below:

Using environment variables

```
log ("message: " <<OSPL_HOME );
log ("message: " <<os.name);
```

Special variables

There are some special variables which can be useful in scripts.

- `curtime_sec` and `curtime_nsec` provide the second and nanosecond parts of the current time.
- `uniqid` provides a unique number for every call, within the same session of the Tester.

- `script_file` and `script_path` provide the scenario file name and the scenario file's path respectively. If the currently executing scenario context changes because of a `call` instruction to another scenario or macro file, using these variables in the called scenario or macro will reflect the respective path and file name of the called script.

Note that these special values are used *without* the << prefix.

6.1.3 Embedded Scripts

Inside a scenario any script compatible with the java ScriptEngineFactory can be used to provide calculated values for fields in a `send`, `check` or `var` instruction, or as a stand-alone statement.

Embedded javascript

```
`
delay = 10.0;
freq = Math.PI * 2 * 0.1;

function get_delay() {
  return delay;
}

function get_x_coordinate(t){
  return Math.sin(get_delay() + t * freq);
}

function get_y_coordinate(t){
  return Math.cos(get_delay() + t * freq);
}
`;

var js => `get_delay(0);`;

repeat OsplTestTopic 0.2 51(
    id=> 2,
    x=>`get_x_coordinate(<<dt)`,
    y =>`get_y_coordinate(<<dt)`
);
```

Stand-alone scripts must be enclosed by left single quotes, and then followed by a semi-colon.

Variables used in the javascript are translated before the evaluation of the script. In this specific case the <<dt is the delta time in the `repeat` function. All javascript in one scenario is executed in the same scope, and functions and variables declared at the beginning of a script are available later in the script.

A specific script language can be selected by providing the name of the script language in the first line of the embedded script: `#!<language>`, for example `#!js`. Note that the language description must not be followed by any other text. See section 5.6, Installing Script Engines, on page 85, for instructions on installing a scripting language for use with the OpenSplice Tester. If no language descriptor is provided on the first line of a script, the default language is used as set in *Preferences*.

More embedded javascript

```
 var js => `
delay = 10.0;
freq = Math.PI * 2 * 0.1;

function get_delay() {
  return delay;
}
function get_x_coordinate(t){
  return Math.sin(get_delay() + t * freq);
}
function get_y_coordinate(t){
  return Math.cos(get_delay() + t * freq);
}
get_delay(0);
`;

 repeat OsplTestTopic 0.2 51(
    id=> 2,
    x=>`get_x_coordinate(<<dt)`,
    y =>`get_y_coordinate(<<dt)`
 );
```

6.1.4 Comments

Comments can have the following formats:

Format of comments

```
// Comments can have the single line C style format

-- Or the single line ADA format

/*
 * Or the multiline C style format
 *
 */
```

Within the scenario editor, comments are displayed in green.

6.1.5 Macros

For repeated scenarios a repetitive part can be split off into a separate script file called a *macro*. Macros can have parameters.

Calling a macro with parameters

```
// call with default t
call send_and_check_test ( id => 1,   x => 3.1 );
call send_and_check_test ( id => 3,   x => 6.1 );
// call with specific t
call send_and_check_test ( id => 5,   x => 3.2, t => 3 );
```

Similarly to `send` and `check` instructions, values for fields can be optional. However, in a macro a default value *must* be provided for a parameter to be optional.

Setting a default value for a macro parameter

```
macro send_and_check_test (
    id : int;
    x : double;
    t : int := 5;
)
```

In this case `t` is optional, `id` and `x` are mandatory.

It is possible to call a scenario using the `call` instruction. Scenarios do not have parameters.

6.2 The Instructions

6.2.1 Send

Instruction to publish a sample of a topic.

```
send <readername> ( [fieldname => value,]*);
```

The `send` instruction may include an 'update parameter'. The `update` parameter name is a combination of the topic name, followed by an underscore (_) and the literal text 'update'. Valid values for the update parameter are `true` and `false`. The `update` parameter allows scenario scripts to send a series of samples that evolve, each from the previous one, without having to explicitly specify all field values in each `send` instruction.

If the `update` parameter is present, then the sample data sent is retained by the topic reader, associated with the topic key field value(s). A subsequent `send` instruction, including the `update` parameter set to `true` with the same topic key field value(s), will have its sample initialized from the retained values, if they exist. The sent sample with then replace the retained values.

A `send` instruction including the `update` parameter set to `false` with the same topic key field value(s) will be initialized from the topic defaults, but the sent sample will be retained.

A `send` instruction without the `update` parameter will be initialized from the topic defaults, and any retained value for the key field value(s) will be removed.

Disposing the topic reader will remove all retained values.

6.2.2 Dispose

Instruction to dispose an instance of a topic.

```
dispose <readername> ( [fieldname => value,]*);
```

6.2.3 Writedispose

Instruction to write dispose an instance of a topic.

```
writedispose <readername> ( [fieldname => value,]*);
```

6.2.4 Check

Instruction to check a sample of a topic.

```
check[_last | _any] | recheck_last <readername> ( [timeout =>
<timeout in seconds>,] [<fieldname> => [!]<value>[:deviation],]*);
```

A timeout value can be provided allowing the check to wait for `<timeout in seconds>` for the sample to arrive. If a sample meeting the criteria of the check is available either directly or within `timeout` seconds the fields as provided in the parameter list will be verified for correctness.

When the value of a field is an output variable:

```
>><varname>
```

Then the value will not be checked but entered in the variable with the name `<varname>`.

There are two special fields, `topicReceived` and `topicDisposed`, which when used will provide a `true` or `false` value into a variable.

When no sample is found which meets the criteria of the check then `topicReceived` will be set to `false` (and the check instruction will not fail); if a sample is received the value will be set to `true`. When a field `topicDisposed` is found, then the variable will be set to `true` if the sample was disposed and `false` if the sample was not disposed. In this case no fail is reported upon a check instruction when the checked sample was disposed.

The value can be given a possible deviation in the form `<value>:<allowed deviation>`. In this case when the value for the field in the received sample is within the range from `value` *minus* `allowed_deviation` to `value` *plus* `allowed_deviation`, the value is considered correct.

The sample which matches the check can be determined in several ways:

1. The topic does not have a keyfield(s) or the topic has keyfield(s) but no value is provided for all keyfield(s). In this case the oldest not checked or marked sample is checked.

2. The topic has keyfield(s) and the check provides a value for all keyfield(s). In this case the last sample with the key is checked, so long as it was not previously checked. If no matching sample (within the possible timeout) is found then the check fails.

3. One or more fields of the check are marked as a query by prefixing the value with a '`!`'. The oldest not checked or marked sample which matches the query is checked. If no matching sample is found (within the possible timeout) the check fails.

4. Instead of `check`, the command `check_last` is used. In this case (as for situations 1 and 3) the last not-previously-checked sample matching the criteria is checked.

5. Instead of `check`, the command `check_any` is used. In this case also previously-checked or marked samples are considered.

6. The command `recheck_last` will always check the last sample matching the criteria, regardless of whether it has previously been checked or not.

6.2.5 Miss

Instruction to check that no sample of a topic was received since the last checked or marked sample for the given key/query. The same rules apply as for the `check` instruction with respect to finding (or not) the matching topic sample.

```
miss <topicname> ([timeout => <timeout_in_seconds>,] [<fieldname>
=> [!]<value>[:<deviation>],]*);
```

6.2.6 Disposed

Instruction to check that an instance of a topic is disposed for the given key/query. The same rules apply as for the `check` instruction with respect to finding the disposed instance. Note that field values are only provided to find a specific instance (either by key or by query) and not verified for values as part of this instruction.

```
disposed <topicname> ([timeout => <timeout_in_seconds>,]
[<fieldname> => [!]<value>,]*);
```

6.2.7 Mark

Mark all samples (with the given key/query) as read. Any regular miss/check function will not 'see' topic samples received before the mark instruction. If no key or query is provided all samples will be marked as read (and therefore not considered for `check` or `check_last` instructions). If a key value or query is provided, all samples matching the key/query will be marked as read.

```
mark <topicname> ( [fieldname => value,]*);
```

6.2.8 Repeat

Instruction to repeatedly send a topic for a specified count or until disposed.

```
repeat <topicname> <period> <count> ( [fieldname => value,]*);
```

If `<count>` is '0' then the repeat will continue until the scenario terminates or until a dispose for the same topic and key. The variable `dt` is available for calculating a field value based on time since the repeat was started. The period indicates the period with which the topic will be sent. Note that a repeat command by itself does not extend the execution of a scenario and that when a scenario finishes (*i.e.* all following instructions are executed) the repeat instruction is terminated automatically. In such a case the wait or message instruction can be used to ensure that the repeat instruction is completed.

6.2.9 Set

The `set` instruction allows the call of a macro in a table-like fashion. The command allows a number of static parameters and variable parameters. The command has the following format:

```
set <macroname> ([<fieldname>=><value>]*)((<fieldname>*),[(<value>*),]*);
```

For example, the following set instruction:

```
set send_and_check_test (
    t => 2)
((      x,id),
  (   3.1, 1),
  (  2.34, 2),
  ( 3.678, 3),
  (  6.34, 4),
  ( 99.99, 5))
```

In this example the `send_and_check_test` macro is called five times, all five calls will be made with `t = 2` and the values for `x` and `id` as indicated by each row of values. This can be very useful for testing of translations.

6.2.10 Execute

The `execute` instruction allows the execution of an application or command line script on the native OS.

```
execute [wait] [log] "<instruction>";
```

If `wait` is set then the instruction will wait for the execute to complete. If `log` is set then the output of the execute will be logged to the *Debug* window (and resulting dump file). When `log` is used `wait` should also be used, to avoid overwriting log messages.

