OpenSplice DDS
Version 6.x
Getting Started Guide

PRISMTECH

OpenSplice DDS

GETTING STARTED GUIDE

PRISMTECH

Copyright Notice

CONTENTS

Table of Contents

Installation and Configuration

Platform-specific Information

PRISMTECH

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.

This *Getting Started Guide* contains:

- general information about OpenSplice DDS

- a list of documents and how to use them

- initial installation and configuration information for the various platforms which OpenSplice DDS supports (additional 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.*.

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.

i Item contains helpful hint or special information.

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

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

C C language specific.

C++ C++ language specific.

C# C# language specific.

Java Java language specific.

Hypertext links are shown as *blue italic underlined*.

On-Line (PDF) versions of this document: Items shown as cross-references (*e.g. Contacts* on page x) act as hypertext links; click on the reference to go to the item.

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

Courier fonts (also *italic* and **bold**) indicate programming code and file names.

Extended code fragments are shown using Courier font in shaded boxes:

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

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

Italics and ***bold italics*** are used to indicate new terms, or emphasise an item.

Sans-serif and **bold sans-serif** indicate Graphical User Interface (GUI) and Integrated Development Environment (IDE) elements and commands; for example, 'click the Cancel button' and 'choose **File > Save**'.

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	**European Head Office**
PrismTech Corporation	PrismTech Limited
400 TradeCenter	PrismTech House
Suite 5900	5th Avenue Business Park
Woburn, MA	Gateshead
01801	NE11 0NG
USA	UK
Tel: +1 781 569 5819	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*

PRISMTECH

ABOUT OPENSPLICE DDS

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

1.3 OpenSplice DDS Summary

PrismTech's OpenSplice DDS, is a second generation, fully compliant OMG DDS implementation, offering support for all the DCPS profiles (minimum profile, ownership profile, content subscription profile and persistence profile). OpenSplice DDS was initially developed as SPLICE-DDS by Thales Naval Netherlands (TNL), one of the co-authors of the DDS specification and is the result of TNL's over 15 year experience in developing distributed information systems for naval Combat Management Systems (CMS). This field proven middleware is used as the 'information backbone' of TNL's TACTICOS CMS currently deployed in over 18 navies around the world. OpenSplice DDS is the 2nd-generation COTS evolution of this successful product and consists of several modules that cover the full OMG specification as well as provision of total lifecycle support by an integrated productivity tool suite:

- **OpenSplice DDS core modules** cover the 'Minimum' and 'Ownership' profiles that provide the basic publish-subscribe messaging functions. The minimum profile is meant to address real time messaging requirements, where performance

and small footprint are essential. The ownership profile provides basic support for replicated publishers where 'ownership' of published data is governed by 'strength' indicating the quality of published information.

- **OpenSplice DDS** *content subscription* **and** *persistence* **profiles** provide the additional information management features, key for assuring high information availability (fault-tolerant persistence of non-volatile information) as well as powerful 'content aware' features (filters and queries), thus enabling unmatched performance for the full range of small scale embedded up to large scale fault-tolerant systems.

Free evaluation licenses for OpenSplice DDS are available by e-mailing *sales@prismtech.com*. Currently supported platforms include Solaris Sparc, Linux x86, VxWorks PowerPC, x86 and VxSimulator and Windows x86, and supported languages are C, C++ (standalone or in seamless cohabitation with any ORB and related C++ compiler), Java and C#.

1.4 OpenSplice DDS Architecture

1.4.1 Overall

To ensure scalability, flexibility and extensibility, OpenSplice DDS has an internal architecture that, when selected, uses shared memory to 'interconnect' not only all applications that reside within one computing node, but also 'hosts' a configurable and extensible set of services. These services provide 'pluggable' functionality such as networking (providing QoS driven real-time networking based on multiple reliable multicast 'channels'), durability (providing fault tolerant storage for both real-time 'state' data as well as persistent 'settings'), and remote control & monitoring 'soap service' (providing remote web based access using the SOAP protocol from the OpenSplice DDS Tuner tools).

1.4.2 Scalability

OpenSplice DDS is capable of using a shared-memory architecture where data is physically present only once on any machine, and where smart administration still provides each subscriber with his own private 'view' on this data. This allows a subscriber's data cache to be perceived as an individual 'database' that can be content-filtered, queried, etc. (using the content-subscription profile as supported by OpenSplice DDS). This shared-memory architecture results in an extremely small footprint, excellent scalability and optimal performance when compared to implementations where each reader/writer are 'communication endpoints' each with its own storage (in other words, historical data both at reader and writer) and where the data itself still has to be moved, even within the same physical node.

PRISMTECH

1.4.3 **Configuration**

OpenSplice DDS is highly configurable, even allowing the architectural structure of the DDS middleware to be chosen by the user at deployment time. OpenSplice DDS can be configured to run using a **shared memory** architecture, where both the DDS related administration (including the optional pluggable services) and DDS applications interface directly with shared memory. Alternatively, OpenSplice DDS also supports a **single process** library architecture, where one or more DDS applications, together with the OpenSplice administration and services, can all be grouped into a single operating system process. Both deployment modes support a configurable and extensible set of services, providing functionality such as:

- *networking* - providing QoS-driven real-time networking based on multiple reliable multicast 'channels'

- *durability* - providing fault-tolerant storage for both real-time state data as well as persistent settings

- *remote control and monitoring SOAP service* - providing remote web-based access using the SOAP protocol from the OpenSplice Tuner tool

- *dbms service* - providing a connection between the real-time and the enterprise domain by bridging data from DDS to DBMS and *vice versa*

The OpenSplice DDS middleware can be easily configured, on the fly, using its pluggable architecture: the services that are needed can be specified together with their optimum configuration for the particular application domain, including networking parameters, and durability levels for example).

There are advantages to both the single process and shared memory deployment architectures, so the most appropriate deployment choice depends on the user's exact requirements and DDS scenario.

1.4.4 **Single Process Library Architecture**

This deployment allows the DDS applications and OpenSplice administration to be contained together within one single operating system process. This single process deployment option is most useful in environments where shared memory is unavailable or undesirable. As dynamic heap memory is utilized in the single process deployment environment, there is no need to pre-configure a shared memory segment which in some use cases is also seen as an advantage of this deployment option.

Each DDS application on a processing node is implemented as an individual, self-contained operating system process (*i.e.* all of the DDS administration and necessary services have been linked into the application process). This is known as a

single process application. Communication between multiple single process applications co-located on the same machine node is done via the (loop-back) network, since there is no memory shared between them.

An extension to the single process architecture is the option to co-locate multiple DDS applications into a single process. This can be done be creating application libraries rather than application executables that can be 'linked' into the single process in a similar way to how the DDS middleware services are linked into the single process. This is known as a **single process application cluster**. Communication between clustered applications (that together form a single process) can still benefit from using the process's heap memory, which typically is an order of magnitude faster than using a network, yet the lifecycle of these clustered applications will be tightly coupled.

The Single Process deployment is the default architecture provided within OpenSplice and allows for easy deployment with minimal configuration required for a running DDS system.

Figure 1 shows an overview of the single process architecture of OpenSplice DDS.
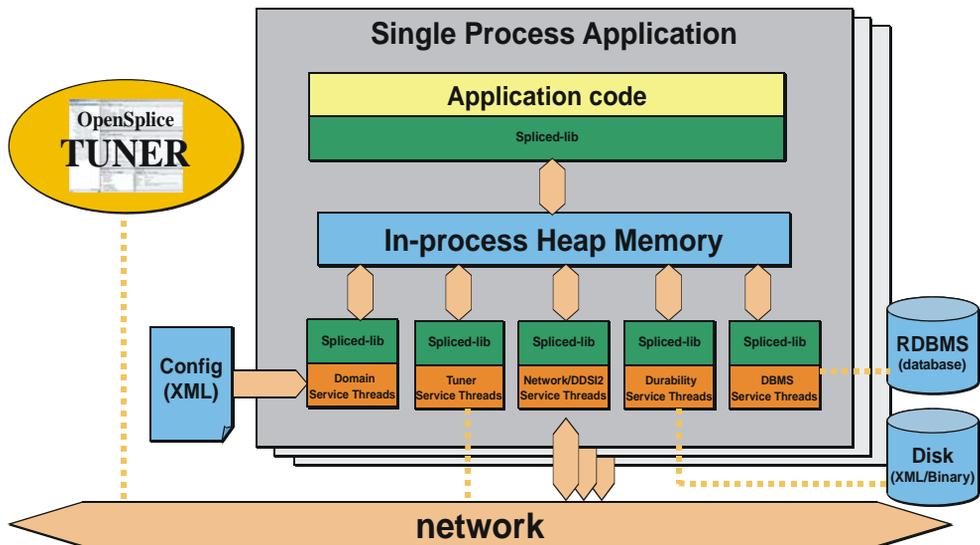
Figure 1 The OpenSplice Single Process Architecture

1.4.5 Shared Memory architecture

In the shared memory architecture data is physically present only once on any machine but smart administration still provides each subscriber with his own private view on this data. Both the DDS applications and OpenSplice administration interface directly with the shared memory which is created by the OpenSplice

daemon on start up. This architecture enables a subscriber's data cache to be seen as an individual database and the content can be filtered, queried, etc. by using the OpenSplice content subscription profile.

