

OpenSplice DDS

Version 5.x

Getting Started Guide



OpenSplice DDS

GETTING STARTED GUIDE



Part Number: OS-GSG

Doc Issue 36, 18 August 2010

Copyright Notice

© 2010 PrismTech Limited. All rights reserved.

This document may be reproduced in whole but not in part.

The information contained in this document is subject to change without notice and is made available in good faith without liability on the part of PrismTech Limited or PrismTech Corporation.

All trademarks acknowledged.

A close-up, low-angle photograph of a computer keyboard, focusing on the central and right-hand keys. The keys are white with dark lettering. A white grid pattern is overlaid on the entire image, creating a sense of depth and perspective. The lighting is soft, highlighting the texture of the keys and the grid lines.

CONTENTS

Table of Contents

Preface

About the Getting Started Guide	ix
Contacts	x

About OpenSplice DDS

Chapter 1	Why OpenSplice DDS	3
	1.1 What is OpenSplice DDS?	3
	1.2 Why Use It?	3
Chapter 2	Product Details	5
	2.1 Key Components	5
	2.1.1 Services	5
	2.1.2 Tools	5
	2.2 Key Features	5
	2.3 Language Bindings	5
	2.4 Platforms	6

Using OpenSplice DDS

Chapter 3	Documentation	9
Chapter 4	Information Sources	11
	4.1 Product Information	11
	4.1.1 Knowledge Base	11
	4.1.2 Additional Technical Information	11
	4.2 Support	11

Installation and Configuration

Chapter 5	Installation and Configuration	15
	5.1 Prerequisites	15
	5.2 Installation for UNIX Platforms	15
	5.3 Installation for Windows Platforms	16

	5.4 Installation on other platforms	17
	5.5 Configuration	17
	5.6 Examples	19
	5.6.1 The PingPong Example	21
	5.6.2 The Tutorial Example	22
	5.6.3 Using the OpenSplice Tools	23
	5.7 Tailoring the C++ API	24
Chapter 6	Licensing OpenSplice	27
	6.1 General	27
	6.1.1 Development and Deployment Licenses	27
	6.2 Installing the License File	27
	6.3 Running the License Manager Daemon	28
	6.3.1 Utilities	29
 Platform-specific Information		
Chapter 7	VxWorks 5.5.1	33
	7.1 Building a kernel	33
	7.2 Building the Examples	34
	7.2.1 To build the simple PingPong example	34
	7.2.2 To build the Tutorial example	34
	7.3 Running the Examples	34
	7.3.1 How to start <i>spliced</i> and related services	35
	7.3.2 To run the C PingPong example from a shell within Tornado	35
	7.3.3 To run the C Tutorial from a shell within Tornado	36
Chapter 8	VxWorks 6.x	37
	8.1 Installation	37
	8.2 VxWorks Kernel Requirements	37
	8.3 Deploying OpenSplice DDS	38
	8.4 OpenSplice Examples	40
	8.4.1 Importing Example Projects into Workbench	40
	8.4.2 Building Example Projects with Workbench	40
	8.4.3 Deploying OpenSplice Examples	41
	8.4.3.1 Deploying PingPong	41
	8.4.3.2 Deploying the Chat Tutorial	42
Chapter 9	Integrity	43
	9.1 The <i>ospl_projgen</i> command	43
	9.1.1 Description of the arguments	43
	9.1.2 Using mmstat and shmdump diagnostic tools on Integrity	44
	9.2 PingPong Example	44

9.3 Changing the <i>ospl_projgen</i> arguments	47
9.3.1 Changing the generated OpenSplice DDS project using <i>Multi</i>	47
9.4 The <i>ospl_xml2int</i> tool	48
9.4.1 The <i>ospl_xml2int</i> command	49
9.4.2 Description of the arguments	49
9.5 Critical warning about <i>Object 10</i> and <i>Object 11</i>	51
9.6 Amending OpenSplice DDS configuration with <i>Multi</i>	52

Preface

About the Getting Started Guide

The *Getting Started Guide* is included with the OpenSplice DDS *Documentation Set*. This guide is the starting point for anyone using, developing or running applications with OpenSplice DDS.

The *Getting Started Guide* contains:

- general information about OpenSplice DDS
- a list of documents and how to use them
- initial installation and configuration information (detailed information is provided in the *User* and *Deployment Guides*)
- details of where additional information can be found, such as the OpenSplice FAQs, Knowledge Base, bug reports, etc.
- a *Bibliography* listing background information, recommended texts and reference material which users and developers may find useful

Intended Audience

The *Getting Started Guide* is intended to be used by anyone who wishes to use the *OpenSplice DDS* product.

Conventions

The conventions listed below are used to guide and assist the reader in understanding the *Getting Started Guide*.



Item of special significance or where caution needs to be taken.



Item contains helpful hint or special information.



Information applies to Windows (*e.g.* XP, 2003, Windows 7) only.



Information applies to Unix based systems (*e.g.* Solaris) only.



C language specific



C++ language specific



Java language specific

Hypertext links are shown as *[blue italic underlined](#)*.

On-Line (PDF) versions of this document: Items shown as cross references to other parts of the document, *e.g.* *Contacts* on page x, behave as hypertext links: users can jump to that section of the document by clicking on the cross reference.

```
% Commands or input which the user enters on the
command line of their computer terminal
```

Courier, **Courier Bold**, or *Courier Italic* fonts indicate programming code. The Courier font can also be used to indicate file names (in order to distinguish the file name from the standard text).

Extended code fragments are shown as small Courier font contained in shaded, full width boxes (to allow for standard 80 column wide text), as shown below:

```
NameComponent newName[] = new NameComponent[1];

// set id field to "example" and kind field to an empty string
newName[0] = new NameComponent ("example", "");
```

Italics and ***Italic Bold*** are used to indicate new terms, or emphasise an item.

Arial Bold is used to indicate user related actions, *e.g.* **File > Save** from a menu.

Step 1: One of several steps required to complete a task.

Contacts

PrismTech can be reached at the following contact points for information and technical support.

USA Corporate Headquarters

PrismTech Corporation
400 TradeCenter
Suite 5900
Woburn, MA
01801
USA

Tel: +1 781 569 5819

European Head Office

PrismTech Limited
PrismTech House
5th Avenue Business Park
Gateshead
NE11 0NG
UK

Tel: +44 (0)191 497 9900

Fax: +44 (0)191 497 9901

Web: <http://www.prismtech.com>
Technical questions: crc@prismtech.com (Customer Response Center)
Sales enquiries: sales@prismtech.com

A close-up, low-angle shot of a computer keyboard, likely a laptop, with a white grid overlay. The grid lines are thin and white, creating a pattern of squares and rectangles across the entire image. The keyboard keys are visible, with some characters like "!" and "1" being prominent. The overall color palette is a mix of light and dark purples and blues, giving it a modern, tech-oriented feel.

ABOUT OPENSPLICE DDS

CHAPTER

1 *Why OpenSplice DDS*

1.1 What is OpenSplice DDS?

The purpose of OpenSplice DDS is to provide an infrastructure and middleware layer for real-time distributed systems. This is a realisation of the *OMG-DDS-DCPS Specification for a Data Distribution Service* based upon a Data Centric Publish Subscribe architecture.

1.2 Why Use It?

OpenSplice DDS provides an infrastructure for real-time data distribution and offers middleware services to applications. It provides a real-time data distribution service that aims at:

- reducing the complexity of the real-time distributed systems
- providing an infrastructure upon which fault-tolerant real-time systems can be built
- supporting incremental development and deployment of systems

2 *Product Details*

2.1 Key Components

OpenSplice DDS's include the key components listed here.

2.1.1 Services

- *Domain Service* (`spliced`) - manages a DDS domain
- *Durability Service* - responsible for handling non-volatile data
- *Networking Service* - responsible for handling communication between a node and the rest of the nodes on 'the network'
- *Tuner Service* - responsible for providing control and monitoring functionality for OpenSplice DDS Systems

2.1.2 Tools

- *IDL Preprocessor* - generates topic types, type-specific readers and writers
- *OpenSplice Tuner* - provides monitor and control facilities on a specified DDS domain
- *OpenSplice Configurator* - simplifies the process for configuring the services

2.2 Key Features

- OpenSplice DDS is the most complete second generation OMG DDS implementation that supports all DCPS profiles.
- OpenSplice DDS is proven in the field for a decade of deployment in mission critical environments.
- Targets both real-time embedded and large-scale fault-tolerant systems.
- Highly optimised implementation from DDS users for DDS users.
- Total lifecycle support from prototyping through to remote maintenance.

2.3 Language Bindings

OpenSplice DDS is available for the following languages:

- C (standalone C)
- C++ (standalone C++ and CORBA C++)

- Java

OpenSplice can be used stand-alone for the C and Java languages, referred to as *StandAlone C* (SAC) and *StandAlone Java* (SAJ) respectively.

The C++ language binding, referred to as *CORBA C Plus Plus* (CCPP) can only be used in combination with a C++ ORB. The ORB and compiler combination used to generate the default CCPP library (supplied with the OpenSplice release) is mentioned in the *Release Notes*. See Section 5.7, *Tailoring the C++ API*, for additional information.

The C++ language binding, referred to as *StandAlone C Plus Plus* (SACPP) can be used without an ORB. The compiler used to generate the default library supplied with the OpenSplice Release is mentioned in the *Release Notes*.

C++ applications can be developed without a dependency to a C++ ORB by using the StandAlone C API (SAC) directly from the C++ application code.

2.4 Platforms

The platforms supported by OpenSplice DDS are listed in the *Release Notes*.

Please refer to *Platform-specific Information* (page 31 onwards) for information about using OpenSplice DDS on specific platforms.