6.2.11 Log

The `log` instruction logs a message to the *Debug* window. Log messages can provide information immediately (*e.g.* a step being made in a script, or a value of some variable) or post-execution as part of the logfile which includes the full content of the *Debug* window.

```
log ("message" [optional var]);
```

6.2.12 Message

The `message` instruction opens a dialog with the message and allows the operator to provide feedback and a OK/NOK indication. The feedback plus OK/NOK indication are logged to the *Debug* window.

```
message ("message text" [optional var]);
```

This instruction is useful for semi-automatic testing of user interfaces where the GUI part is done manually using message instructions.

6.2.13 Fail

The `fail` instruction fails the execution of the scenario (final result). The execution terminates.

```
fail ("message" [optional var]);
```

The `fail` instruction can be useful in combination with an if instruction, for instance when a complex check is executed using javascript.

6.2.14 Call

The `call` instruction calls a macro or scenario. The name of the macro/scenario is the filename without extension. Macros must be on the `macropath` as provided in the configuration file. The *Macrolist* window displays all available macros. Also note that the macro name **must** be unique throughout all of the available macros because the path is not part of the selection of a macro (just the filename without extension).

```
call <macroname> ([<parametername> => <value>,]*);
```

6.2.15 Reader

The `reader` instruction allows the creation or deletion of a reader. When the keyword `dispose` is used the reader (if it exists for that topic) will be deleted. When a reader is created the `topicname` is mandatory.

```
reader [dispose] (<topicname> [, <qos> [,<partition>
[,<readername>]]]);
```

The qos can be provided in short notation (2 or 4 characters):

```
< v | l | t | p >< b | r >[h][<S|E><D|S>
```

where

< v | l | t | p > Volatile, local transient, transient or persistent

< b | r > Best effort or reliable

[h] History, for a "keep" of 10 which allows for the reception of 10 samples with the same key in one poll interval

<S|E> Shared or exclusive ownership

<D|S> Ordering based on Destination or Source time stamp

6.3 Instructions for Graphs

6.3.1 Graph

The `graph` instruction allows manipulation or save of the graph. It has the following parameters:

```
X
Y
Key
Color
Title
xUnits
yUnits
save => <name>
show => true|false
reset => true|false
```

Note that all graphs have the same X component; when omitted the X will be the sample time. If the Y parameter is set, then a new trace is created for the current graph. The X, key, color, title and units are used for this trace if provided.

If `reset` is `true`, then the graph is cleared (*i.e.* all existing traces are deleted) before creating any new trace. If `show` is `true` then the graph is made visible after adding the trace; when `false`, then the graph is hidden after adding the trace. When `save` is `true` the graph will be saved to an image file after the trace has been added.

6.3.2 Column

The `column` instruction allows the creation of an extra column from a script for use by the graph instruction.

```
column [clear] (<fieldname> [, <columnname>]);
```

When the optional `clear` is set then the column for the field with name `fieldname` will be removed. When `columnname` is omitted, the `columnname` will be the same as the fieldname.

6.4 Instructions for Flow Control

6.4.1 Wait

The `wait` instruction forces a wait in the execution of the script. The time is provided in seconds.

```
wait (<time in seconds>);
```

Value can be a variable.

6.4.2 If

The `if` instruction allows conditional execution of instructions.

```
If (val1 <operator> val2) then

    <true instruction list>

[else

    <false instruction list>]

endif;
```

Where `<Operator>` is one of '==', '!=', '>', '<', '>=', '<=', '||', '&&'.

Expressions can be layered with brackets:

```
((<<x>0) && (<<y>0))
```

6.4.3 For

The `for` instruction allows the execution of a list of instructions multiple times.

```
for ,<var> in 1 ..  10 loop

    <instruction list which can use <<var>

endloop;
```

or

```
for <var> in (a,b,c) loop

    <instruction list which can use <<var>

endloop;
```

6.4.4 Exit

The `exit` instruction exits the scenario.

```
exit;
```

6.5 Instructions for the Message Interface

6.5.1 Write

The `write` instruction writes a message to the interface.

```
write <interface>.<message> ([<fieldname> => <value>,]*);
```

6.5.2 Read

The `read` instruction checks a received message from the interface.

```
read <interface>.<message> ([<fieldname> => <value>,]*);
```

6.5.3 Connect

The `connect` instruction calls the `connect` of the interface. The functionality depends on the implementation in the interface.

```
connect <interface>;
```

6.5.4 Disconnect

The `disconnect` instruction calls the `disconnect` of the interface. The functionality depends on the implementation in the interface.

```
disconnect <interface>;
```

6.5.5 Control

The `control` instruction allows the execution of special instructions as provided by the interface.

```
control <interface>.<instruction>[ ([<fieldname> => <value>,]*)];
```

6.6 Installing Script Engines

In order to use additional script languages the appropriate script engines must be added to the Java `classpath`. The Java JRE already comes with a JavaScript engine by default (*i.e.* no specific installation is required). More Java script engines are available and can be used to support different scripting languages for the embedded scripts inside the scenario scripts, or for the additional fields.

When Tester starts, the available script engines will be logged (default log file is `/tmp/OSPLTEST.log`).

6.6.1 Jython

Download and install Jython on the target machine. Include `jython.jar`, which is normally located in the Jython installation directory, in the `classpath` of the OpenSplice Tester. Use this language by adding the following line as the first line of each script using the Jython script language:

```
#!jython
```

6.6.2 Jruby

Download and install Jruby on the target machine. Include `jruby.jar`, which is normally located in the `lib` directory in the Jruby installation, in the `classpath` of the OpenSplice Tester. Use this language by adding the following line as the first line of each script using the Jruby script language:

```
#!jruby
```

6.6.3 Groovy

Download and install Groovy on the target machine. Include `groovy-all-<version>.jar`, which is normally located in the `embeddable` directory in the Groovy installation, in the `classpath` of the OpenSplice Tester. Use this language by adding the following line as the first line of each script using the Groovy script language:

```
#!groovy
```

7

Message Interfaces

This section describes how to test applications with non-DDS interfaces.

7.1 Message interfaces

An important feature of the OpenSplice Tester is the support of additional interfaces. When an application under test only has a DDS interface it is probably easy to test automatically by stimulating it from the Vortex OpenSplice Tester with samples and verifying the samples produced by the application under test. When the application under test has a GUI component, the message instruction can be used to perform a semi-automated test where the Vortex OpenSplice Tester performs manual control of the GUI and/or performs visual inspections of the GUI (as instructed in the message instruction).

When an application under test has a non-DDS interface, then the message interface of Vortex OpenSplice Tester can be used. There are a number of constraints on the use of a message interface:

- The interface must consist of a limited number of message types which can be described by a static set of fields with static types.

- It must be possible upon reception of a message over the interface, to determine a message type, and from the message type to interpret the message and determine the value for each field of the message.

If these requirements are met, a message interface can be developed for a specific interface of an application under test. This will allow automated testing where messages are written to the test interface, the message received from the test interface will be added to the sample list and it can be checked in the same manner as DDS samples.

7.2 Getting Started with a Message Interface

The best way to get started with a message interface is to compile and use the `TestInterface`. The `TestInterface` is an example message interface which uses a TCP/IP connection and sends a memory-mapped message with a static structure over this interface. The source for the `TestInterface` is provided and it can be found here:

> `<OSPL_HOME>/examples/tools/ospltest/TestInterface`

To compile the `TestInterface`, `ant` and a JDK1.6 must be installed. To build the `TestInterface`, execute `ant` in the `TestInterface` directory. This will compile the `testinterface` and install the resulting plugin in:

> `<OSPL_HOME>/examples/tools/ospltest/plugins`

To run with the plugin, make sure the plugin path points to this directory. The plugin path can be set in *Preferences*:

Setting the plugins path in Preferences
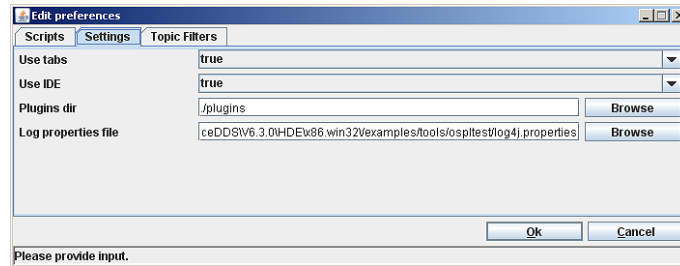
If the plugins directory is changed, the Tester needs to be restarted. Once restarted, make sure that OpenSplice is running (the `TestInterface` registers a topic which will fail is DDS is not running upon startup).

Now two instances of the `testinterface` should show up in the left tab pane (or in separate windows if `Use Tabs` is `false`). Similar to the *Readers* pane, the table will show the available messages and the number of received messages per message type. Since there is no application under test, the `testinterface` is instantiated twice and connected back-to-back. As a result, a message written to the instance "`tst1`" will be received on the instance "`tst2`" and *vice versa*. Also the `testinterface` has created a topic, `OsplTestLogTopic`, and the test interface will write a sample of this topic for each write and read with the content of the message in hexadecimal format.

Messages received on instance tst1
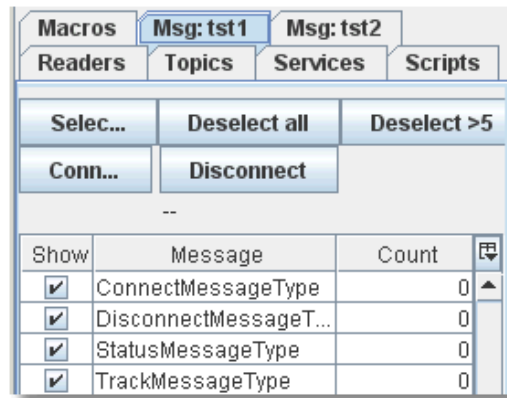
Now select the `test_interface.sd` script, which can be found in `examples/tools/ospltest/scripts`:

The script test_interface.sd

In the script we can see that, similar to the `send` and `check` instruction, the `write` and `read` instructions are used to write a message to the test interface, or read (check) a message received on the test interface.