Typically for advanced DDS users, the shared memory architecture is a more powerful mode of operation and results in extremely low footprint, excellent scalability and optimal performance when compared to the implementation where each reader/writer are communication end points each with its own storage (*i.e.* historical data both at reader and writer) and where the data itself still has to be moved, even within the same platform.

Figure 2 shows an overview of the shared memory architecture of OpenSplice DDS on one computing node. Typically, there are many nodes within a system.
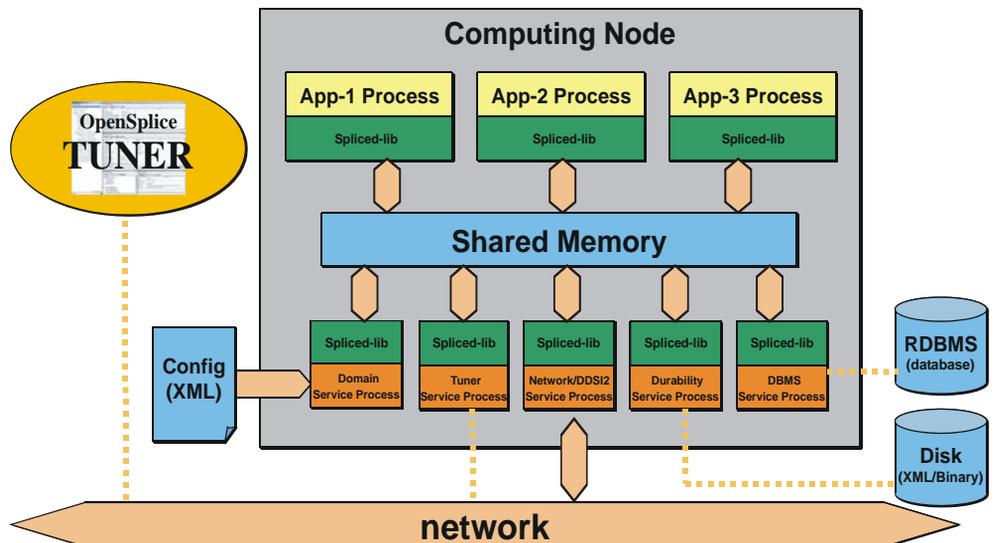
Figure 2 The OpenSplice Shared Memory Architecture

i   *Figure 2* only shows one node whereas there are typically many nodes within a system.

1.5 **OpenSplice DDS Implementation Benefits**

Table 1 below shows the following aspects of OpenSplice DDS, where:

> *Features* significant characteristics of OpenSplice
> *Advantages* shows why a feature is important
> *Benefits* describes how users of OpenSplice can exploit the advantages

Table 1 OpenSplice DDS Features and Benefits

	Features	**Advantages**	**Benefits**
General	Information-centric	Enable dynamic, loosely coupled system.	Simplified & better scalable architectures
	Open standard	'Off the shelf' solutions	Lower cost, no vendor lock in
	Built on proven technology	Intended for most the demanding situations.	Assured quality and applicability
	TNN/PT 'inheritance'	Decade long of 'DDS' experience	Proven suitability in mission critical domain
	Open Source model	Strong and large user community	Security of Supply of most widely used DDS
Functional	Real-time pub/sub	Dynamic/asynchronous data communication	Autonomous decoupled applications
	Persistence profile	Fault tolerant data persistence	Application fault tolerance and data high availability
	Content-sub. Profile	Reduced complexity & higher performance.	Easier application design & scalable systems
Performance	Shared memory	low footprint, instant data availability	Processor Scalability
	Smart networking	Efficient data transport	Network Scalability
	Extensive IDL sup.	Includes unbounded strings, sequences	Data Scalability
Usability	Multiple language	Any (mix) of C, C++, Java, C#	Supports (legacy) code, allows hybrid systems
	Multiple platforms	Any (mix) of Enterprise & RTE Oss	Intercons, enterprise and embedded systems
Interoperability	DDSI/RTPS	Interoperability between DDS vendors	Smooth integration with non-OpenSplice (legacy) DDS systems
Tooling and Ease of use	All metadata at runtime	Dynamic discovery of all 'entity info'	Guaranteed data integrity
	Powerful tooling	Support for complete system lifecycle	Enhanced productivity and System Integration
	Remote connect	Web based remote access & control	Remote diagnostics using standard protocols
Legend:	**Equal** to competition	**Better** than competition	**Far surpassing** competition

PRISMTECH

1.6 **Conclusion**

PrismTech's OpenSplice DDS product complemented by its tool support together encompass the industry's most profound expertise on the OMG's DDS standard and products.

The result is unrivalled functional DDS coverage and performance in large-scale mission-critical systems, fault tolerance in information availability, and total lifecycle support including round-trip engineering. A complete DDS solution to ensure a customer's successful adoption of this exciting new technology and to support delivery of the highest-quality applications with the shortest time to market in the demanding real-time world.

About OpenSplice DDS

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* - makes it possible for you to easily configure, tune, and inspect a deployed system
- *OpenSplice Configurator* - simplifies the process for configuring the services
- *mmstat* - helps to monitor the memory usage within OpenSplice DDS.
- *MDE PowerTools development toolsuite* - a separate installer for modelling your DDS infrastucture and generating the DDS code, increasing productivity by up to ten times.

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.

- OpenSplice DDS supports both Single Process and Shared Memory architectures, covering both ease of use and advanced optimal performance scenarios.

2.3 **Language and Compiler Bindings**

The OpenSplice DDS DCPS API is available for the following languages:

- C
- C++ (including OMG's latest ISO-C++ API)
- C#
- Java

With OpenSplice DDS, there is also the ability to use the DDS and DCPS APIs using a CORBA cohabitation mode. Cohabitation allows you to use objects in both DCPS and CORBA without copying them from one representation to the other. This means that CORBA objects can be published directly in DCPS and the other way around. There is no difference in the DDS API, only in the generated code produced by the idlpp tool in the development process. OpenSplice DDS has CORBA cohabitation for C++ and Java, using (by default) OpenFusion TAO and OpenFusion JacORB. Other variations are available, please check the *Release Notes* for full platform and language ORB coverage.

The full range of language bindings available is:

- C (Standalone only) - sac
- C++ (Standalone and CORBA Cohabitation) - isocpp / sacpp / ccpp
- C# (Standalone only) - sacs
- Java (Standalone and CORBA Cohabitation) - saj / cj

OpenSplice DDS is delivered with a preset compiler for C++. Details of this can be found in the *Release Notes*.

These compilers are the officially-supported set, but we have experience of customers who will use the delivered libraries with slight variants of the compiler. In most cases this works, but PrismTech has provided the source code so that customers can rebuild the C++ APIs for their compiler of choice.

NOTE: PrismTech provides support on the officially-supported platforms due to difficult-to-fix issues with compiler-generated code, but some customers will fund us to qualify OpenSplice on their platform. If you wish to use a variant of an official platform, then as long as the issue can be recreated on the official platform it will be covered under an OpenSplice DDS support contract. If you wish to request support on a specific platform then please contact PrismTech (*http://www.prismtech.com/contact-us*)

PRISMTECH

2.3.1 Building your own C++ and CORBA APIs

2.3.1.1 Building your own ISO C++ API

The OpenSplice DDS DCPS API for the ISO C++ language binding without CORBA cohabitation is delivered using a specific compiler.

To be able to use a different compiler with the OpenSplice DDS ISO C++ API, we deliver the source code for this language with the OpenSplice DDS distribution. This is contained in a directory `<OpenSplice DDS Installation directory>/custom_lib/isocpp` along with a `README` file describing how to generate the custom library.

2.3.1.2 Building your own Standalone C++ API

The OpenSplice DDS DCPS API for the C++ language binding without CORBA cohabitation is delivered using a specific compiler.

To be able to use a different compiler with the OpenSplice DDS Standalone C++ API, we deliver the source code for this language with the OpenSplice DDS distribution. This is contained in a directory `<OpenSplice DDS Installation directory>/custom_lib/sacpp` along with a `README` file describing how to generate the custom library.

2.3.1.3 Building your own CORBA C++ API

The OpenSplice DDS DCPS API for the C++ language binding with CORBA cohabitation is delivered using the OpenFusion TAO ORB and a specific compiler. The ORB 'OpenFusion TAO' can be obtained from PrismTech (*http://www.prismtech.com*).

The directory `<OpenSplice DDS Installation directory>/custom_lib/ccpp` contains the source code for building your own CORBA C++ API library for an ORB and a compiler of your choice. This directory also contains a `README` file, which describes how to generate this custom library.

2.3.1.4 Building your own CORBA Java API

The OpenSplice DDS DCPS API for the Java language binding with CORBA cohabitation is built with OpenFusion JacORB. The ORB 'OpenFusion JacORB' can be obtained from PrismTech (*http://www.prismtech.com*).