A close-up, low-angle shot of a computer keyboard, likely a laptop, with a white grid overlay. The grid lines are thin and white, creating a pattern of squares and rectangles across the keyboard. The keyboard keys are visible, with some characters like "1", "2", and "3" being legible. The overall color palette is muted, with a mix of greys, whites, and a hint of blue. The text "USING OPENSPLICE DDS" is overlaid in a dark blue, serif font, positioned in the upper right quadrant of the image.

USING OPENSPLICE DDS

3 Documentation

The OpenSplice DDS documentation set provides detailed information about OpenSplice DDS, including its API, usage, installation and configuration.

The following table lists all of the documentation and manuals included with OpenSplice DDS. The table includes brief descriptions of the documents and their likely users.

OpenSplice DDS Documentation Set

Document	Description and Use
Release Notes	<p>Lists the latest updates, bug fixes, and last-minute information.</p> <p>For product installers, administrators, and developers, who need to be aware of the latest changes which may affect the Service's performance and usage.</p> <p>A link to the Release Notes is in <i>index.html</i> located in the directory where OpenSplice is installed.</p>
Getting Started Guide	<p>General information about OpenSplice, including installation instructions, initial configuration requirements and instructions on running the OpenSplice examples.</p> <p>For managers, administrators, and developers to gain an initial understanding of the product, as well as for product installers for installing and administering OpenSplice.</p>
Deployment Guide	A complete reference on how to configure and tune the OpenSplice service.
Tutorial Guide	A short course on developing applications with OpenSplice in C. Includes example code in C, C++ and Java.

OpenSplice DDS Documentation Set (Continued)

Document	Description and Use
Tuner Guide	Describes how to use the Tuner tool for monitoring and controlling OpenSplice. For programmers, testers, system designers and system integrators using OpenSplice.
IDL Pre-processor Guide	Describes how to use the OpenSplice IDL pre-processor for C, C++ and Java.
C Reference Guide C++ Reference Guide Java Reference Guide	Each of these reference guides describes the OpenSplice DDS Application Programmers Interface (API) for C, C++ and Java. This is a detailed reference for developers to help them to understand the particulars of each feature of the OpenSplice DDS API.
Examples	Examples, complete with source code, demonstrating how applications using OpenSplice can be written and used. The examples and related instructions are accessed through text files included with the product distribution.
White Papers and Data Sheets	Technical papers providing information about OpenSplice DDS. These technical papers are in Adobe Acrobat PDF™ format and can be obtained from the PrismTech web site at: http://www.prismtech.com

4 *Information Sources*

4.1 Product Information

Links to useful technical information for PrismTech's products, including the OpenSplice DDS and associated components, are listed below.



These links are provided for the reader's convenience and may become out-of-date if changes are made on the PrismTech Web site after publication of this guide. Nonetheless, these links should still be reachable from the main PrismTech Web page located at <http://www.prismtech.com>.

4.1.1 Knowledge Base

The PrismTech Knowledge Base is a collection of documents and resources intended to assist our customers in getting the most out of the OpenSplice products. The Knowledge Base has the most up-to-date information about bug fixes, product issues and technical support for difficulties that you may experience. The Knowledge Base can be found at:

<http://kb.prismtech.com>

4.1.2 Additional Technical Information


Information provided by independent publishers, newsgroups, web sites, and organisations, such as the Object Management Group, can be found on the Prismtech Web site:

<http://www.prismtech.com>

4.2 Support

PrismTech provides a range of product support, consultancy and educational programmes to help you from product evaluation and development, through to deployment of applications using OpenSplice DDS. The support programmes are designed to meet customers' particular needs and range from a basic *Standard* programme to the *Gold* programme, which provides comprehensive, 24 x 7 support.

Detailed information about PrismTech's product support services, general support contacts and enquiries are described on the *PrismTech Support* page reached via the PrismTech Home page at <http://www.prismtech.com>.

The background of the page is a close-up, low-angle photograph of a computer keyboard. The keys are white and slightly worn. A semi-transparent grid of thin white lines is overlaid on the entire image, creating a technical or digital aesthetic. The lighting is soft, highlighting the texture of the keys.

INSTALLATION AND CONFIGURATION

5 Installation and Configuration

Follow the instructions in this chapter to install and configure OpenSplice DDS and its tools. Information on running the OpenSplice examples are provided at the end of the chapter under Section 5.6, *Examples*.

5.1 Prerequisites

The prerequisites for OpenSplice DDS are given in the *Release Notes* included with your OpenSplice DDS product distribution. The *Release Notes* can be viewed by opening the *index.html* located in the root (or base) directory of your OpenSplice DDS installation and following the *Release Notes* link.

5.2 Installation for UNIX Platforms

Step 1: Install OpenSplice DDS

1. Ensure you have sufficient disk space
 - a) A minimum **60 MB** of free disk space is required **during installation** for the OpenSplice Host Development Environment (HDE) packages. This package contains all services, libraries, header-files and tools needed to develop applications using OpenSplice.
 - b) A minimum **35 MB** of free disk space is required **during installation** for the OpenSplice RunTime System (RTS) packages. This package contains all services, libraries and tools to deploy applications using OpenSplice.
2. Install OpenSplice DDS by running the installation wizard for your particular installation, using:

```
OpenSpliceDDS<version>-<platform>.<os>-<E>-installer.<ext>
```

where

<version> - the OpenSplice DDS version number, for example *V5.0*

<platform> - the platform architecture, for example *sparc* or *x86*

<os> - the operating system, for example *solaris8* or *linux2.6*

<E> - the environment, either *HDE* or *RTS*

<ext> - the platform executable extension, either *bin* or *exe*

The directories in the OpenSplice DDS distribution are named after the installation package they contain. Each package consists of an archive and its installation procedure.

Step 2: Configure the OpenSplice DDS environment variables

1. Go to the `<install_dir>/<E>/<platform>` directory, where `<E>` is HDE or RTS and `<platform>` is, for example, `x86.linux2.6`.
2. Source the `release.com` file from the shell command line.

```
% . ./release.com
```

This step performs all the required environment configuration.

Step 3: Install your desired ORB when the C++ language mapping is used with CORBA coexistence. Ensure your chosen ORB and compiler is appropriate for the CCPP library being used (either OpenSplice's default library or other custom-built library). Refer to the *Release Notes* for ORB and compiler information pertaining to OpenSplice DDS' default CCPP library.

5.3 Installation for Windows Platforms

Step 1: Install OpenSplice DDS

1. Ensure you have sufficient disk space
 - a) A minimum **60 MB** of free disk space is required **during installation** for the OpenSplice Host Development Environment (HDE) packages. This package contains all services, libraries, header-files and tools needed to develop applications using OpenSplice DDS.
 - b) A minimum **35 MB** of free disk space is required **during installation** for the OpenSplice RunTime System (RTS) packages. This package contains all services, libraries and tools to deploy applications using OpenSplice DDS.
2. Install OpenSplice DDS by running the installation wizard for your particular installation, using:

```
OpenSpliceDDS<version>-<platform>.<os>-<E>-installer.<ext>
```

where

`<version>` - the OpenSplice DDS version number, for example `V5.0`

`<platform>` - the platform architecture, for example `sparc` or `x86`

`<os>` - the operating system, for example `solaris8` or `linux2.6`

`<E>` - the environment, either `HDE` or `RTS`

`<ext>` - the platform executable extension, either `bin` or `exe`

The directories in the OpenSplice DDS distribution are named after the installation package they contain. Each package consists of an archive and its installation procedure.

Step 2: Install your desired ORB when the C++ language mapping is used with CORBA coexistence. Ensure your chosen ORB and compiler is appropriate for the CCPP library being used (either OpenSplice's default library or other custom-built library). Refer to the *Release Notes* for ORB and compiler information pertaining to OpenSplice DDS' default CCPP library.

5.4 Installation on other platforms

Please refer to *Platform-specific Information* (page 31 onwards) for information about using OpenSplice DDS on specific platforms.

5.5 Configuration

OpenSplice DDS is configured using an XML configuration file, as shown under *XML Configuration Settings* on page 18.

The default configuration file is `ospl.xml` located in `$OSPL_HOME/etc/config`. The default value of the environment variable `OSPL_URI` is set to this configuration file.

The configuration file defines and configures the following OpenSplice services:

- *spliced* - the default service, also called the *domain service*; the domain service is responsible for starting and monitoring all other services
- *durability* - responsible for storing non-volatile data and keeping it consistent within the domain (optional)
- *networking* - realizes user-configured communication between the nodes in a domain
- *tuner* - provides a SOAP interface for the OpenSplice Tuner to connect to the node remotely from any other *reachable* node

The default *Database Size* that is mapped on a shared-memory segment is 10 Megabytes.

The maximum user-creatable shared-memory segment is limited on certain machines, including Solaris, so it must either be adjusted or OpenSplice must be started as root.

A complete configuration file that enables durability as well as networking is shown below. The bold parts are not enabled in the default configuration file, but editing them will allow you to enable support for `PERSISTENT` data (instead of just `TRANSIENT` or `VOLATILE` data) and to use multicast instead of broadcast.

Adding support for `PERSISTENT` data requires you to add the `<Persistent>` element to the `<DurabilityService>` content (see the bold lines in the XML example shown below). In this `<Persistent>` element you can then specify the actual path to the directory for persistent-data storage (if it does not exist, the directory will be created). In the example below this directory is `/tmp/Pdata`.

For the networking service, the network interface-address that is to be used is specified by the `<NetworkInterfaceAddress>` element. The default value is set to *first available*, meaning that OpenSplice will determine the first available interface that is broadcast or multicast enabled. However, an alternative address may be specified as well (specify as *a.b.c.d*).