Execute the script:

Script test_interface.sd running
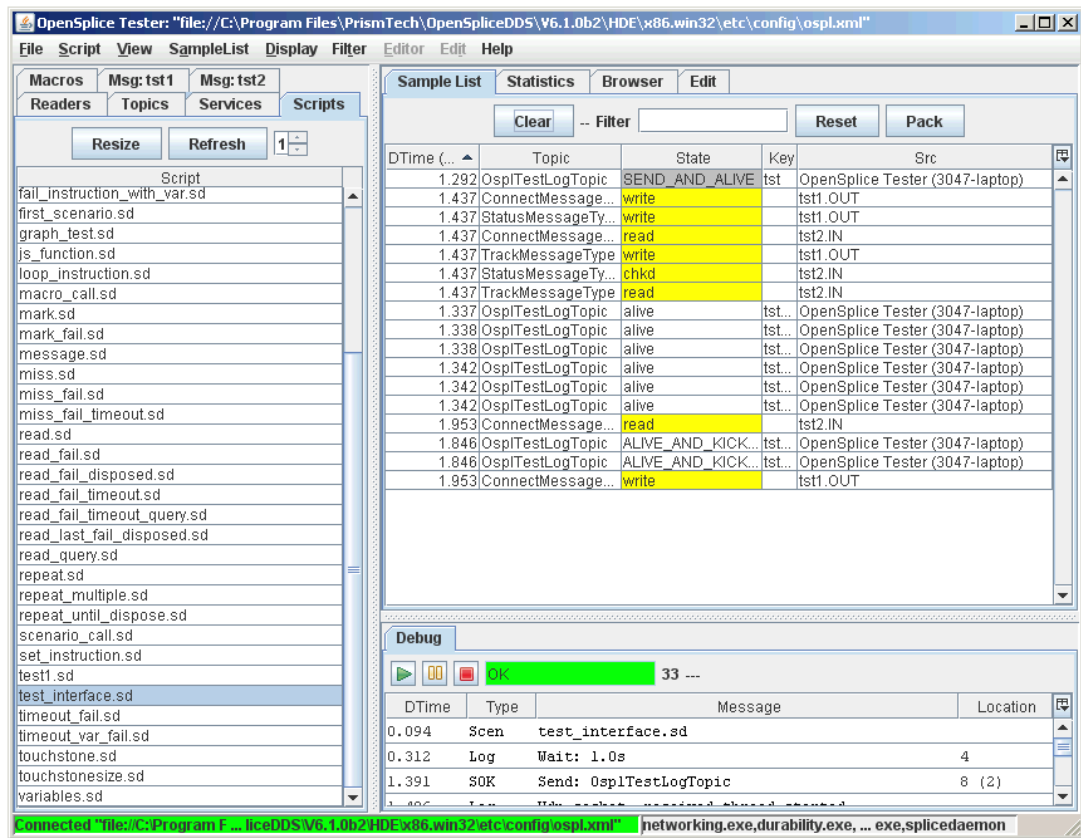
Here we can see that in the sample list, both the DDS samples as well as the `testinterface` samples are available. As a result the interaction is clearly visible.

7.3 Types of interfaces

When integrating a test interface with the Vortex OpenSplice Tester, the following functionality is provided:

- Connect/Disconnect with a parameter
- Write of messages based on parameters of a write call
- Read of messages and display received messages in the sample list
- Check received messages
- Display fields of messages (double click in sample list)
- Hooks upon write/read of a message

The Vortex OpenSplice Tester provides two ways to create such a message interface:

- Basic message interface
- Buffered message interface

7.3.1 Basic message interface

If it is not possible to describe the content of each message in an ADA interface description (*i.e.* a static memory-mapped definition of each message type) or when the definition of the interface exists in another format, like a MIB for an SNMP interface, then it is possible to derive from the basic message interface class:

```
BaseMsgInterface
```

Similarly for the messages a class must be derived from:

```
MsgSample
```

Note that both `BaseMsgInterface` and `MsgSample` contain a considerable number of abstract function which then must be provided in order to be able to edit and display samples, as well as read and write sample on their interface.

7.3.2 Buffered message interface

The example test interface is a buffered message interface. The Vortex OpenSplice Tester provides support for memory-mapped messages and provides all basic functionality for this type of interface. The messages are described using the ADA language type definition for records with a representation clause. This allows to describe message with bit fields, enums, fixed length strings, integer and double values.

On top of the buffered message interface, an implementation using UDP and TCP is available.

When the buffered message interface is used the provided implementation takes care of interpreting the received messages, decode the messages for display in the sample list or display in the sample window. Upon a write instruction a memory buffer will be built using the parameters of the write call and the message definition as provided in the ADA interface description.

ADA Syntax for message definition

For each message a record needs to be defined which describes the exact memory layout of the message. See the ADA message description of the test interface for an example of such a message definition.

Message ID translation

By default in a buffered message interface a base record is defined with an idfield to determine the type of the message. Then a function is called to translate the value of the idfield to a name of the record type with the definition of a message of the received type.

If a message does not contain a single field which can be used to determine the message type then the method:

```
RecordType determineMsgType(ByteBuffer buf)
```

can be overwritten to perform the translation of the received buffer to a message type.

If indeed the id can be retrieved from an id field (enum value) then the function:

```
protected static String transformIdToType(String id)
```

is used to translate the enum label to the name of a record definition. The following translation is done:

- ID is changed to TYPE

- Each character following an underscore ('_') is capitalized, as well as the first character and the remaining characters are made lowercase.

As a result an enum label: `HEARTBEAT_MESSAGE_ID` is translated to `HeartbeatMessageType`.

Of course if a different convention is used for describing enum labels and message names, then the transformId-ToString function can be overridden to perform the required translation.

Message Hooks

It is possible to override message hooks at several stages in the send and receive process. This allows specific processing, such as:

- Automatic reply to each received message (acknowledge messages)

- Fill in automatic fields like sequence numbers, crcs, or timestamps

- Ignore messages for reception, like acknowledge messages or heartbeats

- Perform specific checks such as crc check

See the example test interface for an example of the hooks and a description of their function.

Control functions

The script control function allows implementation specific control functions to be implemented. In the implementation of the derived interface, the following functions can be overridden (note that the base implementation already provides some control commands, overriding these functions must properly include or forward to the base implementation):

```
public String[] getControlCommands()
```

Provide the list of control commands, note that `super.getControlCommands` should be used to include the list of control commands of the base impementation.

```
public void control(String command, ParameterList params,
    ScenarioRuntime runtime, int line, int column)
```

Execute the control command, with the provided parameters and runtime. In case of an error the line and column can be used as the location of the instruction which failed.

Control functions can be used for any specific function as deemed necessary (of course, all must be implemented in the derived interface class):

- Stop sending heartbeats
- Create incorrect crc
- Stop sending acknowledge
- Determine message frequency

8

Google Protocol Buffers

This section describes Tester features for Google Protocol Buffers.

8.1 About Google Protocol Buffers in Tester

In versions of Vortex OpenSplice that support Google Protocol Buffers, Tester is able to read from protocol buffer topics and display its samples as regular field name and value pairs, just as if it were from a regular IDL-defined topic.

The Tester feature for Google Protocol Buffer topic reading is enabled only on Vortex OpenSplice middleware installs where Google Protocol Buffer support is included. For installations where it is not included, the feature is disabled in Tester.

8.2 Viewing type evolutions

The main feature of using Google Protocol Buffers as the type definition for a topic is the ability to change, or 'evolve', a topic's type. Tester can become aware of changes to a protocol buffer topic's type, and can display the topic type definition for each type evolution that is registered.

To view the type evolutions for a protocol buffer topic, right-click the topic in the *Topics* tab, and select *View Topic Type* from the pop-up menu.

View Topic Type from the Topics list
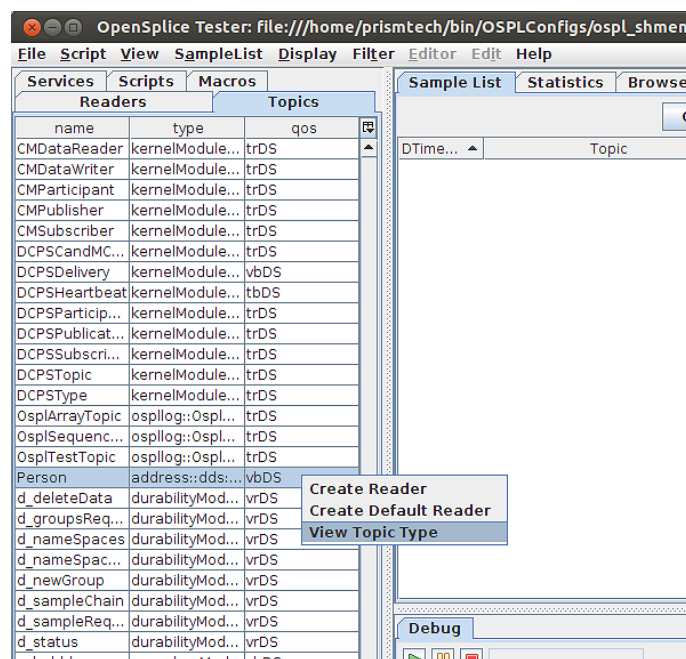
8.2.1 View Topic Type Window

Upon selection of the menu item, the *View Topic Type* window will appear for the selected topic. By default, it shows the type name for the topic and the DDS type definition. If the topic is a protocol buffer topic, it will display additional information:

- The *Evolutions* for the type are displayed in a drop-down combo box. It lists the known evolutions for the type according to the time it was registered in the DDS system. The most recent evolution is at the top of the list and is selected by default.

- The *Type Hash* is an non-editable text field that displays the 128-bit hash that uniquely identifies the selected type evolution for the topic type. The field is highlightable for easy copy/pasting.