The OpenSplice DDS CORBA Java API can also be rebuilt against another CORBA-compliant Java ORB. The directory `<OpenSplice DDS Installation directory>/custom_lib/cj` contains the necessary source code, along with a `README` file which describes how to generate the custom library.

2.4 **Platforms**

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

Please refer to *Platform-specific Information* (page 33 onwards) for information about using OpenSplice DDS on specific platforms.

PRISMTECH

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

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.
RMI over DDS Getting Started Guide	Explains how to take advantage of the client/server interaction paradigm provided by OpenSplice RMI layered over the publish/subscribe paradigm of OpenSplice DDS.
OpenSplice Automated Testing and Debugging Tool User Guide	Provides a complete reference on how to configure the tool and use it to test waveforms generated with the OpenSplice DDS.
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.
C# Reference Guide	Describes the OpenSplice DDS API for C#. Supplied as HTML rather than PDF and found in the product *Release Notes*.
Examples	Examples, complete with source code, demonstrating how applications using OpenSplice can be written and used. Documentation for the examples can be found in the OpenSplice *Release Notes*.
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*

PRISMTECH

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://www.prismtech.com/knowledge-base

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*.

PrismTech

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 5.5, *Examples.*

5.1 OpenSplice DDS Development and Run-Time

OpenSplice DDS is provided in two installers. The ***HDE*** (Host Development Environment) is the standard and it requires approximately 60 Mb of disk space after installation; the ***RTS*** (Run Time System) requires approximately 35 Mb of disk space.

The HDE contains all of the services, libraries, header files and tools needed to develop applications using OpenSplice, and the RTS is a subset of the HDE which contains all of the services, libraries and tools needed to deploy applications using OpenSplice.

5.2 Installation for UNIX and Windows Platforms

Step 1: 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 (this is only necessary on UNIX, as the Windows environment is configured by the OpenSplice installer)

UNIX 1. Go to the `<install_dir>/<E>/<platform>` directory, where `<E>` is HDE or RTS and `<platform>` is, for example, `x86.linux2.6`.

UNIX 2. Source the `release.com` file from the shell command line.

This step performs all the required environment configuration.

Step 3: Install your desired ORB when the C++ language mapping is used with CORBA cohabitation. 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 on Other Platforms**

Please refer to *Platform-specific Information* (page 33 onwards) for information about using OpenSplice DDS on specific platforms.

5.4 **Configuration**

OpenSplice DDS is configured using an XML configuration file, as shown under *XML Configuration Settings* on page 26. It is advisable to use the `osplconf` tool (UNIX) or OpenSplice DDS Configurator Tool (Windows Start Menu) to edit your xml files. The configurator tool provides explanations of each attribute and also validates the input.

The default configuration file is `ospl.xml` located in `$OSPL_HOME/etc/config` (alternative configuration files may also be available in this directory, to assist in other scenarios). 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 deployment specified by the XML configuration file is for a Single Process deployment. This means that the OpenSplice Domain Service, database administration and associated services are all started within the DDS application process. This is implicitly done when the user's application invokes the DDS `create_participant` operation.

The deployment mode and other configurable properties can be changed by using a different `OSPL_URI` file. Several sample configuration files are provided at the same location.

PRISMTECH

If using a shared memory configuration, a `<Database>` attribute is specified in the XML configuration. The default Database Size that is mapped on a shared memory segment is 10 Megabytes

NOTE: 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 parts shown in **bold** 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 its 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="0.05">60.0</ExpiryTime>
        </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>
```

PRISMTECH

```
                <NameSpace durabilityKind="Durable"
                    alignmentKind="Initial_and_Aligner">
                    <Partition>*</Partition>
                </NameSpace>
            </NameSpaces>
            <Persistent>
                <StoreDirectory>/tmp/Pdata</StoreDirectory>
            </Persistent>
        </DurabilityService>
</OpenSplice>
```

5.5 Examples

A great way to get started with OpenSplice DDS is to try running the examples provided with the product. There are many examples in different languages and some with the CORBA cohabitation, showing different aspects of the DCPS API. To give you a feel for how powerful DDS is then we recommend trying `PingPong` and the `Tutorial`.

The way to build and run the examples is dependent on the Platform you are using. Each example has HTML documentation explaining how to build and run it on Unix/Linux and Windows systems. This can be found in the OpenSplice DDS documentation.

For VxWorks and Integrity, please refer to Chapter 7, *VxWorks 5.5.1*, on page 35, Chapter 8, *VxWorks 6.x RTP*, on page 43, and Chapter 10, *Integrity*, on page 59 in this Guide.

5.5.1 Using the OpenSplice Tools

NOTE: The following instructions apply only to the *shared memory* deployment of OpenSplice DDS. When deploying in single process configuration, there is no need to manually start the OpenSplice infrastructure prior to running a DDS application process, as the administration will be created within the application process. Please refer to the OpenSplice *Deployment Guide* for a discussion of these deployment architectures.

The OpenSplice infrastructure can be stopped and started from the Windows Start Menu, as well as the Tuner and Configurator.

Step 1: Manually start the OpenSplice infrastructure

1. Enter `ospl start` on the command line.[1] This starts the OpenSplice services.

2. These log files may be created in the current directory when OpenSplice is started:

   a) `ospl-info.log` - contains information and warning reports

---

1. *ospl* is the command executable for OpenSplice DDS.

b) `ospl-error.log` - contains error reports

i   If OpenSplice DDS is used as a Windows Service then the log files are re-directed to the path specified by OSPL_LOGPATH. (Use the `set` command in the OpenSplice DDS command prompt to see the OSPL_LOGPATH value.)

Step 2: 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 3: Experiment with the OpenSplice tools and applications

1. Use the OpenSplice Tuner to monitor all DDS entities and their (dynamic) relationships

Step 4: 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.

6 *Licensing OpenSplice*

6.1 General

OpenSplice DDS uses Reprise License Manager (RLM) 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.

i *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.

⚠ Some OpenSplice components are licensed individually and you will need the correct feature to be unlocked for you to use them.

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 `rlm`.

If another location is used or the environment has not been setup, then an environment variable, either *RLM_LICENSE* or *prismtech_LICENSE*, 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:

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

If licenses are distributed between multiple license files, the *RLM_LICENSE* or *prismtech_LICENSE* 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 *RLM_LICENSE* or *prismtech_LICENSE* variable must be set to the correct license file location.

To run the license manager, use the following command:

```
%   rlm -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 rlm command will start the PrismTech vendor daemon prismtech, 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 *RLM_LICENSE* or *prismtech_LICENSE* environment variable to point to the License Manager Daemon, using the following syntax:

```
%   RLM_LICENSE=<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:

```
07/05 12:05 (rlm) License server started on rhel4e
```

PRISMTECH

```
07/05 12:05 (rlm) Server architecture: x86_l2
07/05 12:05 (rlm) License files:
07/05 12:05 (rlm)     license.lic
07/05 12:05 (rlm)
07/05 12:05 (rlm) Web server starting on port 5054
07/05 12:05 (rlm) Using TCP/IP port 5053
07/05 12:05 (rlm) Starting ISV servers:
07/05 12:05 (rlm)     ... prismtech on port 35562
07/05 12:05 (prismtech) RLM License Server Version 9.1BL3 for ISV
"prismtech"
07/05 12:05 (prismtech) Server architecture: x86_l2

    Copyright (C) 2006-2011, Reprise Software, Inc. All rights reserved.

    RLM contains software developed by the OpenSSL Project
    for use in the OpenSSL Toolkit (http//www.openssl.org)
    Copyright (c) 1998-2008 The OpenSSL Project. All rights reserved.
    Copyright (c) 1995-1998 Eric Young (eay@cryptsoft.com) All rights
reserved.