The network service may use separate channels, each with their own name and their own parameters (for example the port-number, the queue- size, and, if multicast enabled, the multicast address). Channels are either reliable (all data flowing through them is delivered reliably on the network level, regardless of QoS settings of the corresponding writers) or not reliable (all data flowing through them is delivered at most once, regardless of QoS settings of the corresponding writers). The idea is that the network service chooses the most appropriate channel for each DataWriter, i.e. the channel that fits its QoS settings the best.

Usually, networking shall be configured to support at least one reliable and one non-reliable channel. Otherwise, the service might not be capable of offering the requested reliability. If the service is not capable of selecting a correct channel, the message is sent through the “default” channel. The example configuration defines both a reliable and a non-reliable channel.

The current configuration uses broadcast as the networking distribution mechanism. This is achieved by setting the `Address` attribute in the `GlobalPartition` element to broadcast, which happens to be the default value anyway. This `Address` attribute can be set to any multicast address in the notation *a.b.c.d* in order to use multicast.



If *multicast* is required to be used instead of *broadcast*, then the operating system’s multicast routing capabilities must be configured correctly.

See the *OpenSplice DDS Deployment Manual* for more advanced configuration settings.

Example XML Configuration Settings

```
<OpenSpliceDDS>
  <Domain>
    <Name>OpenSpliceDDSV3.3</Name>
    <Database>
      <Size>10485670</Size>
    </Database>
    <Lease>
      <ExpiryTime update_factor="0.5">5.0</ExpiryTime>
    </Lease>
  </Domain>
</OpenSpliceDDS>
```

```

</Lease>
<Service name="networking">
  <Command>networking</Command>
</Service>
<Service name="durability">
  <Command>durability</Command>
</Service>
</Domain>
<NetworkService name="networking">
  <General>
    <NetworkInterfaceAddress>
      first available
    </NetworkInterfaceAddress>
  </General>
  <Partitioning>
    <GlobalPartition Address="broadcast"/>
  </Partitioning>
  <Channels>
    <Channel name="BestEffort" reliable="false"
      default="true">
      <PortNr>3340</PortNr>
    </Channel>
    <Channel name="Reliable" reliable="true">
      <PortNr>3350</PortNr>
    </Channel>
  </Channels>
</NetworkService>
<DurabilityService name="durability">
  <Network>
    <InitialDiscoveryPeriod>2.0</InitialDiscoveryPeriod>
    <Alignment>
      <RequestCombinePeriod>
        <Initial>2.5</Initial>
        <Operational>0.1</Operational>
      </RequestCombinePeriod>
    </Alignment>
    <WaitForAttachment maxWaitCount="10">
      <ServiceName>networking</ServiceName>
    </WaitForAttachment>
  </Network>
  <NameSpaces>
    <NameSpace durabilityKind="Durable"
      alignmentKind="Initial_and_Aligner">
      <Partition>*</Partition>
    </NameSpace>
  </NameSpaces>
  <Persistent>
    <StoreDirectory>/tmp/Pdata</StoreDirectory>
  </Persistent>
</DurabilityService>
</OpenSplice>

```

5.6 Examples

The way to build and run the examples is dependent on the Platform you are using. For VxWorks and Integrity, please refer to Chapter 8, *VxWorks 6.x*, on page 37, and Chapter 9, *Integrity*, on page 43 in this Guide.

The C and C++ examples on Unix/Linux-based systems have makefiles, while the Java examples have *BUILD* shell scripts.

For Windows use the following procedure:

Step 1: *Set TAO environment*

Required only if the C++ cohabitation examples are to build:

```
set TAO_ROOT=<install TAO path>
set PATH=%TAO_ROOT%\bin;%PATH%
```

Step 2: *Set Microsoft Visual Studio Environment using VS supplied batch file*

For VS 2008:

```
C:\ospl\HDE\X86~1.WIN\examples\dcps>"c:\Program Files\
Microsoft Visual Studio 9.0\Common7\Tools\vsvars32.bat"
```

For VS 2005:

```
C:\ospl\HDE\X86~1.WIN\examples\dcps>"c:\Program Files\
Microsoft Visual Studio 8\Common7\Tools\vsvars32.bat"
```

Step 3: *Set OSPL runtime environment*

```
cd <wherever>\HDE\x86.win32
release.bat
```

For example:

```
C:\ospl\HDE\X86~1.WIN>release.bat
"<<< OpenSplice HDE Release V4.1.1 For x86.win32, Date
2009-06-04 >>>"
```

Step 4: *Change to the examples directory*

```
C:\ospl\HDE\X86~1.WIN>cd examples
```

Step 5: *Build examples*

Either open the solution file and build using the Studio IDE:

```
C:\ospl\HDE\X86~1.WIN\examples>devenv /useenv examples.sln
```

Or build from the command line:

```
C:\ospl\HDE\X86~1.WIN\examples>devenv /useenv examples.sln
/Build Release
```

For debugging purposes, replace Release with Debug:

```
C:\ospl\HDE\X86~1.WIN\examples>devenv /useenv examples.sln
/Build Debug
```


5.6.1 The PingPong Example

The *PingPong* example is a small benchmarking program that bounces a topic back and forth between *ping* and a *pong* applications. It measures and displays the round-trip times of these topics, giving a first impression on some performance characteristics of the product. Command line parameters control things like the payload for the topics and partitions in which they write and read their information.

Step 1: Build the example applications

For the StandAlone C API (SAC):

1. Change to the `$OSPL_HOME/examples/dcps/standalone/C/PingPong` directory (on Windows use the command prompt).
2. Check the usage and functioning by inspecting the `ping.c` file.
3. Build the applications using `make` on the command line on Linux/Solaris platforms. On Windows, follow the procedure starting at *Step 1* on page 20.
4. Run the applications running `sh ./RUN` on the command line on Linux/Solaris platforms or execute `RUN.bat` within the command prompt on the Windows platform.

For the CORBA-coexistent C++ API (CCPP):

1. Change to the
`$OSPL_HOME/examples/dcps/CORBA/C++/OpenFusion/PingPong`
directory (on Windows use the command prompt).
2. Check the usage and functioning by inspecting the `ping.cpp` file.
3. Build the applications using `make` on the command line on Linux/Solaris platforms. On Windows, follow the procedure starting at *Step 1* on page 20.
4. Run the applications running `sh ./RUN` on the command line on Linux/Solaris platforms or execute `RUN.bat` in a command prompt on the Windows platform.

For the StandAlone Java API (SAJ)

1. Change to the
`$OSPL_HOME/examples/dcps/standalone/Java/PingPong`
directory (on Windows use the command prompt).
2. Check the usage and functioning by inspecting the `ping.java` file.
3. Build the applications using `sh ./BUILD` on the command line on Linux/Solaris platforms or execute `BUILD.bat` in a command prompt on Windows.
4. Run the applications running `sh ./RUN` on the command line on Linux/Solaris platforms or execute `RUN.bat` in a command prompt on Windows.

For the StandAlone C++ API (C++ using SAC):

1. Change to the `$OSPL_HOME/examples/dcps/standalone/C++/PingPong` directory (on Windows use the command prompt).
2. Check the usage and functioning by inspecting the `ping.cpp` file.
3. Build the applications using `make` on the command line on Linux/Solaris platforms. On Windows, follow the procedure starting at *Step 1* on page 20.
4. Run the applications running `sh ./RUN` on the command line on Linux/Solaris platforms or execute `RUN.bat` in a command prompt on Windows.

For the CORBA-coexistent Java API (CCJ):

1. Change to the directory
`$OSPL_HOME/examples/dcps/CORBA/Java/JacORB/PingPong`
(on Windows use the command prompt).
2. Check the usage and functioning by inspecting the `ping.java` file.
3. Build the applications using `sh ./BUILD` on the command line on Linux/Solaris platforms or execute `BUILD.bat` in a command prompt on Windows. Note that the environment variable `JACORB_HOME` is used in the `BUILD` scripts so it must be set before they are run.
4. Run the applications running `sh ./RUN` on the command line on Linux/Solaris platforms or execute `RUN.bat` in a command prompt on the Windows platform.

For the StandAlone C# API (SACS):

1. Change to the `$OSPL_HOME/examples/dcps/standalone/CS/PingPong` directory (on Windows use the command prompt).
2. Check the usage and functioning by inspecting the `ping.cs` file.
3. Build the applications using `make` on the command line on Linux/Solaris platforms. On Windows, follow a procedure similar to the one explained at *Step 1* on page 20, but then for the solution file `$OSPL_HOME/examples/dcps/standalone/CS/PingPong.sln..`
4. Run the applications running `sh ./RUN` on the command line on Linux/Solaris platforms or execute `RUN.bat` in a command prompt on Windows.

5.6.2 The Tutorial Example

The tutorial example consists of three separate executables that constitute a very primitive Chatter application. The OpenSplice infrastructure should manually be started before running the Chatter applications executables (see Section 5.6.3, *Using the OpenSplice Tools*). Each Chatter executable can be started using different configurations provided each is running in their own, separate terminal window: this is to avoid having their screen outputs intermingled together in the same terminal window.

The Chatter executables, *Chatter*, *MessageBoard* and *UserLoad*, are run from the command line as shown below.

Chatter(.exe) [userID] [userName]

Starts sending Chat messages for a user with the specified ID and name. Ensure the ID number is a unique and not used by any other user.

MessageBoard(.exe) [userID]

Starts a MessageBoard that displays the Chat messages it receives from all users, except for the user with the specified ID.

UserLoad(.exe)

Starts an application that monitors new users who log on to the chat session and currently connected users who log off. *UserLoad* prints its monitoring information to the terminal for one minute then terminates.