- The *Proto Typedef* tab is added when viewing protocol buffer topic types. It displays a description for all fields and nested types defined in the protocol buffer message for the currently-selected evolution, in a text format emulating the original `.proto` file format. Message type fields found in the typedef that are not defined as nested messages inside the main DDS message type are defined under a separate section, `External Definitions`. These messages have their fully- qualified type name to indicate where they were defined. Please note that this typedef reconstruction is only meant to give the user an idea as to what type of data is found in the topic type. It is not guaranteed to be a 100% reconstruction of the original `.proto` file as it was written, or to be compilable by the `protoc` compiler.

- The *DDS Typedef* tab contains the same kind of type decription as it would for a normal topic, but for protocol buffer topics, it describes the field names and field types for the currently-selected type evolution as it is understood by Tester for sample editing.

View Topic Type window for a protocol buffer topic
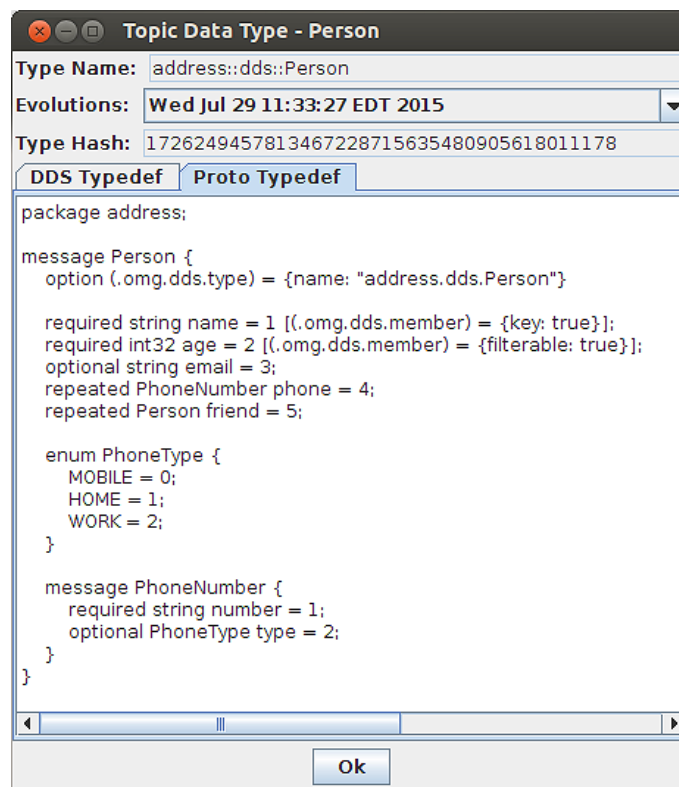
Proto Typedef with external definitions
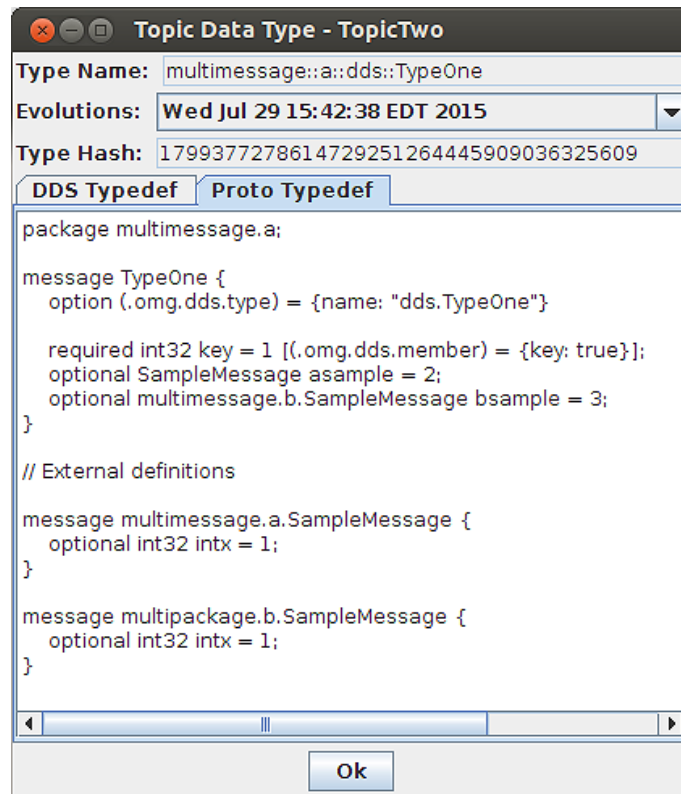
8.3 Reading protocol buffer topics

Tester reads protocol buffer data by reading in the byte sequence data contained in the user data, and then replacing all fields with regular field name and value pairs just as if it were data from a regular topic. The process for creating readers for protocol buffer topics is almost identical to the process described in Section 3.3.3, To Add a Reader from the Topic list, on page 24.

- When creating a reader for a protocol buffer topic *via* the *Create Default Reader* right-click menu option on the *Topic* list, all default settings will be used as before, but with the addition that all incoming samples will be decoded as the most recently-registered type evolution.

- When creating a reader for a protocol buffer topic *via* the *Create Reader* right-click menu option on the *Topic* list, or *via* the *File > Add Reader* main menu option, the *Add Reader* dialog will be presented, as normal. However, it is presented with an extra field to choose the desired type evolution to use to view user data.

Add Reader dialog with type evolution chooser

8.4 Reading protocol buffer topics *via* script

It is possible to declare which type evolution to read protocol buffer topics as, *via* the `Reader` script command (defined in Section 5.2.15, Reader, on page 83). The `topicname` parameter for the reader command can be modified with a type evolution's type hash to specify which type evolution view protocol buffer user data as. The type hash can be viewed and copied from the *View Topic Type* window (see Section 7.2.1 on page 96).

- To create a default reader with the most recently-registered type evolution, the command is:

      reader(Person);

- To create a reader with a specific type evolution, the command is the same, but with the type hash pasted in after the topic name, separated by a '#':

      reader(Person#7397941026954504224985160522196071931 9);

8.5 Editing protocol buffer topic samples

Editing samples from protocol buffer topics is the same as samples from a regular topic. Protobuf fields can be declared as optional or required, so to reflect that in the *Sample Edit* window (see Section 3.4.1.1, To Write Sample Topic data, on page 28), a Cyan color highlight is added to required protocol buffer fields. In all other senses, though, editing samples in either the *Sample Edit* window or in scripting is precisely the same as it is for normal topics.

The Sample Edit Window for a protocol buffer sample

Field	Default	value
Edit sample: Person		
name		Jane Doe
age	0	4
email		jane.doe@somedomain.com
facebookname	NONE	NONE
friend[0].age	0	4
friend[0].email		john.doe@somedomain.com
friend[0].facebookname	NONE	faceb
friend[0].name		John Doe
friend[0].phone[0].number		
friend[0].phone[0].secret	false	
friend[0].phone[0].type	UNDEFINED	
friend[1].age	0	
friend[1].email		
friend[1].facebookname	NONE	
friend[1].name		
friend[1].phone[0].number		
friend[1].phone[0].secret	false	
friend[1].phone[0].type	UNDEFINED	
phone[0].number		0123456789
phone[0].secret	false	false
phone[0].type	UNDEFINED	HOME
phone[1].number		0612345678
phone[1].secret	false	true
phone[1].type	UNDEFINED	MOBILE
phone[2].number		splicer
phone[2].secret	false	false
phone[2].type	UNDEFINED	SKYPE
phone[3].number		
phone[3].secret	false	
phone[3].type	UNDEFINED	

write	writeDispose	dispose	script	check

9

Python Scripting Engine

This section describes writing Python scripts to test OSPL applications.

9.1 About Python Scripting

Python Scripting is a scripting environment for running unit tests and adhoc scripts against a Vortex OpenSplice environment. Python Scripting connects to the OpenSplice environment through the Configuration and Management API. It is based on the Python scripting language, and requires Jython 2.7.0 or later, the Java-based implementation of Python.

9.1.1 Design Goals

Python Scripting is an alternative to the product specific Scenario language found in Vortex Tester. Python Scripting design goals were:

- use a standard scripting language (Python)

- allow users to leverage the tooling environment build around this standard language

- allow users to leverage the testing infrastructure and libraries

- expose DDS semantics, particularly on reading and writing topic data in an easy-to-use API for Python.

- run independently of Vortex Tester

- allow execution of scripts and unit tests both directly from the command line and from an Integrated Development Environment (IDE)

9.2 Configuration

This section explains how to download and configure a Jython scripting engine with the Tester Scripting package. Jython is an Java-based implementation of the Python scripting language.

9.2.1 Prerequisites

A Java 7 or later runtime (JRE) is required. The JAVA_HOME environment variable must be set to the JRE installation directory.

Check that the following command yields a Java version of 1.7 or later. On Linux systems, use:

```
$JAVA_HOME/bin/java -version
```

On Windows systems, use:

```
%JAVA_HOME%\bin\java -version
```

Note that during the installation, it is assumed the the environment variable OSPL_HOME refers the the fully qualified path of the Vortex OpenSplice installation.

9.2.2 Download and install Jython

The Tester Scripting package does not include the Jython scripting engine - it must be downloaded. Jython is an open source project licensed under the Python Software Foundation License Version 2 (http://www.jython.org/license.html). The license is OSI Approved (https://opensource.org/licenses/alphabetical).

Tester Scripting requires Jython 2.7.0 or later. Follow the following steps to download and install Jython:

1. Download the Jython 2.7.0 Installer from the Jython Download page (http://www.jython.org/downloads.html).

2. Install Jython. It is recommended, but not necessary that Jython be installed in the $OSPL_HOME/jython directory (linux) or the %OSPL_HOME%jython directory (Windows). The following command line will install the standard Jython distribution:

```
# for Linux systems
$JAVA_HOME/bin/java -jar jython-installer-2.7.0.jar -s -d $OSPL_HOME/jython -t standard

# For Windows Systems
"%JAVA_HOME%\bin\java" -jar jython-installer.2.7.0.jar -s -d "%OSPL_HOME%\jython" -t standard
```

3. Change to the OSPL_HOME directory.

4. Install the Tester Scripting python package. Use the following command:

```
# for Linux systems
jython/bin/easy_install tools/scripting/OSPLScript-1.0.0.tar.gz

# for Windows systems
jython\bin\easy_install tools\scripting\OSPLScript-1.0.0.tar.gz
```

9.2.3 Verifying the installation

The following steps will verify that the Jython installation is correctly configured.