07/05 12:05 (prismtech)
07/05 12:05 (prismtech) Server started on rhel4e (hostid: 0025643ad2a7)
for:
07/05 12:05 (prismtech)     opensplice_product1 opensplice_product2
07/05 12:05 (prismtech)
07/05 12:05 (prismtech) License files:
07/05 12:05 (prismtech)     license.lic
07/05 12:05 (prismtech)
```

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 *RLM_LICENSE* or *prismtech_LICENSE* would be:

```
35562@rhel4e
```

6.3.1 **Utilities**

A utility program, `rlmutil`, 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:

```
%   rlmutil rlmdown -vendor prismtech
```

```
%   rlmutil rlmdown -c <location>
```

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

The `rlmutil` 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:

```
%   rlmutil rlmhostid
```

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

```
Hostid of this machine: 0025643ad2a7
```

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

PLATFORM-SPECIFIC INFORMATION

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 Solaris-hosted system is being used, and that OpenSplice DDS is installed in `/usr/local/vxworks5.5.1`.

7.1 Building a VxWorks Kernel

Required modules

The following modules are the core system components needed to build the OpenSplice DDS runtime. Please refer to WindRiver's documentation for additional information describing how VxWorks kernels can be built.

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 optional but are useful for HDE (Host Development Environment) development. These modules are required if deploying from the Tornado front end:

Development tool components

- WDB agent components
  - WDB agent services
- WDB target server file system
  - symbol table components

Platform-specific Information

- synchronize host and target symbol tabels
- target shell components
    - target shell

7.2 Scenarios for Building the OpenSplice Examples

There are two scenarios included for building and deploying the OpenSplice examples.

- You can build one DKM containing the example, OpenSplice and all of its required services and support libraries, as well as a default configuration file. (This is the recommended approach).

- Alternatively, separate DKMs are supplied for each of the OpenSplice libraries and services, and the examples can be built as a DKM containing only the examples (we refer to this as the '*AppOnly*' style).

7.2.1 The OpenSplice Examples (All linked in one complete DKM – *recommended*)

7.2.1.1 To build the standalone C PingPong example

At the prompt, `cd` to `examples/dcps/PingPong/c/standalone/` and run `make`.

7.2.1.2 Note about the example projects

The example builds by linking the object produced by compiling the output of `osplconf2c` along with the example application, the `splice` deamon, and services enabled in the configuration XML, into one single downloadable kernel module.

Users producing their own application could of course decide to link the object and library files into a monolithic kernel image instead.

7.2.1.3 The osplconf2c tool

`osplconf2c` is required for example and user applications.

`osplconf2c` is a tool which processes the OpenSplice configuration XML, and produces a source file to be compiled and linked into the final image. It contains the data from the XML file, as well as any environment variables that you require to configure OpenSplice and references to the symbols for the entry points of the OpenSplice services.

Environment variables can be added using the `-e` option. For example, you would use the option `-e "OSPL_LOGPATH=/xxx/yyy"` if you wanted the logs to be placed in `/xxx/yyy`.

The example `makefiles` runs `osplconf2c` automatically.

7.2.1.4 Overriding OpenSplice configuration at runtime

You can override the OpenSplice configuration XML provided to `osplconf2c` at runtime by specifying the URI of a file when starting `ospl_spliced` on the target. For example:

```
ospl_spliced "file:///tgtsvr/ospl.xml"
```

It should be noted, however, that the `osplconf2c` will have generated references to the symbols for the services which are specified in the xml file when it started, and only those services may be used in the new configuration, as other services will not be included in the image. As an exception to this, if the `-d` option is specified then dynamic loading is supported, and DKMs for additional services will be automatically loaded; DKMs for any required 'libraries' must be pre-loaded by the user.

⚠ **NOTE**: Symbol table support will be required in the kernel if the `-d` option is used. Without the `-d` option it should still be possible to statically link OpenSplice with a kernel even if symbol table support is not included, for example for final deployment.

7.2.1.5 Running the Examples

If you included the additional modules listed above (see Section 7.1, *Building a VxWorks Kernel*, on page 35) in the kernel, deployment is done *via* the target server setup from the Tornado shell connection.

7.2.1.6 Background

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 Unix system, the command would be:

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

and on VxWorks:

```
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 *)
cmsoap -> ospl_cmsoap(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)
cmsoap -> ospl_cmsoap_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)
```

7.2.1.7 How to start spliced and related services

For the example below the target server filesystem must be mounted as `/tgtsvr` on the target.

To start the `spliced` service and other additional OpenSplice services open a `windsh` and enter the following commands.

```
cd "$OSPL_HOME/examples/dcps/PingPong/c/standalone"
ld 1,0,"sac_pingpong_kernel.out"
ospl_spliced
```

Note that `spliced` will block when invoked by `ospl_spliced` so open a new `windsh` to run the following Pong command:

```
pong ( "PongRead PongWrite" )
```

After the Pong application has started you can open another `windsh` and start Ping. However, if you are running the Ping application on another target board you must load and start `spliced` on that target also, as described above.

```
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")
```

The `ospl-info.log` file can be inspected to check the deployment has been successful. By default, this is written to the `/tgtsvr` directory.

The *moduleShow* command can be used within the VxWorks shell to see that the service modules have loaded, and the the 'i' command should show that tasks have started for these services.

7.2.1.8 The osplconf2c command

Usage

```
osplconf2c -h
osplconf2c [-d [-x]] [-u <URI>] [-e <env=var> ]... [-o <file>]
```

Description of options

-h, -?   List available command line arguments and give brief reminders of their functions.

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

-o <file>   Name of the generated file.

-e <env=var>   Environment setting for configuration of OpenSplice;
e.g. `-e "OSPL_LOGPATH=/xxx/yyy"`.

-d   Enable dynamic loading.

-x   Exclude xml.

7.3 The OpenSplice Examples (Alternative scenario, with multiple DKMs)

⚠ Loading separate DKMs is not recommended by PrismTech.

Note about the example projects

Please ensure that any services called by a configuration XML contain an explicit path reference within the command tag; for example:

```
<Command>/tgtsvr/networking</Command>
```

7.3.1 To build the standalone C pingpong example

At the prompt, `cd` to `examples/dcps/PingPong/c/standalone/` and run

```
make -f Makefile_AppOnly
```

7.3.2 How to start spliced and related services

To start the `spliced` service and other additional OpenSplice services, load the core OpenSplice shared library that is needed by all OpenSplice DDS applications, and then the `ospl` utility symbols. This can be done using a VxWorks shell on as many boards as needed. The `ospl` entry point can then be invoked to start OpenSplice.

```
cd "$OSPL_HOME"
ld 1,0,"lib/libddscore.so"
ld 1,0,"bin/ospl"
os_putenv("OSPL_URI=file:///tgtsvr/etc/config/ospl.xml")
ospl("start")
```

⚠ Please note that in order to deploy the cmsoap service for use with the OpenSplice DDS Tuner, it must be configured in `ospl.xml` and the libraries named `libcmxml.so` and `libddsrrstorage.so` must be pre-loaded:

```
ld 1,0,"lib/libddscore.so"
ld 1,0,"lib/libddsrrstorage.so"
ld 1,0,"lib/libcmxml.so"
ld 1,0,"bin/ospl"
os_putenv("OSPL_URI=file:///tgtsvr/etc/config/ospl.xml")
os_putenv("PATH=/tgtsvr/bin")
ospl("start")
```

7.3.3 To run the C PingPong example from winsh

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

```
cd "$OSPL_HOME"
ld 1,0,"lib/libdcpsgapi.so"
ld 1,0,"lib/libdcpssac.so"
cd "examples/dcps/PingPong/c/standalone"
ld 1,0,"sac_pingpong_kernel_app_only.out"
pong("PongRead PongWrite")
```

After the Pong application has started you can open another `windsh` and start Ping. However, if you are running the Ping application on another target board you must load and start `spliced` on that target also, as described above.

```
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")
```

The `ospl-info.log` file can be inspected to check the deployment has been successful. By default, this is written to the `/tgtsvr` directory.

PRISMTECH

The `moduleShow` command can be used within the VxWorks shell to see that the service modules have loaded, and the the 'i' command should show that tasks have started for these services.

7.3.4 Load-time Optimisation: pre-loading OpenSplice Service Symbols

Loading `spliced` and its services may take some time if done exactly as described above. This is because the service Downloadable Kernel Modules (DKM) and entry points are dynamically loaded as required by OpenSplice.

It has been noted that the deployment may be slower when the symbols are dynamically loaded from the Target Server File System. However, it is possible to improve deployment times by optionally pre-loading service symbols that are known to be deployed by OpenSplice.

In this case OpenSplice will attempt to locate the entry point symbols for the services and invoke those that are already available. This removes the need for the dynamic loading of such symbols and can equate to a quicker deployment. When the entry point symbols are not yet available (*i.e.* services have not been pre-loaded), OpenSplice will dynamically load the services as usual.

For example, for an OpenSplice system that will deploy `spliced` with the networking and durability services, the following commands could be used:

```
cd "$OSPL_HOME"
ld 1,0,"lib/libddscore.so"
ld 1,0,"bin/ospl"
ld 1,0,"bin/spliced"
ld 1,0,"bin/networking"
ld 1,0,"bin/durability"
os_putenv("OSPL_URI=file:///tgtsvr/etc/config/ospl.xml")
os_putenv("PATH=/tgtsvr/bin")
ospl("start")
```

The `ospl-info.log` file describes whether entry point symbols are resolved having been pre-loaded, or the usual dynamic symbol loading is required.

7.3.5 Notes

In this scenario `osplcon2c` has been used with the `-x` and `-d` options to create an empty configuraion which allows dynamic loading, and the resulting object has been included in the provided `libddsos.so`.

If desired the end user could create a new `libddsos.so` based on `libddsos.a` and a generated file from `osplconf2c` without the `-x` option, in order to statically link some services but also allow dynamic loading of others if the built-in xml is later overridden with a file URI. (See Section 7.2.1.4, *Overriding OpenSplice configuration at runtime*, on page 37.)

CHAPTER

8 *VxWorks 6.x RTP*

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

WIN 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 5.4, *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:

PRISMTECH

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 3* shows this configuration.
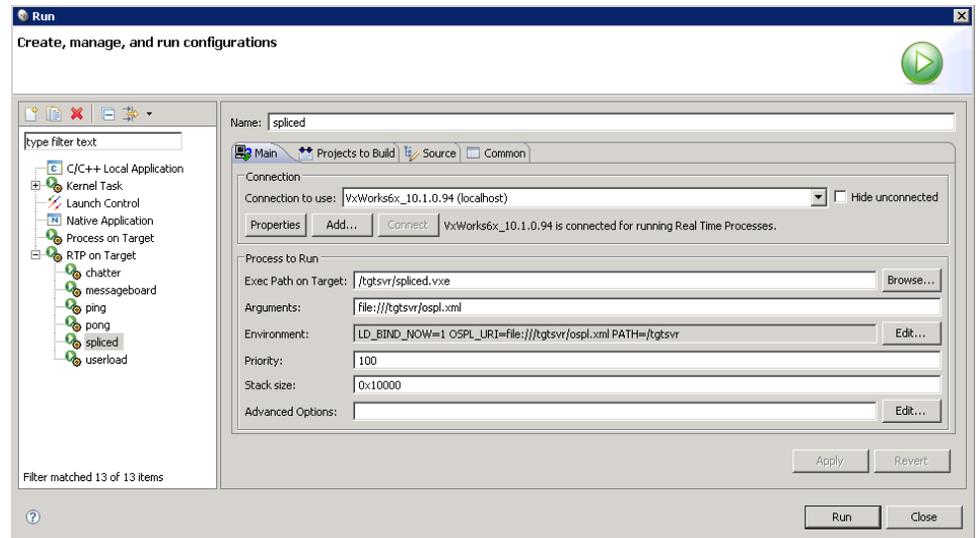
Figure 3 Workbench showing spliced deployment configuration

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 4*. (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.
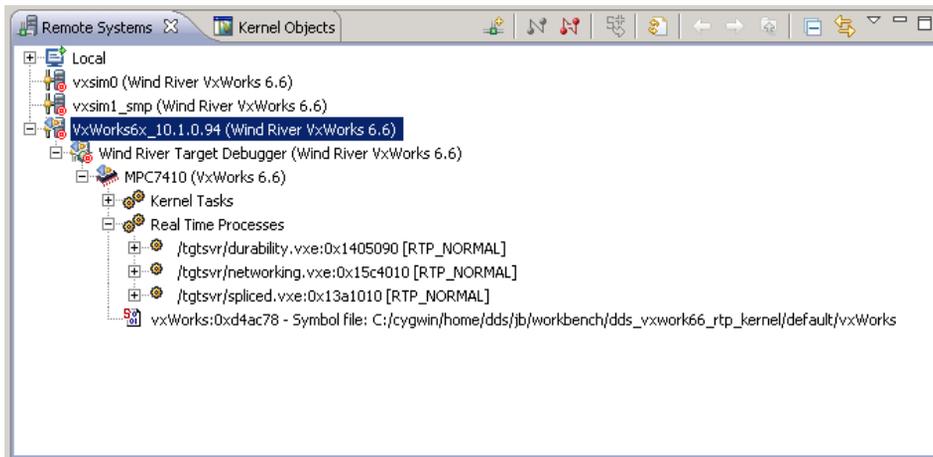
Figure 4 Workbench showing deployed OpenSplice RTPs

8.4 **OpenSplice Examples**

PrismTech provides a number of examples both for C and C++ that are described in Section 5.5, *Examples*, on page 27. 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 that described above.

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 un-checked.

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**.

PRISMTECH

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.

PRISMTECH

9

VxWorks 6.x Kernel Mode

This chapter provides a brief description of how to build the kernel and the supplied examples, and how to run those examples, using VxWorks 6.x kernel and the Workbench 'front end'. For more information about VxWorks 6.x, please refer to WindRiver's documentation.

9.1 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 choosing **File > New > VxWorks Image Project**.

9.2 Deploying OpenSplice DDS

Type a name for the project then select the appropriate Board Support Package and Tool Chain (for example `pcPentium4` and `gnu`). Leave all of the kernel options to be used blank except for the **SMP** option, which must match the OpenSplice build you are working with (*i.e.* it must be checked *only* for SMP builds of OpenSplice), and on the **Configuration Profile** dialog choose **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`)
- built-in symbol table (`INCLUDE_STANDALONE_SYM_TBL`)
- synchronize host and target symbol tables
- target shell components
  - target shell

To successfully complete the C++ examples you will also require

- **C++ components > standard library** (`FOLDER_CPLUS_STDLIB`)

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

9.2.1 Special notes for this platform

If any kernel tasks which will call onto OpenSplice API's are to be created before `ospl_spliced` is started then the user must ensure that the function `os_procInstallHook` (which takes no parameters) is called before they are started. There only needs to be one call to `os_procInstallHook`; however, mutiple calls are harmless.

9.3 OpenSplice Examples

PrismTech provides the `pingpong` example both for C and C++ that are described in Section 5.5, *Examples*, on page 27. 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 that described above.

Each project contains a `README` file briefly explaining the example and the parameters required to run it.

9.3.1 Importing Example Projects into Workbench

The example projects can be imported into Workbench by choosing **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 un-checked.

9.3.2 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**.

9.4 Running the Examples (All linked in one complete DKM - *recommended*)

Scenarios for building the OpenSplice examples

There are two included scenarios for build and deployment of the OpenSplice examples.

You can build one DKM (Downloadable Kernel Module) containing the example, OpenSplice, and all of its required services and support libraries, as well as a default configuration file. (This is the recommended approach.)

Alternatively, seperate DKMs are supplied for each of the OpenSplice libraries and services, and each example can be built as a separate DKM (containing only the example), which we refer to as '*AppOnly*' style.

9.4.1 Running the examples on two targets

9.4.1.1 The C pingpong example

Right-click on `wb_sac_pingpong_kernel` and then choose **Rebuild Build Project**.

Next configure the targets to use the target server filesystem, mapped as on the target as `/tgtsvr`.

Copy the `wb_sac_pingpong_kernel/PENTIUM4gnu/sac_pingpong_kernel/Debug/sac_pingpong_kernel.out` built above to the target server for each board as `sac_pingpong_kernel.out`.

Open a target shell connection to each board and in the C mode shell run:

```
ld 1,0,"/tgtsvr/sac_pingpong_kernel.out"
ospl_spliced
```

Open another target shell connection to one board and run:

```
pong "PongRead PongWrite"
```

Open another target shell on the other board and run:

```
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"
```

9.4.1.2 The C++ pingpong example

Right-click on `wb_sacpp_pingpong_kernel` and then choose **Rebuild Build Project**.

Next configure the targets to use the target server filesystem, mapped as on the target as `/tgtsvr`.

Copy the `wb_sacpp_pingpong_kernel/PENTIUM4gnu/sacpp_pingpong_kernel/Debug/sac_pingpong_kernel.out` built above to the target server for each board as `sacpp_pingpong_kernel.out`.

Open a target shell connection to each board and in the C mode shell run:

```
ld 1,0,"/tgtsvr/sacpp_pingpong_kernel.out"
ospl_spliced
```

Open another target shell connection to one board and run:

```
pong "PongRead PongWrite"
```

Open another target shell on the other board and run:

```
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"
```

9.4.2 Running the examples on one target

9.4.2.1 The C pingpong example

Right-click on `wb_sac_pingpong_kernel` and then choose **Rebuild Build Project**.

Next configure the targets to use the target server filesystem, mapped as on the target as `/tgtsvr`.

Copy the `wb_sac_pingpong_kernel/PENTIUM4gnu/sac_pingpong_kernel/Debug/sac_pingpong_kernel.out` built above to the target server as `sac_pingpong_kernel`.out.

Open a target shell connection and in the C mode shell run:

```
ld 1,0,"/tgtsvr/sac_pingpong_kernel.out"
ospl_spliced
```

Open another target shell connection and run:

```
pong "PongRead PongWrite"
```

Open another target shell and run:

```
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"
```

9.4.2.2 The C++ pingpong example

Right-click on `wb_sacpp_pingpong_kernel` and then choose **Rebuild Build Project**.

Next configure the targets to use the target server filesystem, mapped as on the target as `/tgtsvr`.

Copy the `wb_sacpp_pingpong_kernel/PENTIUM4gnu/sacpp_pingpong_kernel/Debug/sacpp_pingpong_kernel.out` built above to the target server as `sacpp_pingpong_kernel`.out.

Open a target shell connection and in the C mode shell run:

```
ld 1,0,"/tgtsvr/sacpp_pingpong_kernel.out"
ospl_spliced
```

Open another target shell connection and run:

```
pong "PongRead PongWrite"
```

Open another target shell and run:

```
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"
```

9.4.3 Using a different path

⚠ If you want or need to use a path other than `/tgtsvr` (*e.g.* if you are using a different filesystem) then you need to change the path set by the `-e` options of `osplconf2c` in the `.wrmakefile`.

i You can also set other environment variables with additional `-e` options.

9.4.4 Note about the example projects

The example builds by linking the object produced by compling the output of `osplconf2c` along with the example application, the splice deamon, and services enabled in the configuration XML, into one single downloadable kernel module. Users producing their own application could of course decide to link the object and library files into a monolithic kernel image instead.

9.4.5 Running the Examples (Alternative scenario, with multiple DKMs – *'AppOnly'* style)

⚠ Loading separate DKMs is not recommended by PrismTech.

C++ **NOTE:** There are no C++ examples provided for the *AppOnly* style and there is no `libdcpssacpp.out` DKM because VxWorks only supports C++ modules that are self-contained. However, it should still be possible to link your C++ application with the `libdcpssacpp.a`, and then load the complete DKM after the other OpenSplice DKMs.

9.4.5.1 **The C pingpong example**

Step 1: Right-click on `wb_sac_pingpong_kernel_app_only` for the C example or `wb_sacpp_pingpong_kernel_app_only` for C++, then choose **Rebuild Project**.

Step 2: Next configure the targets to use the target server filesystem, mapped on the target as `/tgtsvr` (use different host directories for each target).

Step 3: Copy the `ospl.xml` file from the distribution to the target server directories, and adjust for your desired configuration.

Step 4: Copy all the services from the `bin` directory in the distribution to the target server directories (for example, `spliced.out`, `networking.out`, *etc.*).

To run the examples on two targets, start the OpenSplice daemons on each target.

Step 5: Open a 'Host Shell' (`windsh`) connection to each board, and in the C mode shell enter:

```
cd "<path to opensplice distribution>"
ld 1,0,"lib/libddscore.out"
ld 1,0,"bin/ospl.out"
os_putenv("OSPL_URI=file:///tgtsvr/ospl.xml")
os_putenv("OSPL_LOGPATH=/tgtsvr")
os_putenv("PATH=/tgtsvr/")
ospl("start")
```

⚠️ Please note that in order to deploy the cmsoap service for use with the OpenSplice DDS Tuner, it must be configured in `ospl.xml` and the libraries named `libcmxml.out` and `libddsrrstorage.out` must be pre-loaded:

```
cd "<path to opensplice distribution>"
ld 1,0,"lib/libddscore.out"
ld 1,0,"lib/libddsrrstorage.out"
ld 1,0,"lib/libcmxml.out"
ld 1,0,"bin/ospl.out"
os_putenv("OSPL_URI=file:///tgtsvr/ospl.xml")
os_putenv("OSPL_LOGPATH=/tgtsvr")
os_putenv("PATH=/tgtsvr/")
ospl("start")
```

Step 6: To load and run the examples:

C *For the C example:*

```
ld 1,0,"lib/libdcpsgapi.out"
ld 1,0,"lib/libdcpssac.out"
cd "examples/dcps/PingPong/c/standalone"
ld 1,0,"sac_pingpong_kernel_app_only.out"
```

Step 7: Open a new 'Host Shell' connection to one board and run:

```
pong "PongRead PongWrite"
```

Step 8: Open another new 'Host Shell' on the other board and run:

```
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"
```

9.4.6 Running the examples on one target

Proceed as described in the section above, but make all `windsh` connections to *one* board, and only load and run `ospl` once.

9.4.6.1 Load-time Optimisation: pre-loading OpenSplice Service Symbols

Loading `spliced` and its services may take some time if done exactly as described above. This is because the service DKMs (Downloadable Kernel Modules) and entry points are dynamically loaded as required by OpenSplice.

It has been noted that the deployment may be slower when the symbols are dynamically loaded from the Target Server File System. However, it is possible to improve deployment times by pre-loading the symbols for the services that are required by OpenSplice.

On startup, OpenSplice will attempt to locate the entry point symbols for the services and invoke them. This removes the need for the dynamic loading of the DKMs providing the symbols, and can equate to a quicker deployment. Otherwise, OpenSplice will dynamically load the service DKMs.

For example, for an OpenSplice system that will deploy spliced with the networking and durability services, the following commands could be used:

```
cd "<path to opensplice distribution>"
ld 1,0,"lib/libddscore.out"
ld 1,0,"bin/ospl.out"
ld 1,0,"bin/spliced.out"
ld 1,0,"bin/networking.out"
ld 1,0,"bin/durability.out"
os_putenv("OSPL_URI=file:///tgtsvr/ospl.xml")
os_putenv("PATH=/tgtsvr/bin")
os_putenv("OSPL_LOGPATH=/tgtsvr")
ospl("start")
```

The `ospl-info.log` file records whether entry point symbols were pre-loaded, or a DKM has been loaded.

9.4.6.2 **Notes**

In this scenario `osplconf2c` has been used with the `-x` and `-d` options to create an empty configuraion which allows dynamic loading. The resulting object has been included in the supplied `libddsos.out`. If desired, the end user could create a new `libddsos.out` based on `libddsos.a` and a generated file from `osplconf2c` without the `-x` option, in order to statically link some services, but also allow dynamic loading of others if the built-in xml is later overridden using a file URI. (See Section 7.2.1.4, *Overriding OpenSplice configuration at runtime*, on page 37.)

9.4.7 **The osplconf2c tool**

`osplconf2c` is required for example and user applications.

`osplconf2c` is a tool which processes the OpenSplice configuration XML, and produces a source file to be compiled and linked into the final image. It contains the data from the XML file, as well as any environment variables that you require to configure OpenSplice and references to the symbols for the entry points of the OpenSplice services.

Environment variables can be added using the `-e` option. For example, you would use the `-e "OSPL_LOGPATH=/xxx/yyy"` option if you wanted the logs to be placed in `/xxx/yyy`.

`osplconf2c` is run automatically by the example projects.

9.4.7.1 **Overriding OpenSplice configuration at runtime**

You can override the OpenSplice configuration XML provided to `osplconf2c` at runtime by specifying the URI of a file when starting `ospl_spliced` on the target; for example:

```
ospl_spliced "file:///tgtsvr/ospl.xml"
```

i It should be noted, however, that the `osplconf2c` will have generated references to the symbols for the services which are specified in the xml file when it started, and only those services may be used in the new configuration, as other services will not be included in the image.

9.4.7.2 **The osplconf2c command**

Usage