You can experiment and observe the effects by running different Chatters with different ID's, using different MessageBoards and observing how each executable interacts with the others. In particular, examine what happens if a MessageBoard is started after one or more Chatters have already begun to send their messages.

Building the Tutorial examples is very similar to building the PingPong examples: the C and C++ examples on Unix/Linux-based systems have makefiles, while the Java examples use *BUILD* scripts that compile and link the executables; for Windows, follow the procedure starting at *Step 1* on page 20. Also, Project files are available for Microsoft Visual Studio for C and C++.

When running the examples in Java, ensure the correct CLASSPATH variable value is used, for example:

On Unix or Linux-based platforms use:

```
% java -classpath $OSPL_HOME/jar/dcpssaj.jar:.
```

On Windows-based platforms use:

```
> java -classpath %OSPL_HOME%\jar\dcpssaj.jar:.
```

5.6.3 Using the OpenSplice Tools

i

The *RUN(.bat)* command for the PingPong examples also starts and stops the OpenSplice infrastructure. The tutorial examples (located in the *Tutorial* directory that is on the same level as the *PingPong* example directory of the specific language binding) can be built in the same way, but they do not provide a *RUN(.bat)* script: instead each executable should be manually started in a separate terminal window (to avoid mixing up their screen output) and the OpenSplice infrastructure should be started and terminated manually.

Step 2: Manually start the OpenSplice infrastructure

1. Enter `ospl start` on the command line.¹ This starts the OpenSplice services.
2. The default configuration file that comes with OpenSplice, which is network enabled, is used.
 - a) The default configuration uses UDP-broadcast.
 - b) UDP-multicast can be used instead of UDP-broadcast, but requires that a multicast address is added in the configuration file (see Section 5.5, *Configuration*, on page 17).
3. These log files may be created in the current directory when OpenSplice is started:
 - a) `ospl-info.log` - contains information and warning-reports
 - b) `ospl-error.log` - contains error reports

Step 3: Start the OpenSplice Tuner Tool

1. Read the *OpenSplice Tuner Guide* (*TurnerGuide.pdf*) before running the Tuner Tool
2. Start the tool by entering `ospltun` on the command line.



The `URI` required to connect is set in the `OSPL_URI` environment variable (default `URI` is: `file://$OSPL_HOME/etc/config/ospl.xml`).

3. The OpenSplice system can now be monitored.

Step 4: Experiment with the OpenSplice tools and applications

1. Start the C, C++ and Java publishers and subscribers and observe how they perform
2. Use the OpenSplice Tuner to monitor all DDS entities and their (dynamic) relationships

Step 5: Manually stop the OpenSplice infrastructure

1. Choose **File > Disconnect** from the OpenSplice Tuner menu.
2. Enter `ospl stop` on the command line: this stops all OpenSplice services.

5.7 Tailoring the C++ API

The pre-compiled C++ API that is delivered with OpenSplice DDS only works in combination with a specific ORB and compiler combination, as stated previously. If another compiler version or another ORB is required, then a custom C++ API library can be built using the source code provided in the

1. `ospl` is the command executable for OpenSplice.

`$OSPL_HOME/custom_lib/ccpp` directory. Detailed instructions on how to create the custom API are provided in the `README.txt` file located in `$OSPL_HOME/custom_lib/ccpp`.

6 Licensing OpenSplice

6.1 General

OpenSplice DDS uses *FLEXNet* to manage licenses. This section describes how to install a license file for OpenSplice DDS and how to use the license manager.

The licensing software is automatically installed on the host machine as part of the OpenSplice distribution. The software consists of two parts:

- OpenSplice DDS binary files, which are installed in `<OpenSplice_Install_Dir>/<E>/<platform>.<os>/bin`, where *OpenSplice_Install_Dir* is the directory where OpenSplice DDS is installed
- License files which determine the terms of the license. These will be supplied by PrismTech.



Licenses: PrismTech supplies an OpenSplice DDS license file, `license.lic`. This file is *not* included in the software distribution, but is sent separately by PrismTech.

6.1.1 Development and Deployment Licenses

Development licenses are on a *per Single Named Developer* basis. This implies that each developer using the product requires a license. OpenSplice DDS is physically licensed for development purposes. OpenSplice DDS is also physically licensed on enterprise platforms for deployment.



The OpenSplice Tuner is sold as a separate product. The development license only includes IDL pre-processor support.

6.2 Installing the License File

Copy the license file to `<OpenSplice_Install_Dir>/etc/license.lic`, where `<OpenSplice_Install_Dir>` is the directory where OpenSplice is installed, on the machine that will run the license manager.

This is the recommended location for the license file but you can put the file in any location that can be accessed by the license manager `lmgrd`.

If another location is used or the environment has not been setup, then an environment variable, either `LM_LICENSE_FILE` or `PTECH_LICENSE_FILE`, must be set to the full path and filename of the license file (either variable can be set; there is no need to set both). For example:

```
PTECH_LICENSE_FILE=/my/lic/dir/license.lic
```

If licenses are distributed between multiple license files, the `LM_LICENSE_FILE` or `PTECH_LICENSE_FILE` variable can be set to point to the directory which contains the license files.

6.3 Running the License Manager Daemon

It is only necessary to run the License Manager Daemon for floating or counted licenses. In this case, the license manager must be running before OpenSplice DDS can be used. The license manager software is responsible for allocating licenses to developers and ensuring that the allowed number of concurrent licenses is not exceeded.

For node-locked licenses, as is the case with all evaluation licenses, then it is not necessary to run the License Manager Daemon but the `LM_LICENSE_FILE` or `PTECH_LICENSE_FILE` variable must be set to the correct license file location.

To run the license manager, use the following command:

```
% lmgrd -c <location>
```

where `<location>` is the full path and filename of the license file. If licenses are distributed between multiple files, `<location>` should be the path to the directory that contains the license files.

The `lmgrd` command will start the PrismTech vendor daemon `PTECH`, which controls the licensing of the OpenSplice DDS software.

To obtain a license for OpenSplice DDS from a License Manager Daemon that is running on a different machine, set either the `LM_LICENSE_FILE` or `PTECH_LICENSE_FILE` environment variable to point to the License Manager Daemon, using the following syntax:

```
% LM_LICENSE_FILE=<port>@<host>
```

where `<port>` is the port the daemon is running on and `<host>` is the host the daemon is running on.

The port and host values can be obtained from the information output when the daemon is started. The format of this output is as shown in the following example:

```
1: 9:55:03 (lmgrd) lmgrd tcp-port 27001
2: 9:55:03 (lmgrd) Starting vendor daemons ...
3: 9:55:03 (lmgrd) Started PTECH (internet tcp_port xxxxx pid
xxxxx)
4: 9:55:03 (PTECH) FLEXlm version 9.2
5: 9:55:04 (PTECH) Server started on ultra5 for: licensedobj1
licensedobj2
```


The <port> value should be taken from the first line of the output. The <server> value should be taken from the last line. From this example, the value for `LM_LICENSE_FILE` or `PTECH_LICENSE_FILE` would be:

```
27001@ultra5
```

6.3.1 Utilities

A utility program, `lmutil`, is available for license server management and administration. One feature of this utility is its ability to gracefully shut down the license manager. To shut down the license manager, preventing the checkout of licenses for the OpenSplice DDS software, run either of the following commands:

```
% lmutil lmdown -vendor PTECH
```

```
% lmutil lmdown -c <location>
```

where <location> is the full path and filename of the license file.

The `lmutil` program is also used to generate a host identification code which is used to generate your license key. To generate the code, run the following command on the license server:

```
% lmutil lmhostid
```

This returns an ID code for the server, which will look similar to:

```
8ac86d5
```

This ID code must be supplied to PrismTech so that your license key can be generated.

WIN

The OpenSplice DDS licensing software also includes `lmtools`, a GUI front end to the `lmutil` utility program.

A close-up, low-angle shot of a computer keyboard, likely a laptop, with a white grid overlay. The grid lines are thin and white, creating a pattern of squares and rectangles across the entire image. The keyboard keys are visible, with some characters like "!" and "1" clearly shown. The overall color palette is a mix of light and dark purples and blues, giving it a modern, tech-oriented feel.

PLATFORM-SPECIFIC INFORMATION

CHAPTER

7

VxWorks 5.5.1

This chapter provides a brief description of how to build the kernel and the supplied examples, and how to run those examples, using VxWorks 5.5.1 and the Tornado 'front end'. For more information about VxWorks 5.5.1 and Tornado, please refer to WindRiver's documentation.



NOTE: The examples given here assume that a Windows-hosted system is being used, and they are shown with OpenSplice DDS installed in C:\OpenSpliceDDS, as Tornado does not support spaces in some paths used.

7.1 Building a kernel

Required modules

The following modules are the core system components needed to build the OpenSpliceDDS runtime:

Operating system components

- POSIX components
 - POSIX timers
 - POSIX threads
- File System and Disk Utilities
 - File System and Disk Utilities

Additional modules

The modules listed below are extra ones that are useful for HDE (Host Development Environment) development. These modules are required if you are deploying from the Tornado front end:

Development tool components

- WDB agent components
 - WDB agent services
- WDB target server file system
 - symbol table components
 - synchronize host and target symbol labels

- target shell components
 - target shell

7.2 Building the Examples

Start the OpenSpliceDDS command prompt by clicking **Start > Programs > OpenSpliceDDS menu entry > OpenSpliceDDS command prompt**, then set up the Tornado shell by running `torVars.bat` with the command `C:\Tornado2.2\host\x86-win32\bin\torVars.bat`.



NOTE: The path will be different if you have installed Tornado somewhere other than in the default directory.

7.2.1 To build the simple PingPong example

At the prompt, `cd` to `examples\dcps\standalone\C\PingPong` and run `make`.