1. Ensure OpenSplice is running by issuing the following command:

```
ospl start
```

2. Start OpenSplice Tester to create some test topics used below:

```
ospltest
```

3. Start the Jython interpreter:

```
# for Linux systems
jython/bin/jython -Djava.ext.dirs=$OSPL_HOME/jar

# for Windows systems
jython\bin\jython -Djava.ext.dirs=%OSPL_HOME%\jar
```

4. Enter the following commands at the interpreter command prompt:

```
>>> from osplscript import dds
>>> topic = dds.findTopic('OsplTestTopic')
>>> OsplTestTopic = dds.classForTopic(topic)
>>> data = OsplTestTopic()
>>> data.id = 1
>>> data.description = 'Smoke test'
```

```
>>> writer = dds.Writer(topic)
>>> writer.write(data)
```

```
>>> reader = dds.Reader(topic)
>>> sample = reader.take()
>>> readData = sample.getData()
```

```
>>> assert data.id == readData.id
>>> assert data.description == readData.description
```

9.3 A Quick Tour of OSPL Scripting

The following is a brief tour of the scripting engine's capabilities.

9.3.1 Prerequisites

This demo assumes a shell instance that has been initialized with the release.com script found in the OSPL installation directory. In particular, the quick tour relies on the $OSPL_HOME, $OSPL_URI and $LD_LIBRARY_PATH variables being appropriately set. To run the script, do the following:

```
cd OSPL-install-directory
. release.com
```

9.3.2 Preliminaries

Start OSPL, typically with:

```
ospl start
```

Then, start Vortex OpenSplice Tester:

```
ospltest
```

Tester is used to define topics used in the scripting engine, and to observe samples. From within Tester, create a default Reader on the following topics: OsplTestTopic.

9.3.3 Writing and Reading samples

The instructions below assume that you have installed and configured Jython in $OSPL_HOME/jython. Start the OSPL Scripting engine:

```
$OSPL_HOME/jython/bin/jython -Djava.ext.dirs=$OSPL_HOME/jar
```

You will see a standard start up banner from the Jython engine similar to the following:

```
Jython 2.7.0 (default:9987c746f838, Apr 29 2015, 02:25:11)
[Java HotSpot(TM) Server VM (Oracle Corporation)] on java1.7.0_80
Type "help", "copyright", "credits" or "license" for more information.

>>>
```

The text '>>>' is the interpreter prompt.

To start our script, import the DDS module from OSPL Script. Enter the following at the prompt:

```
>>> from osplscript import dds
```

As part of the import, a connection is made to OSPL. If OSPL is not running, or if the environment variables are not correctly set, you may receive an error at this point.

The dds module provides allows you to find topics, and create readers and writers. We will start with find the OsplTestTopic created when we started Tester. Enter the following at the prompt:

```
>>> t = dds.findTopic('OsplTestTopic')
>>> # The next statement create a Python type from the topic
>>> OsplTestTopic = dds.classForTopic(t)
>>> # You can then instantiated instances of this class
>>> d1 = OsplTestTopic()
>>> # You can then set fields in the data
>>> d1.id = 1
>>> d1.index = 100
>>> d1.description = 'Hello from osplscript'
>>> d1.state = 'boost'
>>> d1.x = 1.1
>>> d1.y = 2.2
>>> d1.z = 3.3
>>> d1.t = 4.4
```

Once some data is created, we can the create a writer for the topic, and write the data:

```
>>> # create a Writer from the topic object we found previously
>>> w = dds.Writer(t)
>>> w.write(d1)
```

Once the write has executed, example the Sample List in Tester. You should see a new sample in the list.

Next, we can create a reader, and read the sample we have just written:

```
>>> # create a Reader on the topic, and read a sample
>>> r = dds.Reader(t)
>>> s1 = r.take()
>>> # s1 is a Sample, the user data is access via getData()
>>> rd1 = s1.getData()
```

We can check that the read data is what we expected:

```
>>> assert rd1.id == 1
>>> assert rd1.index == 100
>>> assert rd1.description == 'Hello from osplscript'
```

You can continue your exploration by writing a sample via Tester, and confirming that you can read it using OSPL Script. When you are done, exit the interpreter by typing:

```
>>> exit()
```

9.3.4 Working with QoS settings

By default, OSPL Scripting creates a publisher and a subscriber using the partition pattern '*'. All other publisher and subscriber QoS policies are DDS defaults. Similarly, data readers and data writers have, by default, QoS policies derived from topic to which they are bound. If default QoS policies do not statisfy your requirements, you can explicitly create publishers and subscribers, and assign them explicity QoS policies. Similarly, you can explicitly assign QoS policies to data readers and data writers that you create.

The following example shows the explicit creation of a Subscriber:

```
>>> from osplscript import dds, qos
>>> topic = dds.findTopic('OsplTestTopic')
>>> # create an explicit subscriber on a partition 'test'
>>> sub = dds.Subscriber(
>>>     qos.SubscriberQos().withPolicy(
>>>         qos.Partition().withName('test'))
```

```
>>>     )
>>> # create a reader with topic-derived defaults
>>> drDefault = sub.reader(topic)
>>> # create a reader with explicity reader QoS policies
>>> drExplicit = sub.reader(
>>>     topic,
>>>     qos.DataReaderQos().withPolicy(
>>>         qos.Durability().withVolatile())
>>>     )
```

The following example shows adding explicit QoS policies to a data writer on the default publisher:

```
>>> dwExplicit = dds.Writer(
>>>     topic,
>>>     qos.DataWriterQos().withPolicy(
>>>         qos.Durability().withVolatile())
>>>     )
```

The QoS classes for publishers, subscribers, data readers and data writers are, respectively, PublisherQos, SubscriberQos, DataReaderQos and DataWriterQos. Although OSPL Scripting does not allow creating of topics, topics can return their QoS settings via a TopicQos instance. The help for each of these QoS classes describes the applicable policies. The help for policy classes describes all the methods available to configure the policy. You can view help on QoS classes and policies via the help function:

```
>>> help(qos.SubscriberQos)
>>> help(qos.Parition)
>>> help(qos.Durability)
```

9.3.5 Working with WaitSets

OSPL Scripting implements DDS wait sets with read conditions, query conditions and status conditions. This allows your code to block until data is available on a data reader. Here is a simple example:

```
>>> from osplscript import dds
>>>
>>> # find the topic, create a data reader and wait set
>>> topic = dds.findTopic('OsplTestTopic')
>>> dr = dds.Reader(topic)
>>> ws = dds.WaitSet()
>>>
>>> # create a read condition of Alive, NotRead, New samples
>>> rc = dr.readCondition(dds.DataState().withAlive().withNotRead().withNew())
>>>
>>> # attach the read condition to the wait set
>>> ws.attachCondition(rc)
>>>
>>> # wait...
>>> ws.waitForConditions()
>>>
>>> # waiting returned, we have a sample
>>> sample = dr.take()
>>> # do something with the sample
```

The waitForConditions() method can accept optional arguments, include a timeout. See the help for details:

```
>>> help(dds.WaitSet.waitForConditions)
```

9.3.6 Filtering data

OSPL Scripting allows you to filter data from a reader using a 'selector'. The selector can also be used to create a condition for a wait set.

A selector is created via a reader's newSelectBuilder() method. A 'select builder' allows you to specify state conditions as well as an optional query expression. The following example creates a selector:

```
>>> from osplscript import dds
>>>
>>> # find the topic, create a data reader
>>> topic = dds.findTopic('OsplTestTopic')
>>> dr = dds.Reader(topic)
>>>
>>> # create a selector for Alive, NotRead, New samples with 'index = 100'
>>> selector = dr.newSelectBuilder().withAlive().withNotRead().withNew() \
>>>     .content('index = 100').build()
```

Each of the 'selector builder' methods returns the builder, so that calls can be chained as above. Apart from build(), none of the builder methods is required. A selector with no filters works identically to the data reader from which it was created.

Once created, a selector can be used like a data reader, with take() or read() methods:

```
>>> # use the selector like a reader
>>> sample = selector.take()
```

Alternatively, the selector can be used with a waitset, by calling the selector's condition() method:

```
>>> # Use the selector in conjuction with a waitset
>>> ws = dds.WaitSet()
>>> ws.attachCondition(selector.condition())
>>> ws.waitForConditions()
>>> # the selector now has data...
>>> sample = selector.take()
```

In the above example, the selector.condition() method returns a QueryCondition. If the content() method had not been called, a ReadCondition would have been returned.

You can create a QueryCondition directly, and then use it with a wait set:

```
>>> queryCond = reader.queryCondition(
>>>     'index = 100', [],
>>>     dds.DataState().withAlive().withNotRead().withNew())
>>> ws.attachCondition(queryCond)
```

A selector, however, has the advantage of providing your with filtered access to the data that triggered the query condition. Creating a query or read condition explicitly does not provide such filtering; the data reader from which the condition was defined will still return all available samples, whether they satisfy the condition or not. For this reason, selectors are the preferred method for defining data filters, waiting for filtered data availability, and for accessing the filtered data.