```
osplconf2c -h
osplconf2c [-u <URI>] [-e <env=var> ]... [-o <file>]
```

Description of options

-h, -?   List available command line arguments and give brief reminders of their functions.

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

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

-e *\<env=var\>*Environment setting for configuration of OpenSplice

e.g. `-e "OSPL_LOGPATH=/xxx/yyy"`

9.4 Running the Examples (All linked in one complete DKM - recommended)

10 *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.*

10.1 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.

10.1.1 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/flashed 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*.)

10.1.2 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 mmstat tool can be controlled via telnet on port 2323 (by default).

10.2 PingPong Example

(Please refer to 5.5, *Examples*, on page 27 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 5* below:
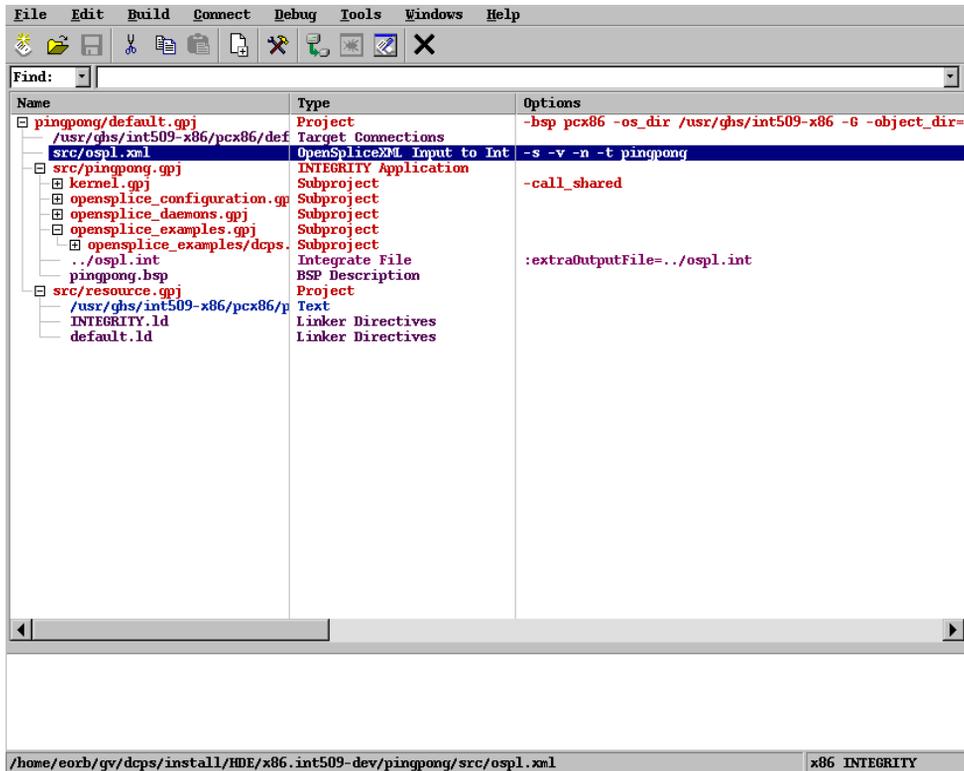
Figure 5 Integrity: project defaults

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 `ospl_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 `ospl.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 5.5, *Examples*, on page 27), if `ospl_projgen` is run with the **-t chat** argument.

10.3 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.

10.3.1 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 6* below.
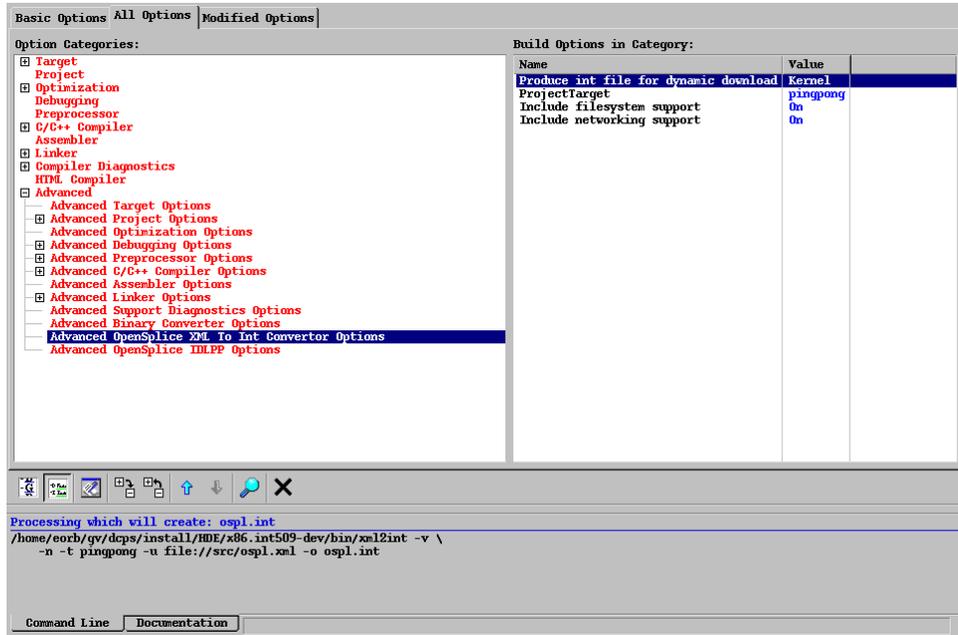
Figure 6 Integrity: changing project options in Multi

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 10.4.1, *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 10.6, *Amending OpenSplice DDS Configuration with Multi*, on page 68, 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.

10.4 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.

10.4.1 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.

10.4.2 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 60) 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                    ResourceStore
                Name                    ResCon
                OtherObjectName         DDS_Connection
        EndObject

        Object  11
                Link                    ResourceStore
                Name                    ConnectionLockLink
                OtherObjectName         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 7* below:

PRISMTECH
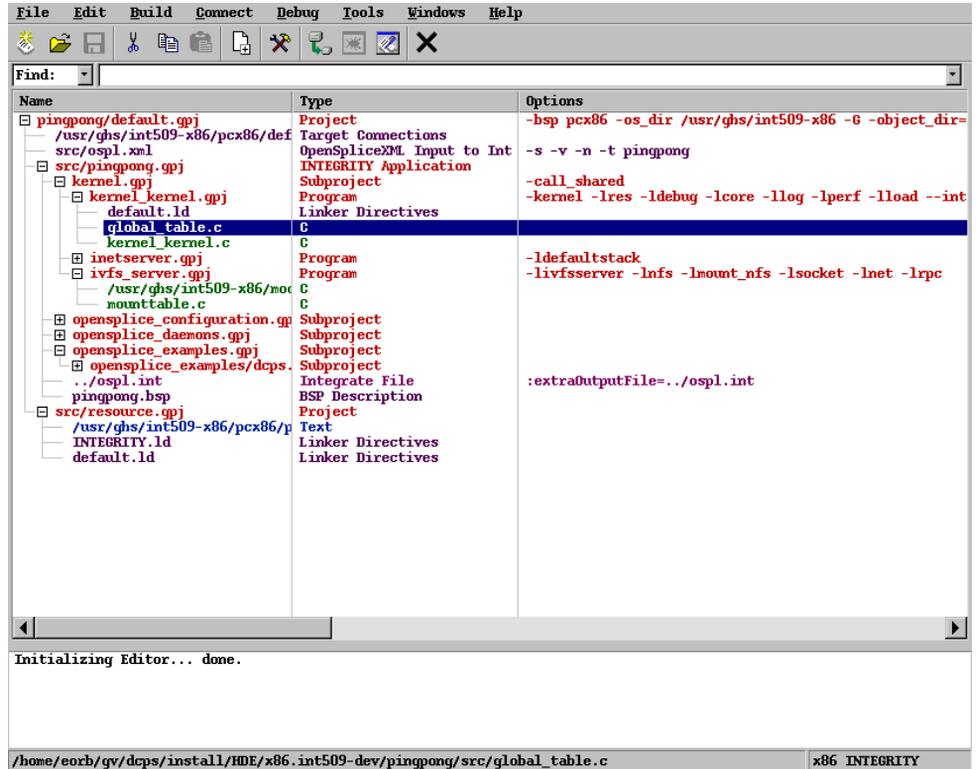
Figure 7 Integrity: changing `global_table.c` and `mounttable.c`

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

10.5 **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 ospl.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!*

10.6 **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 8* below:
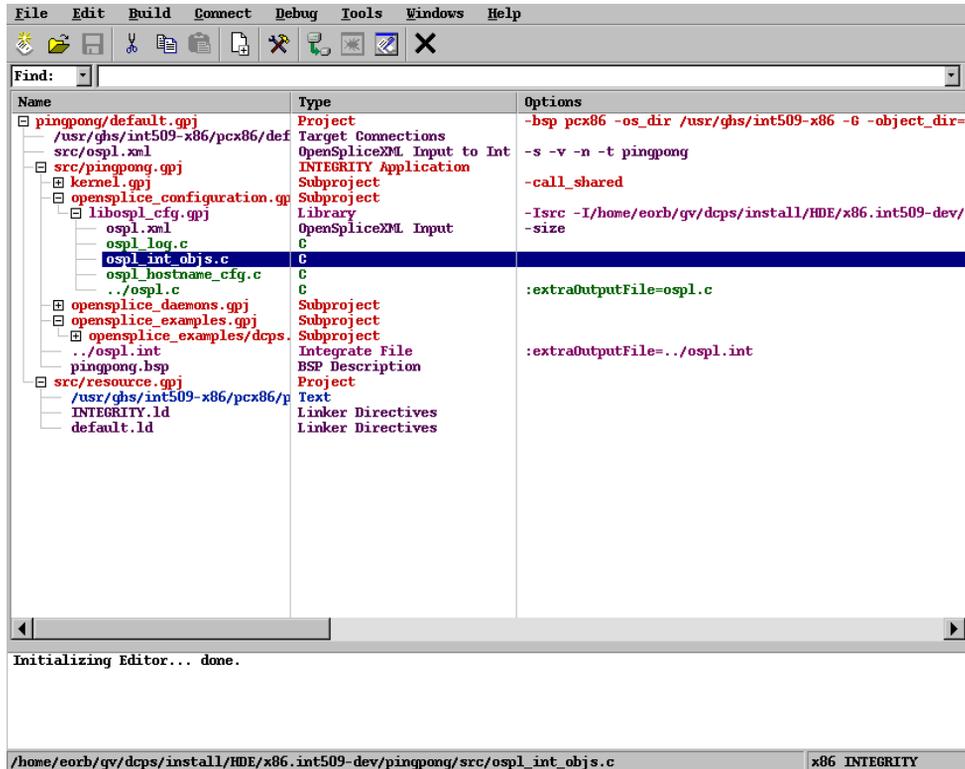
Figure 8 Integrity: changing OpenSplice DDS configuration in Multi

⚠️ 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 5.4, *Configuration*, on page 24 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.)

11 *Windows CE*

This chapter provides a brief description of how to deploy OpenSplice DDS on Windows CE.

11.1 **Prerequisites**

OpenSplice DDS requires certain environment variables to be present; as Windows CE does not support traditional environment variables, these are simulated by creating registry entries which contain the required data. References in this chapter to 'environment variables' are therefore actually references to values in the Windows CE registry.

The environment variables expected by OpenSplice DDS are:

PATH

The PATH variable must include the directory containing the OpenSplice DDS executables that may be launched by the ospl utility.

OSPL_URI

This variable contains the location of the default ospl.xml configuration file which is used when not otherwise specified.

The descriptions in this chapter assume that the values shown in the table below have been added to the registry key