If the operation is successful, the `exec` directory will contain `ping.out`, `pong.out` and `PingPong-SharedLib.out`.

7.2.2 To build the Tutorial example

At the prompt, `cd` to `examples\dcps\standalone\C\Tutorial` and run `make`.

If the operation is successful, the `exec` directory will contain `Chatter.out`, `MessageBoard.out`, `UserLoad.out` and `Chatter-SharedLib.out`.

7.3 Running the Examples

If you included the additional modules listed above (see 7.1, *Building a kernel*) in your kernel, deployment is done *via* the target server setup from the Tornado shell connection.

For the example below, the target server has been created with the target server file system set to `C:\OpenSpliceDDS\<VersionNumber>\HDE\x86.vxworks5.5`, and it must be mounted to `/tgtsvr` on the target.

All OpenSplice DDS tools or services have unique entry points. These entry points all take a string; the string is parsed into the necessary arguments and passed on. To start `ospl` on a linux system, the command would be:

```
ospl start file:///ospl.xml
```

and on VxWorks it would be:

```
ospl "start file:///ospl.xml"
```

Note that the arguments are separated by spaces.

Other commands:

```
ospl -> ospl(char *)
spliced -> ospl_spliced(char *)
```

```
networking -> ospl_networking(char *)
durability -> ospl_durability(char *)
mmstat -> ospl_mmstat(char *)
shmdump -> ospl_shmdump(char *)
```

The standard ‘main’ equivalent entry points are:

```
ospl -> ospl_unique_main(int argc, char ** argv)
spliced -> ospl_spliced_unique_main(int argc, char ** argv)
networking -> ospl_networking_unique_main(int argc, char ** argv)
durability -> ospl_durability_unique_main(int argc, char ** argv)
mmstat -> ospl_mmstat_unique_main(int argc, char ** argv)
shmdump -> ospl_shmdump_unique_main(int argc, char ** argv)
```

You can use the standard argv argc version entry when you need to use arguments with embedded spaces. For example, for ospl you would use:

```
osplArgs = malloc(12)
*osplArgs = "ospl"
*(osplArgs+4) = "start"
*(osplArgs+8) = "file:///tgtsvr/etc/config/ospl.xml"
ospl_unique_main (2, osplArgs)
```

7.3.1 How to start **spliced** and related services

To start the spliced service, configure the xml file and load the core dds shared lib that is needed by all OpenSplice DDS applications. You can do this on as many boards as needed.

Please be patient! Use the **i** command within the shell to see when the tasks are completed.

```
cd "C:\OpenSpliceDDS\<VersionNumber>\HDE\x86.vxworks5.5"
ld 1,0,"lib/libddscore.so"
ld 1,0,"bin/ospl"
os_putenv("OSPL_URI=file:///tgtsvr/etc/config/ospl-1.xml")
os_putenv("PATH=/tgtsvr/bin")
ospl("start")
```

7.3.2 To run the C PingPong example from a shell within Tornado

After the spliced and related services have started, you can start Pong (the publisher side of the demo):

```
ld 1,0,"lib/libdcpsgapi.so"
ld 1,0,"lib/libdcpssac.so"
ld 1,0,"examples/dcps/standalone/C/PingPong/exec/
PingPong-SharedLib.out"
ld 1,0,"examples/dcps/standalone/C/PingPong/exec/Pong.out"
pong("PongRead PongWrite")
```

After the Pong application has started you can start Ping, which is the subscriber side.

If you are working on one board and running locally, just give the commands as below (with a single address space, all the shared functionality has already been loaded in previous steps).

However, if you are running the `Pong` application on another board you must load and start `spliced` as described in 7.3.1, *How to start `spliced` and related services*, then load the two language-specific APIs `libdcpsgapi.so` and `lib/libdcpspac.so`, and then the shared library for the example `PingPong-SharedLib.out`.

```
ld 1,0,"examples/dcps/standalone/C/PingPong/exec/Ping.out"
ping("100 100 m PongRead PongWrite")
ping("100 100 q PongRead PongWrite")
ping("100 100 s PongRead PongWrite")
ping("100 100 b PongRead PongWrite")
ping("100 100 f PongRead PongWrite")
ping("1 10 t PongRead PongWrite")
```

7.3.3 To run the C Tutorial from a shell within Tornado

After the `spliced` and related services have started, you can start `UserLoad`:

```
ld 1,0,"lib/libdcpsgapi.so"
ld 1,0,"lib/libdcpspac.so"
ld 1,0,"examples/dcps/standalone/C/Tutorial/exec/
Chatter-SharedLib.out"
ld 1,0,"examples/dcps/standalone/C/Tutorial/exec/UserLoad.out"
UserLoad
```

After `UserLoad` has started you can start `MessageBoard` on the same board in another shell:

```
ld 1,0,"examples/dcps/standalone/C/Tutorial/exec/
MessageBoard.out"
MessageBoard
```

After the `MessageBoard` application has started you can start the `Chatter` application. As with the `PingPong` example, you can start this on the same board from another shell or on another board after starting `spliced`.

```
ld 1,0,"examples/dcps/standalone/C/Tutorial/exec/Chatter.out"
Chatter("1 John")
```

When you want to terminate the application, use:

```
Chatter("-1")
```


CHAPTER

8 VxWorks 6.x

OpenSplice DDS is deployed on the VxWorks 6.x operating system as Real Time Processes (RTPs). For more information about RTPs please refer to WindRiver's VxWorks documentation.

8.1 Installation



The following instructions describe installing OpenSplice DDS for VxWorks 6.x on the Windows host environment.

Start the installation process by double-clicking the OpenSplice DDS Host Development Environment (HDE) installer file. Follow the on-screen instructions and complete the installation. When asked to configure the installation with a license file, select **No**. The installer will create an OpenSplice DDS entry in **Start > Programs** which contains links to the OpenSplice tools, documentation, and an Uninstall option.



Please note that WindRiver's Workbench GUI must be run in an environment where the OpenSplice variables have already been set. If you chose to set the OpenSplice variables globally during the installation stage, then Workbench can be run directly. Otherwise, Workbench must be run from the OpenSplice DDS command prompt. Start the command prompt by clicking **Start > Programs > OpenSpliceDDS menu entry > OpenSpliceDDS command prompt**, then start the Workbench GUI. On VxWorks 6.6 the executable is located at

```
<WindRiver root directory>\workbench-3.0\wrwb\platform\
eclipse\wrwb-x86-win32.exe
```

This executable can be found by right-clicking on the WindRiver's Workbench **Start** menu item and selecting **Properties**.

8.2 VxWorks Kernel Requirements

The VxWorks kernel required to support OpenSplice DDS on VxWorks 6.x is built using the development kernel configuration profile with the additional posix thread components enabled. A kernel based on this requirement can be built within Workbench, by starting the Workbench GUI and selecting **File > New > VxWorks Image Project**.

Type a name for the project then select the appropriate Board Support Package and Tool Chain (for example `mcpn805` and `gnu`). Leave the kernel options to be used as blank, and on the **Configuration Profile** dialog select **PROFILE_DEVELOPMENT** from the drop-down list.

Once the kernel configuration project has been generated, the additional required functionality can be enabled:

- POSIX threads (`INCLUDE_POSIX_PTHREADS`)
- POSIX thread scheduler in RTPs (`INCLUDE_POSIX_PTHREAD_SCHEDULER`)
- built-in symbol table (`INCLUDE_STANDALONE_SYM_TBL`)

Note that the Workbench GUI should be used to enable these components so that dependent components are automatically added to the project.

8.3 Deploying OpenSplice DDS

As described in Section 5.5, *Configuration*, OpenSplice DDS is started with the OpenSplice domain service `spliced` and a number of optional services described within the OpenSplice configuration file (`ospl.xml`). On VxWorks 6.x, a Real Time Process for each of these services is deployed on to the target hardware. The sample `ospl.xml` configuration file provided with the VxWorks 6.x edition of OpenSplice has particular settings so that these RTPs can operate effectively.

The instructions below describe how to deploy these RTPs using the Workbench GUI and the Target Server File System (TSFS), although the processes can be deployed by using commands and other file system types.

- Step 1:** Start the Workbench and create a connection to the target hardware using the **Remote Systems** view.
- Step 2:** Create a connection to the host machine. In the **Properties** for the connection, make part of the host's file system available to VxWorks using the TSFS by specifying both the `-R` and `-RW` options to `tgtsvr`. For example, connecting with the option `-R c:\x -RW` will enable read and write access to the contents of the `c:\x` directory from the target hardware under the mount name `/tgtsvr`.
- Step 3:** Activate the new connection by selecting it and clicking **Connect**.
- Step 4:** With a connection to the target hardware established, create a new RTP deployment configuration for the connection by right-clicking on the connection and selecting **Run > Run RTP on Target...**
- Step 5:** Create a new configuration for the `spliced` deployment that points to the `spliced.vxe` executable from the OpenSplice installation. The following parameters should be set in the dialog:

RTP configuration for spliced.vxe

Exec Path on Target	/tgtsvr/spliced.vxe
Arguments	file:///tgtsvr/ospl.xml
Environment	LD_BIND_NOW=1 OSPL_URI=file:///tgtsvr/ospl.xml PATH=/tgtsvr
Priority	100
Stack Size	0x10000

For simplicity it has been assumed that `spliced.vxe` and the other executables (located in the `bin` directory of the installation) and `ospl.xml` (located in the `etc/config` directory of the installation) have been copied to the directory made available as `/tgtsvr` described above. It is possible, if required, to copy the entire OpenSplice installation directory to the `/tgtsvr` location so that all files are available, but please be aware that log and information files will be written to the same `/tgtsvr` location when the `spliced.vxe` is deployed.