9.3.7 Query Expressions, Query Parameters and their Limitations

Both the selector builder's content() method and the QueryCondition() constructor allow the query expression to contain substitution parameters of the form {n}, where n is a zero-based index into an list of string parameter values. For example, we could filter OsplTestTopic's index value to be between an upper and lower bound, and specify the query as follows:

```
>>> from osplscript import dds
>>>
>>> # find the topic, create a data reader
>>> topic = dds.findTopic('OsplTestTopic')
>>> dr = dds.Reader(topic)
>>>
>>> # create a selector for Alive, NotRead, New samples with 'index = 100'
>>> selector = dr.newSelectBuilder().withAlive().withNotRead().withNew() \
```

```
>>>       .content('index >= {0} and index < {1}', ['100', '200']) \
>>>       .build()
```

We could use similar parameters in creating a QueryCondition directly:

```
>>> queryCond = reader.queryCondition(
>>>     'index >= {0} and index < {1}',
>>>     ['100', '200'],
>>>     dds.DataState().withAlive().withNotRead().withNew())
```

OSPL Scripting attempts to replace the passed parameter values in the query expression, formatting the values as valid query expression constants. However, because of limitations in the APIs available to OSPL Scripting, this formatting is imperfect. In particular, the following values are likely to be formatted incorrectly:

- enumeration values will incorrectly be quoted

- boolean values will incorrectly be quoted

- string values that can be converted to numbers will incorrectly be unquoted.

The work around for all these limitations is to avoid using parameter substitution. For example, instead of the following parameterized condition against the 'state' enumeration field in OsplTestTopic:

```
>>> # DON'T DO THIS FOR AN ENUMERATED FIELD
>>> selector = dr.content('state = {0}', ['init']).build()
```

Instead, write the condition without substitution:

```
>>> selector = dr.content('state = init').build()
```

9.3.8 Using Coherent access

OSPL Scripting now supports Group and Topic coherence when reading and writing samples.

Group coherence allows a publisher to release a group of samples, possibly spanning several topics to subscribers in a group. Subscribers will not see any samples in a coherent group until the publisher has completed group. A subscriber using Group coherence may, if desired, retrieve the samples, across all readers in the group, in the order that the publisher wrote them.

Topic coherence allows a publisher to release changes across multiple instances of the sample topic as a coherent set. Subscribers will not see any samples in a coherent set util the publisher has completed the set. A subscriber using Topic coherence may, if desired, retrieve the samples with-in a specific topic, in the order that the publisher wrote them.

To establish a publisher or subscriber with coherent access, use the Presentation Policy when creating the publisher or subscriber QoS:

```
>>> from osplscript import dds, qos
>>>
>>> # create a presentation policy, enabling Group coherence and ordered access
>>> groupPresentation = qos.Presentation().withCoherentAccess().withGroup() \
>>>     .withOrderedAccess()
>>>
>>> # create a publisher with the policy
>>> pub = dds.Publisher(qos.PublisherQos().withPolicy(groupPresentation))
>>>
>>> # create a subscriber with the policy
>>> sub = dds.Subscriber(qos.SubscriberQos().withPolicy(groupPresentation))
```

To create a presentation policy with Topic scope, use the withTopic() method. To create a presentation policy with Instance scope (the default), use the withInstance() method.

To write samples using any coherent scope, the publisher object offers the methods beginCoherentChanges() and endCoherentChanges(). Each call beginCoherentChanges() should be matched with a call to endCoherentChanges(). Calls may be nested, for programmer convenience, but only the outer most pair of calls have

any impact. Within a coherent change, any of the publisher's data writers may be used to write samples. When the coherent change is completed (via a call to endCoherentChanges()), the samples are released to subscribers.

To enable subscribers to read coherent changes, the subscriber object offers two methods: beginAccess() and endAccess(). Their use is optional, but without them, the subscriber will not guarantee that samples will be returned according to the coherent groups in which they were created.

The following code pattern my be used to access samples written with Group coherence, or with ordered access:

```
>>> # sub is a subscriber with Group coherence presentation policy
>>> # the subscriber must create data readers for all the topics
>>> # it wants to access
>>> sub.beginAccess()
>>> # return the dataReaders, in the order their corresponding writers
>>> # wrote samples
>>> drList = sub.dataReaders()
>>> for dr in drList:
>>>     # take (or read) ONLY ONE sample for each element of the list
>>>     sample = dr.take()
>>>     # using the reader's topicDescription() method to identify the topic
>>>     if 'foo' == dr.topicDescription().getName():
>>>         # do something with a 'foo' sample
>>> sub.endAccess()
```

When not using Group access or ordered access, a subscriber may still use the dataReaders() method, however, in this case, dataReaders() will return a set of readers with available data. The subscribing application can then take or read as many samples as are available from each reader in the set.

Note: Using waitsets with group coherence

The DDS specification indicates that subscribers will raise a 'data available on readers' event when a new group is release. However, a defect in underlying APIs prevent OPSL Scripting from receiving this event. In other words, the following will not work:

```
>>> # This will NOT WORK!!!
>>> from osplscript import status
>>> ws = dds.WaitSet()
>>> sc = sub.statusCondition()
>>> sc.setEnabledStatuses([status.DataOnReadersStatus])
>>> ws.attachCondition(sc)
>>> ws.waitForConditions()
>>> # will NEVER get here
```

As a work around, attached status conditions from each of the subscriber's readers to the wait set:

```
>>> # This will work
>>> ws = dds.WaitSet()
>>>
>>> # do this for each reader (dr) you care about
>>> sc = dr.statusCondition()
>>> sc.setEnabledStatuses([status.DataAvailableStatus])
>>> ws.attachCondition(sc)
>>>
>>> ws.waitForConditions()
```

9.3.9 Creating a unit test script

Although using an interactive interpreter provides instant feedback, it is more likely that you will create script files, and execute them. In this section, we will create and execute a script that performs a unit test using the stand python unittest module. Start by created a text file in your favourite editor. Call the file firstUnitTest.py. Copy and paste the text below:

```python
import unittest
from osplscript import dds


class firstUnitTest(unittest.TestCase):

        def testReadOsplTestTopic(self):
            t = dds.findTopic('OsplTestTopic')
            dw = dds.Writer(t)
            OsplTestTopic = dds.classForTopic(t)
            wdata = OsplTestTopic()
            wdata.id = 5
            wdata.x = 5.1
            wdata.y = 5.2
            wdata.z = 5.3
            wdata.t = 5.4
            wdata.state = 'hit'
            wdata.index = 5
            wdata.description = 'test'
            dw.write(wdata)

            dr = dds.Reader(t)
            sample = dr.take()
            rdata = sample.getData()

            self.assertEqual(wdata.id, rdata.id)
            self.assertEqual(wdata.x, rdata.x)
            self.assertEqual(wdata.y, rdata.y)
            self.assertEqual(wdata.z, rdata.z)
            self.assertEqual(wdata.t, rdata.t)
            self.assertEqual(wdata.state, rdata.state)
            self.assertEqual(wdata.index, rdata.index)
            self.assertEqual(wdata.description, rdata.description)

# standard python to run the unit test from the command line
if __name__ == "__main__":
        #import sys;sys.argv = ['', 'Test.testName']
        unittest.main()
```

This test case essentially repeats the test we created in the interpreter. To run the test, enter the following command in your shell command prompt:

```
$OSPL_HOME/jython/bin/jython –Djava.ext.dirs=$OSPL_HOME/jar firstUnitTest.py
```

The script engine will respond with output like the following:

```
.
----------------------------------------------------------------------
Ran 1 test in 0.107s

OK
```

The output is compact. Python's unit test philosophy is to minimize output except in the case of test failures. Experiment with the test to introduce a failure, and see how the output changes.

9.3.10 Working with more Complex topics

The OsplTestTopic used above is simple. This section examines working with more complex topics – ones that include sequences, arrays, nested structures and unions. As with the preceding examples, Tester should be running, as it creates the samples that are used in this example. If OSPL Scripting is not running, start it:

```
$OSPL_HOME/jython/bin/jython –Djava.ext.dirs=$OSPL_HOME/jar
```

The use the following Python to find the OsplSequenceTopic DDS topic and create a Python class from it:

```
>>> from osplscript import dds
```

```
>>> seqTopic = dds.findTopic('OsplSequenceTopic')
>>> seqClass = dds.classForTopic(seqTopic)
>>> seqInstance = seqClass()
```

From there, you can explore the instance data object. Fundamentally, it behaves pretty much like a C 'struct' would: top level fields are accessed via the dot notation:

```
>>> seqInstance.id = 1
```

Fields that are arrays or sequences may be indexed with zero-based value. Standard python sequence methods may be used to add and remove elements from the lists. (Fields declared as arrays are pre-allocated to the declared size. OsplSequenceTopic contains no array fields, but you can explore OsplArrayTopic, which does.)

```
>>> seqInstance.iVector.append(1)
>>> seqInstance.iVector.append(2)
>>> assert seqInstance.iVector[0] == 1
>>> assert seqInstance.iVector[1] == 2
>>> del seqInstance.iVector[0]
>>> assert seqInstance.iVector[0] == 2
```

If a class includes a a sequence of structures, then a fieldName_new() method is created so you can instantiate instances of the class:

```
>>> seqInstance.pVector.append(seqInstance.pVector_new())
>>> seqInstance.pVector[0].state = 'boost'
```

9.3.11 Creating a sample time-line

The OSPL Scripting engine includes module (osplscript.recorder) that can automatically read samples from one or more topics and return these in the order received. These sample sequence can then be queried using standard python mechanisms to create tests that consider multiple samples at once. To use the recorder module, do the following:

```
>>> from osplscript.recorder import Recorder
>>> t1Recorder = Recorder('OsplTestTopic')
>>> t2Recorder = Recorder('OsplSequenceTopic')
>>> # ... time passes ...
>>> # get a snapshot of samples recorded
>>> t1Samples = t1Recorder.getSamples()
>>> t2Samples = t2Recorder.getSamples()
>>> # clear the recorded samples, but keep recording
>>> t1Recorder.clearSamples()
>>> # stop recording, the sample list is still available, but no longer updated
>>> t1Recorder.stop()
```

9.4 Using Eclipse and PyDev to create and run OsplScript files

Eclipse is a popular open source IDE. PyDev is a Python specific open source add-on for Eclipse. This chapter describes using Eclipse and PyDev.

9.4.1 Download and Installation

Eclipse may be obtained from the Eclipse Download page. Choose the **IDE for Java Developers** download. Installation Instructions are available on the Eclipse Install Guide page.