The screen shot from Workbench in *Figure 1* shows this configuration.

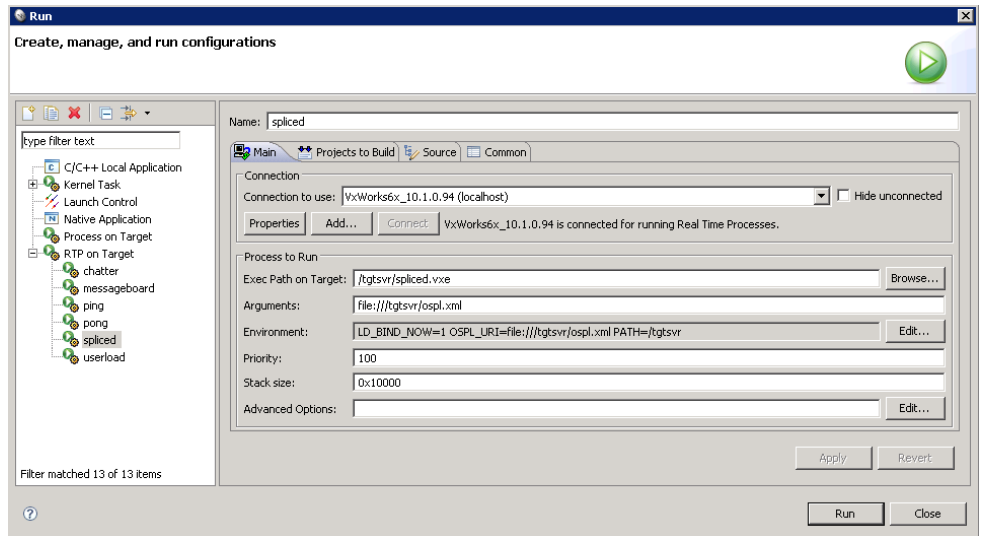


Figure 1 Workbench showing spliced deployment configuration

The configuration can be deployed by clicking **Run**, where an RTP for each service described in the configuration file should be created. These can be seen in Workbench in the Real Time Processes list for the target connection. An example is shown below in *Figure 2*. (The list may need to be refreshed with the **F5** key.)

Deployment problems are listed in `ospl-error.txt` and `ospl-info.txt`, which are created in the `/tgtsvr` directory if the configuration described above is used.

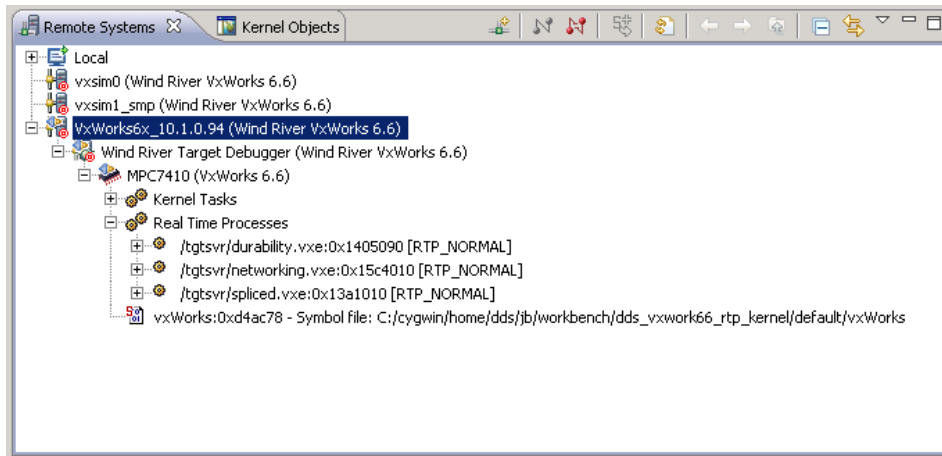


Figure 2 Workbench showing deployed OpenSplice RTPs

8.4 OpenSplice Examples

PrismTech provides PingPong and Tutorial examples both for C and C++ that are described in Section 5.6, *Examples*, on page 19. These example are provided in the form of Workbench projects which can be easily built and then deployed on to the target hardware in a similar process to above.

The projects are named `Dcps-PingPong-c` (and `-cpp`) and `Dcps-Tutorial-c` (and `-cpp`). Each project contains a `README` file briefly explaining the example and the parameters required to run it.

8.4.1 Importing Example Projects into Workbench

The example projects can be imported into Workbench by clicking **File > Import... > General > Existing Projects into Workspace**.

In the **Import Projects** dialog, browse to the `examples` directory of the OpenSplice installation. Select the required projects for importing from the list that Workbench has detected.

Ensure that the **Copy projects into workspace** box is checked, so that any changes made to these projects do not affect the initial installation.

If prompted to convert project files to the new Workbench format, select **Yes**.

8.4.2 Building Example Projects with Workbench

Projects in a workspace can be built individually or as a group.

Build a single project by selecting it and then click **Project > Build Project**.

Build all projects in the current workspace by clicking **Project > Build All**.

8.4.3 Deploying OpenSplice Examples

The PingPong and the Tutorial examples are run in identical ways with the same parameters for both C and C++. These should be deployed onto the VxWorks target with the arguments described in the README files for each project.

8.4.3.1 Deploying PingPong

The PingPong example consists of the `ping.vxe` and `pong.vxe` executables. If these executables have been copied to the directory made available as `/tgtsvr` as described in Section 8.3, *Deploying OpenSplice DDS*, RTP configurations should have the following parameters:

RTP configuration for pong

Exec Path on Target	/tgtsvr/pong.vxe
Arguments	PongRead PongWrite
Environment	LD_BIND_NOW=1 OSPL_URI=file:///tgtsvr/ospl.xml
Priority	100
Stack Size	0x10000

RTP configuration for ping

Exec Path on Target	/tgtsvr/ping.vxe
Arguments	10 10 s PongRead PongWrite
Environment	LD_BIND_NOW=1 OSPL_URI=file:///tgtsvr/ospl.xml
Priority	100
Stack Size	0x10000

When deployment is successful, the console shows output from both the `ping` and `pong` executables. The console view can be switched to show the output for each process by clicking the **Display Selected Console** button.

8.4.3.2 Deploying the Chat Tutorial

The Chat Tutorial consists of the `chatter.vxe`, `messageboard.vxe` and `userload.vxe` executables. If these executables have been copied to the directory made available as `/tgtsvr` as described in Section 8.3, *Deploying OpenSplice DDS*, RTP configurations should have the following parameters:

RTP configuration for userload

Exec Path on Target	/tgtsvr/userload.vxe
Arguments	
Environment	LD_BIND_NOW=1 OSPL_URI=file:///tgtsvr/ospl.xml
Priority	100
Stack Size	0x10000

RTP configuration for messageboard

Exec Path on Target	/tgtsvr/messageboard.vxe
Arguments	
Environment	LD_BIND_NOW=1 OSPL_URI=file:///tgtsvr/ospl.xml
Priority	100
Stack Size	0x10000

RTP configuration for chatter

Exec Path on Target	/tgtsvr/chatter.vxe
Arguments	1 User1
Environment	LD_BIND_NOW=1 OSPL_URI=file:///tgtsvr/ospl.xml
Priority	100
Stack Size	0x10000

When deployment is successful, the console will show output from each RTP. In particular the message board will show the messages sent by the `chatter` process. The console view can be switched to show the output for each process by clicking the **Display Selected Console** button.

9 Integrity

The `ospl_projgen` tool is in the `HDE/bin` directory of the DDS distribution. It is a convenience tool provided for the Integrity platform in order to aid in the creation of GHS Multi projects for running the DDS-supplied PingPong example, the Touchstone performance suite, and the Chatter Tutorial. If desired, these generated projects can be adapted to suit user requirements by using Multi and the `ospl_xml2int` tool, which is also described in this chapter.

9.1 The `ospl_projgen` command

The `ospl_projgen` tool has the following command line arguments:

```
ospl_projgen -h
ospl_projgen [-s <flash|ram>|-d] [-n] [-v] [-t <target>]
               [-l <c|c++>|c++onc] [-u <URI>] -b <bsp name>
               [-m <board model>] -o <directory> [-f]
```

Arguments shown between square brackets [] are optional; other arguments are mandatory. Arguments may be supplied in any order. All of the arguments are described in detail below.

9.1.1 Description of the arguments

- h** List the command line arguments and give brief reminders of their functions.
- s <flash|ram>** Use this argument if you wish to generate a project that will be statically linked with the kernel. The two options for this argument determine whether the resulting kernel image will be a flashable image or a loadable image. If both this argument and the **-d** argument are omitted the default of a statically-linked ram-loadable image will be generated.
- d** Use this argument to produce a project file that will yield a dynamic download image.
- NOTE:** Arguments **-s** and **-d** are mutually exclusive.
- n** Use this argument if you want to include the GHS network stack in your project
- v** Use this argument if you want to include filesystem support in your project.
- t <target>** Use this argument to specify which address spaces to include in your project. Use **-t list** to show a list of available targets. (Targets available initially are examples supplied with OpenSplice DDS and Integrity itself.)

- l** **<c / c++ / c++onc>** Use this argument to specify the language for your project. The default is **c++**.
- u** **<URI>** Use this argument to identify which configuration file to use. You can omit this argument if you have the environment variable `OSPL_URI` set, or use it if you want to use a different configuration file from the one referred to by `OSPL_URI`. The default is **`$OSPL_URI`**. The `xml2int` tool uses this configuration file when generating the Integrate file for your project.
- b** **<bsp name>** Use this argument to specify the BSP name of your target board. Use **-b list** to show a list of supported target boards.
- m** **<board model>** Use this argument to specify the model number for the target board. Use **-b <bsp name> -m list** to show a list of supported model numbers. (There are no separate model numbers for pcx86 boards.)
- o** **<directory>** Use this argument to specify the output directory for the project files. The name you supply here will also be used as the name for the image file that will be downloaded/flushed onto the Integrity board.
- f** Use this argument to force overwrite of the output directory.

When you run the tool, the output directory specified with the **-o** argument will be created. Go into this directory, run GHS Multi, and load the generated project.

If the output directory already exists and the **-f** argument has been omitted, `ospl_projgen` will exit without generating any code and will notify you that it has stopped.

NOTE: The `NetworkInterfaceAddress` configuration parameter is *required* for Integrity nodes which have more than one ethernet interface, as it is not possible to determine which are broadcast/multicast enabled. (See sections 3.5.2.1 and 3.9.2.1 Element *NetworkInterfaceAddress* in the *Deployment Guide*.)

9.1.2 Using `mmstat` and `shmdump` diagnostic tools on Integrity

When `mmstat` or `shmdump` targets are specified to `ospl_projgen` an address space will be added to the generated project. There will also be an appropriate `mmstat.c` or `shmdump.c` file generated into the project. In order to configure these, the command line arguments can be edited in the generated `.c` files. The `shmdump` tool can be controlled via telnet on port 2323 (by default).

9.2 PingPong Example

(Please refer to 5.6.1, *The PingPong Example*, on page 21 for a description of this example application.)

To generate a project for the C++ PingPong example, follow these steps:

Step 1: The `I_INSTALL_DIR` environment variable must be set to point to the Integrity installation directory on the host machine before running `ospl_projgen`. For example:

```
% export I_INSTALL_DIR=/usr/ghs/int509
```

Step 2: Navigate to the `examples/dcps/standalone/C++/PingPong` directory

Step 3: Run `ospl_projgen` with the following arguments:

```
% ospl_projgen -s ram -v -n -t pingpong -l c++ -b pcx86 -o projgen
```

Step 4: Go into the `projgen` directory, which contains `default.gpj` and a `src` directory. (`default.gpj` is the default Multi project that will build all the sub-projects found in the `src` directory, and the `src` directory contains all the sub-projects and generated files produced by the tool.)

Step 5: Start Multi:

```
% multi default.gpj
```

You should see a screen similar to the one in *Figure 3* below:

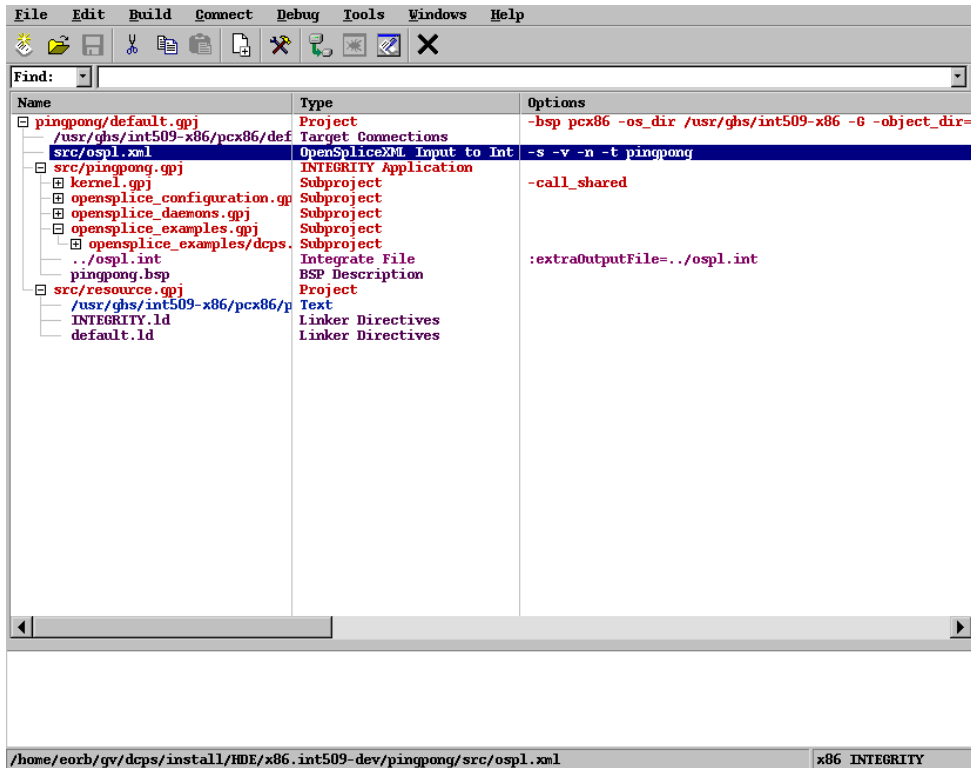


Figure 3 Integrity: project defaults

Step 6: If no changes are required to the project, right-click on `default.gpj` and then click **Build** to build the project.

Upon successful completion of the build process, an image is generated (in our case called `projgen`) in the `src` directory and you are now ready to either dynamically download the resulting image to the board or load the kernel image onto the board (depending on the arguments you have specified) and run the PingPong example.

If `osp1_projgen` is run and the project built as described above, the generated image will contain:

- GHS Integrity OS (Kernel, Networking, and Filesystem support)
- OpenSplice DDS (including `spliced` and the services described in the `osp1.xml` file)
- the PingPong example

Once the image has been downloaded to the board, the `pong` “Initial task” should be started and then the `ping` `AddressSpace` can be started in the same way, so that the example begins the data transfer. Parameters are not required to be passed to the Integrity processes because the `ospl_projgen` tool generates code with particular values that simulate the passing of parameters.

This also applies to the Chat Tutorial (see 5.6.2, *The Tutorial Example*, on page 22), if `ospl_projgen` is run with the `-t chat` argument.

9.3 Changing the `ospl_projgen` arguments

If changes are subsequently required to the arguments that were originally specified to the `ospl_projgen` tool, there are two choices:

- a) Re-run the tool and amend the arguments accordingly, *or*
- b) Make your changes through the Multi tool.

The first method guarantees that your project files will be produced correctly and build without needing manual changes to the project files. To use this method, simply follow the procedure described above but supply different arguments.

The second method is perhaps a more flexible approach, but as well as making some changes using Multi you will have to make other changes by hand in order for the project to build correctly.

The following section describes the second method.

9.3.1 Changing the generated OpenSplice DDS project using *Multi*

You can make changes to any of the settings you specified with `ospl_projgen` by following these steps:

- Step 1:** Right-click on the highlighted `ospl.xml` file (as shown above) and click **Set Options....**
- Step 2:** Select the **All Options** tab and expand the **Advanced** section.
- Step 3:** Select **Advanced OpenSplice DDS XML To Int Convertor Options**. In the right-hand pane you will see the options that you have set with the `ospl_projgen` tool with their values, similar to *Figure 4* below.

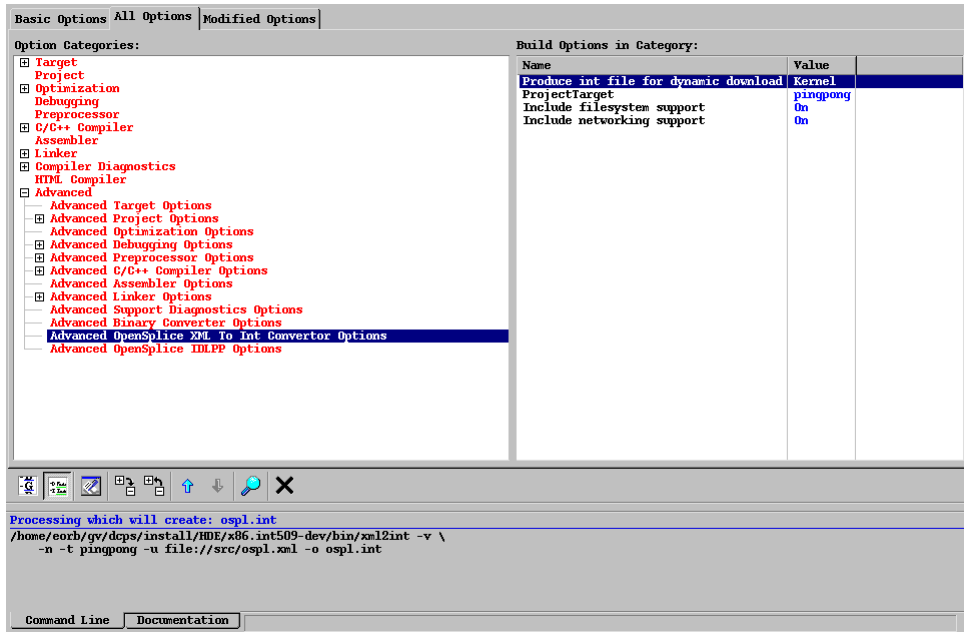


Figure 4 Integrity: changing project options in Multi

Step 4: Right-click on the parameter that you want to change. For example, if you don't need filesystem support to be included in the kernel image, right-click on **Include filesystem support** and set the option to **Off**.

The arguments for `xml2int` in the bottom pane are updated to reflect any changes that you make. If you switch off filesystem support, the `-v` argument is removed from the arguments. (The `xml2int` tool is used to generate the `ospl.int` Integrate file that will be used during the Integrate phase of the project. For more information on `xml2int`, please see section 9.4.1, *The `ospl_xml2int` command*, below.)

Note that if you do remove filesystem support from the kernel image you should also remove all references to the `ivfs` library, and make appropriate changes to the `ospl_log.c` file as well. See section 9.6, *Amending OpenSplice DDS configuration with Multi*, on page 52, for information about `ospl_log.c`.