Once Eclipse is installed, start it and choose or create a workspace. You are then ready to proceed with installing PyDev. See the PyDev Download page for instructions - instructions appear on the right-hand side of the page under **Quick Install > Update Manager**.

9.4.2 Configuration

Once PyDev is installed, you must configure it with the location of you Jython installation:

1. Start Eclipse, and choose a workspace if prompted.

2. From the menu, choose **Window -> Preferences**.

3. In the left-hand tree, find and click **PyDev > Interpreters > Jython interpreters**.

4. Click the **Add** button.

5. Enter a name for the Jython interpreter. Example: Tester Script

6. Browser for the jython.jar file in the root directory of the Tester Script Jython installation.

7. Click OK.

8. Still in the Preferences dialog on the PyDev > Interpreters > Jython interpreters page, select your newly created interpreter entry in the upper list. Then click the **Environment** tab in the lower half of the dialog. 9. Add the following environment variables: OSPL_HOME, OSPL_URI and LD_LIBRARY_PATH. Their values should be the same as those found in your command line environment. 10. Click the **Libraries** tab in the lower half of the dialog. 11. Click **New Folder** and browse for and select the **jar** directory under your Vortex OpenSplice installation directory. 12. Click **OK** to complete the folder selection. 13. Click **OK** to close the preferences dialog.

You are new ready to create a new Jython project.

9.4.3 Creating a PyDev Project

This section describes creating a project for editing Python files.

1. Start Eclipse and choose a workspace, if it is not already running.

2. From the menu, choose **File -> New -> Project**.

3. In the new project wizard, select **PyDev > PyDev Project**. Click **Next**.

4. Enter a project name.

5. Ensure the **Project type** is set to **Jython**.

6. Ensure that the **Interpreter** is set to your Tester Script Jython interpreter configured above.

7. Click **Finish**.

8. In the Package Explorere, right click over the newly created project, and choose **Properties**.

9. In the left-hand pane, click **PyDev - PYTHONPATH**.

10. Click the **External Libraries** tab.

11. Click **Add zip/jar/egg**, and choose cmapi.jar from the jar subdirectory of your Vortex OpenSplice installation. Click **OK**.

12. Click **Add zip/jar/egg**, and choose osplscript.jar from the jar subdirectory of your Vortex OpenSplice installation. Click **OK**.

13. Click **OK** to close the properties dialog.

You may be prompted to switch to the PyDev perspective. This is optional. The PyDev perspective adapts the Eclipse display for editing python files with PyDev. If you are new to Eclipse, it is recommended that your switch to the PyDev perspective. The following instructions assume you are in this perspective.

9.4.4 Create a Python script

Python files may declare classes, define unit tests, or just provide instructions that are to be executed when the file is run. To create a Python file:

1. From the menu choose **File** -> **New PyDev Module**.

2. If not set, browser for a Source Folder, which must be a directory in your project.

3. Optionally, enter a package name.

4. Enter the name of the python file. PyDev will add a .py extension automatically.

5. Click **OK**

An editor will open, and you will be prompted for a template for the newly created file. The most common choices are: * one of UnitTest variations, if you want to write tests. * Main, if you want to write a python script to be executed directly by the interpreter * Class, if you want to define python classes to be consumed by other modules.

9.4.5 Running a Python script

To run a python script (or unit test), do the following:

1. Right click anywhere in the editor and choose **Run As** -> **Jython Run**. (For unit tests, choose **Run As** -> **Jython unit-test**.)

9.4.6 Debugging a Python script

Launching a debugger is similar to running a script. Right click the script, and choose **Debug As** and then the appropriate sub-menu item. While debugging, note the following:

- You can set break points in a script by clicking in the left margin of the script editor.

9.5 Using PyCharm to create and run Tester Scripting

PyCharm is a Python specific IDE developed by Jet Brains, the makes of IntelliJ IDEA. PyCharm comes in several forms. This chapter describes using the free Community Edition.

Note that similar instructions apply for using IntelliJ IDEA with the Python Plugin, as the Python Plugin adds very similar capabilities.

9.5.1 Download and Installation

PyCharm may be obtained from the PyCharm Download page. Choose the Community Edition download. The page includes a link to Installation Instructions appropriate to your platform.

9.5.2 Configuration

This subsection explains how to configure PyCharm so that is will work with the Jython installation you created and configured with Tester Script.

1. Start PyCharm. On the Welcome script.

2. Click the **Configure** drop down near the bottom of the screen, and then choose **Settings**.

3. In the left-hand tree, find **Default Project**, expand it and select **Project Interpreter**.

4. On the right-hand side of the dialog, click the gear icon, and choose **Add Local**.

5. In the file selection dialog, browse for the your Jython executable from your Jython installation. The Jython installation instructions used the following location: $OSPL_HOME/jython/bin/jython

6. Click OK to close the selection dialog. Click OK again to close the Settings dialog.

You are new ready to create a new Jython project.

9.5.3 Creating a PyCharm Project

This section describes creating Python projects in PyCharm.

1. Start PyCharm, if it is not already running.

2. If the Welcome screen is showing, click **Create New Project**. Otherwise, select **New Project** from the **File** menu.

3. Provide a project name and location. Verify that the Interpreter is the Tester Scripting Jython interpreter. Click **Create**.

9.5.4 Create a Python script

Python files may declare classes, define unit tests, or just provide instructions that are to be executed when the file is run. To create a Python file:

1. From the menu choose **File** -> **New...**.

2. In the pop-up that displays, click **Python File**.

3. Provide a name. If you do not add it, PyCharm will add a .py extension.

4. The **Kind** drop down allows you to choose between **Python file** or **Python unit test**.

5. Click **OK** to create the file. The file will open in an editor.

9.5.5 Running a Python script

Running a script requires some the first time setup:

1. From the menu, choose **Run** -> **Edit Configurations**.

2. In the left-hand tree, expand **Defaults** and click **Python**

3. Press the button containing ellipsis at the end of the **Environment Variables** line.

4. Add the following environment variables (as defined in your environment): OSPL_HOME, OSPL_URI and LD_LIBRARY_PATH

5. Click **OK** to close the Environment Variables dialog

6. **In the Interpreter Options edit box, enter:** -Djava.ext.dirs=<full-path-to-OSPL_HOME>/jar

7. Ensure **Python intepreter** is set to the Jython implementation you created earlier.

If you plan on running Python unit tests, you will have to repeat the above steps for the **Python tests** default.

Once the default configurations are setup, you can run Python script as follows:

1. Right click anywhere in the editor and choose **Run**

9.5.6 Debugging a Python script

Once run configurations are setup, debug is essentially another form of running. Note the following:

• You can launch a debug session by right clicking in a script editor, and choosing **Debug**

• You can set break points in a script by clicking in the left margin of the script editor.

10

Appendix A

10.1 Scripting BNF

This Appendix gives the formal description of the Tester Scripting language.

10.1.1 TOKENS

```
TOKENS
<DEFAULT> SKIP : {
" "
| "\t"
| "\n"
| "\r"
| <"//" (~["\n","\r"])* ("\n" | "\r" | "\r\n")>
| <"--" (~["\n","\r"])* ("\n" | "\r" | "\r\n")>
| <"/*" (~["*"])* "*" ("*" | ~["*","/"] (~["*"])* "*")* "/">
}

<DEFAULT> TOKEN : {
<INTEGER_LITERAL: <DECIMAL_LITERAL> (["l","L"])?
| <HEX_LITERAL> (["l","L"])?
| <OCTAL_LITERAL> (["l","L"])?>
| <#DECIMAL_LITERAL: (["+","-"])? ["0"-"9"] (["0"-"9"])*>
| <#HEX_LITERAL: "0" ["x","X"] (["0"-"9","a"-"f","A"-"F"])+>
| <#OCTAL_LITERAL: "0" (["0"-"7"])*>
| <FLOATING_POINT_LITERAL: (["+","-"])? (["0"-"9"])+ "." (["0"-"9"])*
        (<EXPONENT>)? (["f","F","d","D"])?
| "." (["0"-"9"])+ (<EXPONENT>)? (["f","F","d","D"])?
| (["0"-"9"])+ <EXPONENT> (["f","F","d","D"])?
| (["0"-"9"])+ (<EXPONENT>)? ["f","F","d","D"]>
| <#EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+>
| <CHARACTER_LITERAL: "\'" (~["\'","\\","\n","\r"]
| "\\" (["n","t","b","r","f","\\","\'","\""]
| ["0"-"7"] (["0"-"7"])?
| ["0"-"3"] ["0"-"7"] ["0"-"7"])) "\'">
| <STRING_LITERAL: "\"" (~["\"","\\","\n","\r"]
| "\\" (["n","t","b","r","f","\\","\'","\""]
| ["0"-"7"] (["0"-"7"])?
| ["0"-"3"] ["0"-"7"] ["0"-"7"]
| ["\n","\r"]
| "\r\n"))* "\"">
| <HASH_LITERAL: "#"(["0"-"9"])+>
| <JAVASCRIPT: "`" (~["`"])* "`">
}

<DEFAULT> TOKEN : {
<SEND: "send">
| <REPEAT: "repeat">
```

```
| <PERIODIC: "periodic">
| <MACRO: "macro">
| <DISPOSE: "dispose">
| <WRITEDISPOSE: "writedispose">
| <WAIT: "wait">
| <WAITABS: "waitabs">
| <CALL: "call">
| <RUN: "run">
| <CHECK: "check">
| <CHECK_LAST: "check_last">
| <CHECK_ANY: "check_any">
| <RECHECK_LAST: "recheck_last">
| <DISPOSED: "disposed">
| <MARK: "mark">
| <MISS: "miss">
| <MARKMSG: "markmsg">
| <MISSMSG: "missmsg">
| <SCENARIO: "scenario">
| <UNIQID: "uniqid">
| <VAR: "var">
| <END: "end">
| <MSG: "message">
| <LOG: "log">
| <FAIL: "fail">
| <CLEAR: "clear">
| <IF: "if">
| <THEN: "then">
| <ELSE: "else">
| <ENDIF: "endif">
| <FOR: "for">
| <IN: "in">
| <LOOP: "loop">
| <ENDLOOP: "endloop">
| <WHILE: "while">
| <READER: "reader">
| <WRITE: "write">
| <READ: "read">
| <CONNECT: "connect">
| <DISCONNECT: "disconnect">
| <EXEC: "execute">
| <CONTROL: "control">
| <SET: "set">
| <COLUMN: "column">
| <GRAPH: "graph">
| <REVERSE_FAIL: "reverse_fail">
| <EXIT: "exit">
}

<DEFAULT> TOKEN : {
<IDENTIFIER: <LETTER> (<LETTER> | <DIGIT>)*>
| <#LETTER: ["$","A"-"Z","_","a"-"z"]>
| <DIGIT: ["0"-"9"]>
}
```