Similarly you can change any other option and the changes are applied instantly.

Step 5: When the changes are complete, rebuild the project by right-clicking on `default.gpj` and then click **Build** to build the project.

9.4 The `ospl_xml2int` tool

The `ospl_xml2int` tool is used to inspect your OpenSplice DDS configuration file (`ospl.xml`) and generate an appropriate Integrate file (`ospl.int`). For more information on Integrate files please consult the Integrity manual.

9.4.1 The *ospl_xml2int* command

The *ospl_xml2int* tool can be run with the following command line arguments:

```
ospl_xml2int -h
ospl_xml2int [-s|-d] [-v] [-n] [-t <target>] [-u <URI>]
              [-o <file>]
```

Arguments shown between square brackets [] are optional; other arguments are mandatory. Arguments may be supplied in any order. All of the arguments are described in detail below.

9.4.2 Description of the arguments

- h** List the command line arguments and give brief reminders of their functions
- s** Generate for static linkage with kernel.
- d** Use this argument to generate an Integrate file that will yield a dynamic download image. If both this argument and the **-s** argument are omitted the default of a statically-linked image will be generated.

NOTE: arguments **-s** and **-d** are mutually exclusive.

- v** Include filesystem support.

- n** Include network support.

- t <target>** Available targets:

chat	include chat tutorial
pingpong	include PingPong example
touchstone	include Touchstone
mmstat	include mmstat
shmdump	include shmdump

Multiple **-t** arguments may be given. This enables you to use *mmstat* and/or *shmdump* (see *Using mmstat and shmdump diagnostic tools on Integrity* on page 44) in conjunction with one of the examples.

- u <URI>** Identifies the configuration file to use (default: \${OSPL_URI}).

- o <file>** Name of the generated Integrate file.

Applications linking with OpenSplice DDS must comply with the following requirements:

- The *First* and *Length* parameters must match those of *spliced* address space (these are generated from *ospl.xml*).
- The address space entry for your application in the Integrate file must include entries as shown in the example below.

Have a look at the *ospl.int* for the PingPong example if in doubt as to what the format should be. (Make sure that you have built the project first or else the file will be empty).

Example ospl.int contents

```

AddressSpace
.
.
.

Object 10
Link
Name ResourceStore
OtherObjectName ResCon
DDS_Connection
EndObject

Object 11
Link
Name ResourceStore
OtherObjectName ConnectionLockLink
DDS_ConnectionLock
EndObject

Object 12
MemoryRegion your_app_name_database
MapTo splice_database
First 0x20000000
Length 33554432
Execute true
Read true
Write true

EndObject
.
.
EndAddressSpace

```



NOTE: If you make any changes to the `ospl.int` file generated by the project and then you make any changes to the `ospl.xml` file and rebuild the project, the changes to the `ospl.int` file will be overwritten.

Make sure that you also edit the `global_table.c` and `mounttable.c` files to match your setup. These files can be found under `src/projgen/kernel.gpj/kernel_kernel.gpj` and `src/projgen.gpj/kernel.gpj/ivfs_server.gpj` as shown in *Figure 5* below:

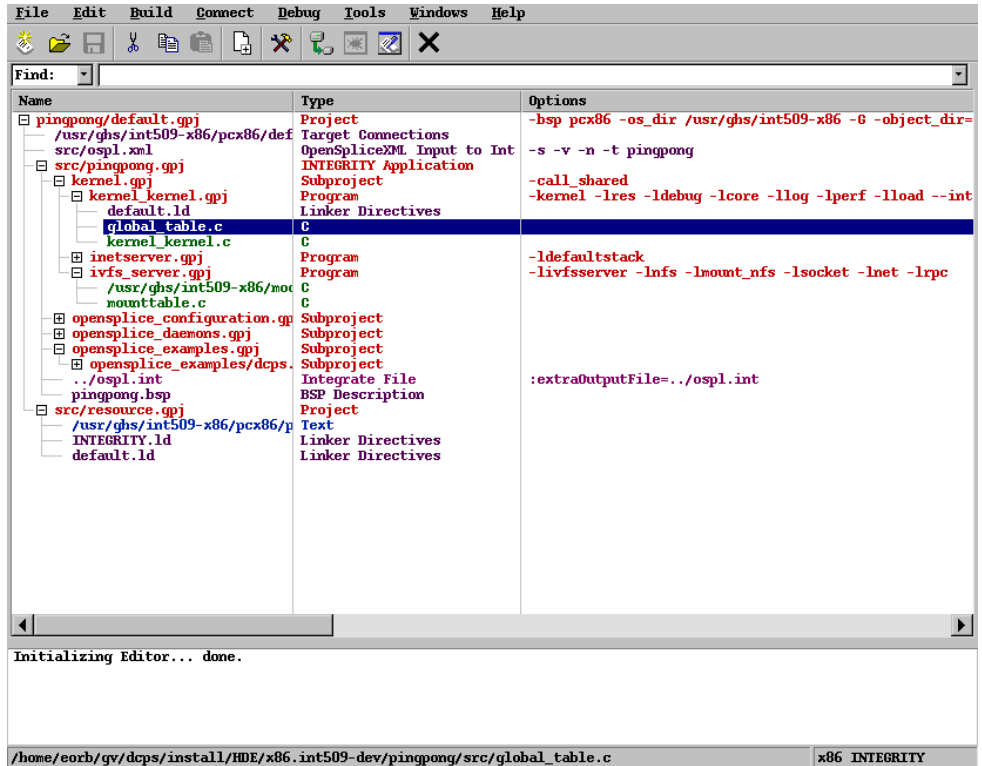


Figure 5 Integrity: changing global_table.c and mounttable.c

Once you have made all of the required changes to osp1.int, you must rebuild the whole project. Your changes will be picked up by OpenSplice DDS automatically.

9.5 Critical warning about *Object 10* and *Object 11*

We have used Object 10 and Object 11 in various address spaces to declare a semaphore and a connection object, but they may already be in use on your system.



You can change these numbers, in the osp1.int file, but if you do then you *must* change *all* of the address spaces where Object 10 and Object 11 are defined (except those for ResourceStore as noted below). The value replacing 10 must be the same for every address space, and likewise for the value replacing 11. You *must* change *all* references in order for OpenSplice DDS to work correctly.



The only exception is the ResourceStore address space. Object 10 and Object 11 are unique to the OpenSplice DDS ResourceStore and they MUST NOT be altered. If you do change them, OpenSplice DDS WILL NOT WORK!

9.6 Amending OpenSplice DDS configuration with *Multi*

You can make changes to the OpenSplice DDS configuration from Multi by editing the files under the project `src/projgen.gpj/opensplice_configuration.gpj/libospl_cfg.gpj`. See Figure 6 below:

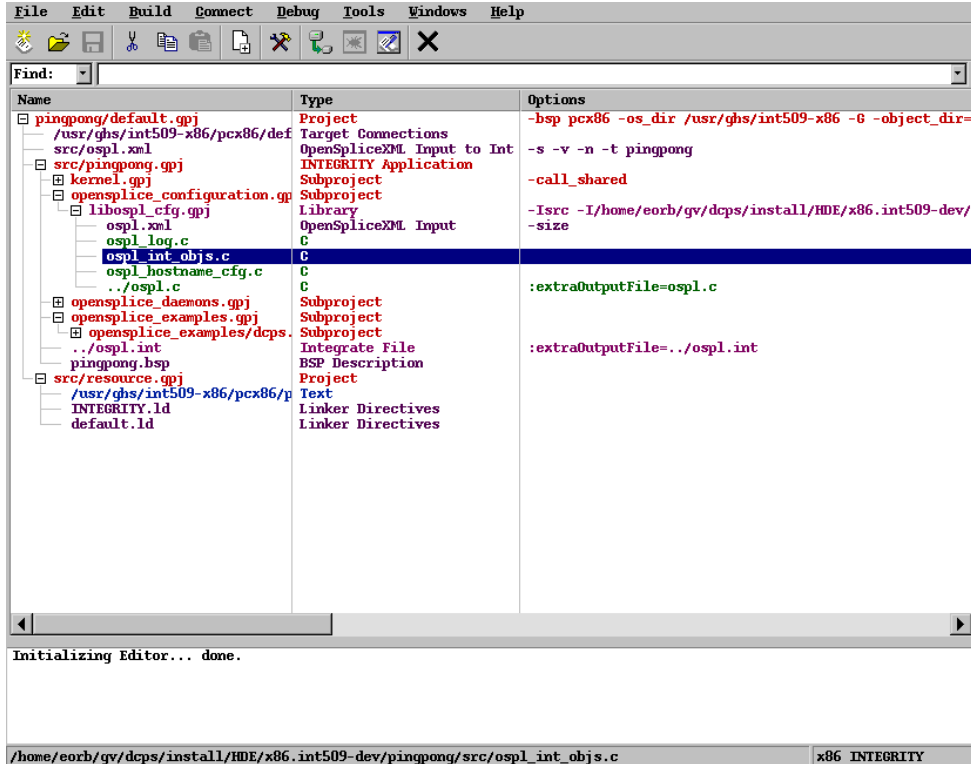


Figure 6 Integrity: changing OpenSplice DDS configuration in Multi



There are five files here but you may *only* change `ospl.xml` and `ospl_log.c`. The others must **NOT** be altered!

ospl.xml This is your OpenSplice DDS configuration file. (See Section 5.5, *Configuration*, on page 17 for more information about the options an OpenSplice DDS configuration file may have.)

ospl_log.c This file determines where the log entries (errors, warnings and informational messages) from OpenSplice DDS go. The way the default file is generated by `ospl_projgen` depends on whether you have specified filesystem support or not. (See comments within the file for more information.)