10.1.2 NON-TERMINALS

```
NON-TERMINALS
   Scenario := <SCENARIO> <IDENTIFIER> ( InstructionList )? <END>
               <SCENARIO>
   Macro := <MACRO> <IDENTIFIER> "(" ( ArgumentList )? ")"
               ( InstructionList )? <END> <MACRO>
```

```
      | <SCENARIO> <IDENTIFIER> ( InstructionList )? <END> <SCENARIO>
InstructionList   := ( Instruction )+
Instruction := SendInstruction
    | RepeatInstruction
    | PeriodicInstruction
    | DisposeInstruction
    | WriteDisposeInstruction
    | WaitInstruction
    | WaitabsInstruction
    | CheckInstruction
    | CheckLastInstruction
    | CheckAnyInstruction
    | RecheckLastInstruction
    | DisposedInstruction
    | MarkInstruction
    | MarkMsgInstruction
    | MissInstruction
    | MissMsgInstruction
    | CallInstruction
    | ForInstruction
    | WhileInstruction
    | SetInstruction
    | VarDeclaration
    | IfInstruction
    | MessageInstruction
    | ClearInstruction
    | LogInstruction
    | FailInstruction
    | ReaderInstruction
    | WriteInstruction
    | ReadInstruction
    | ConnectInstruction
    | DisconnectInstruction
    | ExecuteInstruction
    | ControlInstruction
    | ColumnInstruction
    | GraphInstruction
    | ReverseFailInstruction
    | ExitInstruction
    | ScriptInvocation
ReaderInstruction := <READER> ( <DISPOSE> )? "(" Constant
             (<HASH_LITERAL>)? ( "," <IDENTIFIER> ( "," Constant
                  ( "," Constant )? )? )? ");"
ColumnInstruction := <COLUMN> ( <CLEAR> )? "(" Constant
             ( "," Constant )? ");"
GraphInstruction  := <GRAPH> "(" ParameterList ");"
MessageInstruction   := <MSG> "(" <STRING_LITERAL> ( Constant )? ");"
LogInstruction := <LOG> "(" <STRING_LITERAL> ( Constant )? ");"
FailInstruction   := <FAIL> "(" <STRING_LITERAL> ( Constant )? ");"
ControlInstruction   := <CONTROL> <IDENTIFIER> "." <IDENTIFIER> (
             ( "(" ParameterList ( ( ");" ) | ( ")" ";" ) ) ) | ( ";" ) )
ClearInstruction  := <CLEAR> ";"
ExitInstruction   := <EXIT> ( <IF> <FAIL> )? ";"
ScriptInvocation  := Script ";"
SendInstruction   := <SEND> <IDENTIFIER> ( ( "." <IDENTIFIER> ) )?
             "(" ( ParameterList )? ");"
RepeatInstruction := <REPEAT> <IDENTIFIER> FloatValue IntValue "("
             ( ParameterList )? ");"
PeriodicInstruction  := <PERIODIC> <IDENTIFIER> <IDENTIFIER>
             FloatValue IntValue "(" ( ParameterList )? ");"
WriteInstruction  := <WRITE> <IDENTIFIER> "." <IDENTIFIER> "("
             ( ParameterList )? ");"
VarDeclaration := <VAR> FieldName "=>" Constant ";"
```

```
DisposeInstruction   := <DISPOSE> <IDENTIFIER> ( ( "."
              <IDENTIFIER> ) )? "(" ( ParameterList )? ");"
WriteDisposeInstruction := <WRITEDISPOSE> <IDENTIFIER> ( ( "."
              <IDENTIFIER> ) )? "(" ( ParameterList )? ");"
CheckInstruction   := <CHECK> <IDENTIFIER> ( ( "." <IDENTIFIER> ) )?
              "(" ( ChkParameterList )? ");"
CheckLastInstruction := <CHECK_LAST> <IDENTIFIER> ( ( "."
              <IDENTIFIER> ) )? "(" ( ChkParameterList )? ");"
CheckAnyInstruction   := <CHECK_ANY> <IDENTIFIER> ( ( "."
              <IDENTIFIER> ) )? "(" ( ChkParameterList )? ");"
RecheckLastInstruction  := <RECHECK_LAST> <IDENTIFIER> ( ( "."
              <IDENTIFIER> ) )? "(" ( ChkParameterList )? ");"
ReadInstruction   := <READ> <IDENTIFIER> "." <IDENTIFIER> "("
              ( ChkParameterList )? ");"
MarkMsgInstruction   := <MARKMSG> <IDENTIFIER> "." <IDENTIFIER> "("
              ( ChkParameterList )? ");"
MissMsgInstruction   := <MISSMSG> <IDENTIFIER> "." <IDENTIFIER> "("
              ( ChkParameterList )? ");"
ConnectInstruction   := <CONNECT> <IDENTIFIER> ( Constant )? ";"
DisconnectInstruction   := <DISCONNECT> <IDENTIFIER> ";"
DisposedInstruction  := <DISPOSED> <IDENTIFIER> ( ( "."
              <IDENTIFIER> ) )? "(" ( ChkParameterList )? ");"
MissInstruction   := <MISS> <IDENTIFIER> ( ( "." <IDENTIFIER> ) )?
              "(" ( ChkParameterList )? ");"
MarkInstruction   := <MARK> <IDENTIFIER> ( ( "." <IDENTIFIER> ) )?
              "(" ( ChkParameterList )? ");"
CallInstruction   := <CALL> <IDENTIFIER> ( ( "." <IDENTIFIER> ) )?
              "(" ( ParameterList )? ");"
SetInstruction := <SET> <IDENTIFIER> "(" ( ParameterList )? ")"
              "(" "(" ParamHeaderList ")" ParamSetList ");"
ParamHeaderList   := <IDENTIFIER> ( "," ParamHeaderList )?
ParamSetList   := "," ParamSet ( ParamSetList )?
ParamSet := "(" ParamValueList ")"
ParamValueList := Constant ( "," ParamValueList )?
IfInstruction   := <IF> "(" CompareExpression ")" <THEN>
              InstructionList ( <ELSE> InstructionList )? <ENDIF> ";"
CompareExpression := CalcExpression ( CompareOperator
              CompareExpression )?
CalcExpression := PrimaryExpression ( CalcOperator CalcExpression )?
PrimaryExpression := Constant
   | "(" CompareExpression ")"
CompareOperator   := "=="
   | "!="
   | ">"
   | "<"
   | ">="
   | "<="
   | "||"
   | "&&"
CalcOperator   := "|"
   | "&"
   | "+"
   | "-"
   | "*"
   | "/"
ForInstruction := <FOR> <IDENTIFIER> <IN> (
              ( IntValue ".." IntValue )
   | "(" VarList ")" ) <LOOP> InstructionList <ENDLOOP> ";"
WhileInstruction  := <WHILE> "(" CompareExpression ")" <LOOP>
              InstructionList <ENDLOOP> ";"
VarList  := Constant ( "," VarList )?
WaitInstruction   := <WAIT> "(" Constant ");"
WaitabsInstruction   := <WAITABS> "(" Constant ");"
```

```
ExecuteInstruction   := <EXEC> ( <WAIT> )? ( <LOG> )?
              <STRING_LITERAL> ";"
ReverseFailInstruction  := <REVERSE_FAIL> ";"
ParameterList  := Parameter ( "," Parameter )* ( "," )?
Parameter   := FieldName "=>" Constant
ChkParameterList  := ChkParameter ( "," ChkParameter )* ( "," )?
ChkParameter   := FieldName "=>" ( "!" )? Constant ( ":" Constant )?
ArgumentList   := Argument ( Argument )*
Argument := FieldName ":" FieldName ( ":=" Constant )? ";"
FieldName   := <IDENTIFIER> ( "[" <INTEGER_LITERAL> "]" )?
   ( ( "." FieldName ) )?
IntValue := <INTEGER_LITERAL>
    | "<<" <IDENTIFIER>
    | <IDENTIFIER>
FloatValue  := <FLOATING_POINT_LITERAL>
    | "<<" <IDENTIFIER>
    | <IDENTIFIER>
Constant := <INTEGER_LITERAL>
    | <FLOATING_POINT_LITERAL>
    | <CHARACTER_LITERAL>
    | <STRING_LITERAL>
    | ">>" <IDENTIFIER>
    | ">>" <JAVASCRIPT>
    | "<<" <IDENTIFIER> ( "." <IDENTIFIER> )?
    | <IDENTIFIER>
    | <UNIQID>
    | <JAVASCRIPT>
Script   := <JAVASCRIPT>
```

11 Contacts & Notices

11.1 Contacts

PrismTech Corporation
400 TradeCenter
Suite 5900
Woburn, MA
01801
USA
Tel: +1 781 569 5819

PrismTech Limited
PrismTech House
5th Avenue Business Park
Gateshead
NE11 0NG
UK
Tel: +44 (0)191 497 9900

PrismTech France
28 rue Jean Rostand
91400 Orsay
France
Tel: +33 (1) 69 015354

Web: http://www.prismtech.com

E-mail: info@prismtech.com

11.2 Notices