```
HKEY_LOCAL_MACHINE\Software\PrismTech\OpenSpliceDDS\
<OpenSpliceVersion>
```

Windows CE Registry keys

Name	Type	Data
PATH	REG_SZ	\NAND Flash\OpenSpliceDDS\ <OpenSpliceVersion>\HDE\armv4i.wince
OSPL_URI	REG_SZ	file://NAND Flash/OpenSpliceDDS/ <OpenSpliceVersion>/HDE/armv4i.wince/ etc/config/ospl.xml

⚠ All OpenSplice DDS dynamic link library (dll) files must also be copied into the \Windows directory on the Windows CE device prior to deployment.

11.2 Setting Registry Values with a CAB File

In development, a CAB file can be used to register the necessary variables in the registry. Place the CAB file in the Cold Boot directory on the target device (*i.e.* \NAND Flash\ColdBootInit) to make the registry settings available as soon as the device has booted.

11.2.1 Alternatives to CAB file

Microsoft's Windows CE Remote Registry Editor can be used instead of a CAB file to set the necessary registry values. Alternatively, PrismTech also provides a convenient method of editing the registry variables by way of the ospl utility using the getenv and putenv parameters (described below).

Please refer to Microsoft's Windows CE documentation for detailed information about CAB files and the Remote Registry Editor.

11.3 The OpenSplice DDS Environment

OpenSplice DDS requires the contents of the bin, lib and etc directories from within the OpenSplice DDS installation to be available on the Windows CE target hardware. For development purposes, Microsoft's ActiveSync can be used to load these on to the target system. The following description assumes that the bin, lib and etc directories have been copied from the OpenSplice DDS installation onto the target at the following location:

```
\NAND Flash\OpenSpliceDDS\<OpenSpliceVersion>\HDE\armv4i.wince
```

For simplicity the whole OpenSpliceDDS installation directory can be copied to the \Nand Flash directory.

The following description explains deployment on Windows CE by using the Windows CE console. It is assumed that the console's PATH variable has been set to point to the directory containing the OpenSplice DDS executables. For example:

```
PATH \NAND Flash\OpenSpliceDDS\<OpenSpliceVersion>\HDE\
armv4i.wince\bin;%PATH%
```

(All OpenSplice DDS dynamic link library (dll) files must have been copied into the \Windows directory on the Windows CE device prior to deployment.)

When running OpenSplice executables on the command prompt, it is useful to redirect any output to text files by using the '>' operator.

If the PATH and OSPL_URI variables have not already been set *via* a CAB file on device boot up, use the following commands to set those values manually:

```
ospl putenv PATH "\NAND Flash\OpenSpliceDDS\<OpenSpliceVersion>\
HDE\armv4i.wince\bin" > osplputenv-path.txt
```

PRISMTECH

```
ospl putenv OSPL_URI
"file://NAND Flash/OpenSpliceDDS/<OpenSpliceVersion>/
HDE/armv4i.wince/etc/config/ospl.xml" > osplputenv-ospluri.txt
```

The values can be checked if required by using Microsoft's Windows CE Remote
Registry Editor, or by running the `ospl getenv` command:

```
ospl getenv PATH > osplgetenv-path.txt
```

11.4 Secure Networking

The secure networking service uses OpenSSL for cryptography support. To use this
feature, the library `libeay32.dll` is required; it must be copied to the `\Windows`
directory on the Windows CE device.

OpenSplice is tested against OpenSSL version 0.9.8i. This may be built as described
below.

11.4.1 Building OpenSSL for Windows CE 6.0

This section describes the steps required to get an OpenSSL build for Windows CE.
The version of OpenSSL used is 0.9.8i. The third-party library `wcecompat` is used,
which also has to be built manually for Windows CE 6.0.

(The description that follows is based on the one given at
http://blog.csdn.net/sooner01/archive/2009/06/22/4289147.aspx.)

11.4.1.1 Prerequisites

The following are needed to make an OpenSSL build for Windows CE 6.0:

• Microsoft Visual Studio 2005 (VS2008 might also work but it has not been tested)

• An installed WinCE 6.0 SDK to be targeted. In this description the target SDK is
'`WinCE-GS3Target`'

• Perl

You will need to install Active Perl, from *http://www.activestate.com/ActivePerl*.
(Note that perl by MSYS does not create correct `makefiles`.)

• OpenSSL

The OpenSSL sources can be downloaded from *http://www.openssl.org/*. In this
description we use version 0.9.8i. Other versions might not work with the steps
described here.

• `wcecompat` compatibility library

The `wcecompat` library adds the functionality to the C Runtime Library
implementation of Windows CE which is needed in order to build OpenSSL for
Windows CE. Obtain this from *github.com/mauricek/wcecompat*. Note that you

should *not* download the latest version; browse the history and download the version committed on November 21, 2008 named '`updates for OpenSSL 0.9.9`' with commit number '`f77225b...`'.

11.4.1.1.1 Build `wcecompat`

Step 1: Extract the `wcecompat` download to an appropriate location. In the description below the location `C:\wcecompat` is used but you can use any location you want.

Step 2: Start Visual Studio 2005 and open a Visual Studio 2005 command prompt.

Step 3: Go to the wcecompat directory (`C:\wcecompat`)

Step 4: Set the building environment:

```
set OSVERSION=WCE600

set TARGETCPU=ARMV4I

set PLATFORM=VC-CE

set PATH=C:\Program Files\Microsoft Visual Studio
8\VC\ce\bin\x86_arm;C:\Program Files\Microsoft Visual Studio
8\Common7\IDE;%PATH%

set INCLUDE=C:\Program Files\Windows CE
Tools\wce600\WinCE-GS3Target\include\ARMV4I

set LIB=C:\Program Files\Windows CE
Tools\wce600\WinCE-GS3Target\lib\ARMV4I;C:\Program
Files\Microsoft Visual Studio 8\VC\ce\lib\armv4
```

If you target a different SDK, replace the text '`WinCE-GS3Target`' in the lines above with your own SDK.

Step 5: Call `perl config.pl` to create the makefile configuration.

Step 6: Call `nmake` to build the `wcecompat` library.

Step 7: Exit the command prompt and exit Visual Studio to be sure of starting with a clean environment in the next stage.

11.4.1.1.2 Build OpenSSL

Step 1: Extract OpenSSL to any location you like.

Step 2: Apply the OpenSSL WinCE patch which can be found at *http://marc.info/?l=openssl-dev&m=122595397822893&w=2*.

Step 3: Start Visual Studio 2005 and open a command prompt.

Step 4: Go to your openSSL directory.

Step 5: Set the building environment:

```
set OSVERSION=WCE600

set TARGETCPU=ARMV4I

set PLATFORM=VC-CE
```

```
set PATH=C:\Program Files\Microsoft Visual Studio
8\VC\ce\bin\x86_arm;C:\Program Files\Microsoft Visual Studio
8\VC\bin;C:\Program Files\Microsoft Visual Studio
8\VC\PlatformSDK\bin;C:\Program Files\Microsoft Visual Studio
8\Common7\Tools;C:\Program Files\Microsoft Visual Studio
8\Common7\IDE;C:\Program Files\Microsoft Visual Studio
8\Common\Tools;C:\Program Files\Microsoft Visual Studio
8\Common\IDE;C:\Program Files\Microsoft Visual Studio 8\;%PATH%
```

```
set INCLUDE=C:\Program Files\Microsoft Visual Studio
8\VC\ce\include;C:\Program Files\Windows CE
Tools\wce600\WinCE-GS3Target\include\ARMV4I;C:\Program
Files\Windows CE Tools\wce600\WinCE-GS3Target\include;C:\Program
Files\Microsoft Visual Studio 8\VC\ce\atlmfc\include;C:\Program
Files\Microsoft Visual Studio 8\SmartDevices\SDK\SQL
Server\Mobile\v3.0;
```

```
set LIB=C:\Program Files\Windows CE
Tools\wce600\WinCE-GS3Target\lib\ARMV4I;C:\Program
Files\Microsoft Visual Studio
8\VC\ce\atlmfc\lib\ARMV4I;C:\Program Files\Microsoft Visual
Studio 8\VC\ce\lib\ARMV4I
```

```
set WCECOMPAT=C:\wcecompat
```

If you target a different SDK, replace the text '*WinCE-GS3Target*' in the lines above with your own SDK. Also, change the wcecompat directory to your own if you used a different location.

Step 6: Type perl Configure VC-CE to set up the compiler and OS.

Step 7: Type ms\do_ms to build the makefile configuration.

Step 8: Type nmake -f ms\cedll.mak to build the dynamic version of the library.

11.4.1.1.3 Troubleshooting

If you get the following error message:

```
PTO -c .\crypto\rsa\rsa_pss.c
cl : Command line warning D9002 : ignoring unknown option '/MC'
rsa_pss.c
f:\openssl\openssl98\crypto\rsa\rsa_pss.c(165) : error C2220:
warning treated as error - no 'object' file generated
f:\openssl\openssl98\crypto\rsa\rsa_pss.c(165) : warning C4748:
/GS can not protect parameters and local variables from local
buffer overrun because optimizations are disabled in function
NMAKE : fatal error U1077: '"F:\Program Files\Microsoft Visual
Studio 8\VC\ce\bin\x86_arm\cl.EXE"' : return code '0x2'
Stop.
```

Then remove '*/WX*' in the makefile (ce.mak).

11.5 Deploying OpenSplice DDS

ospl start

This command will start the OpenSplice DDS `splicedaemon` and OpenSplice DDS services specified within the configuration referred to by the `OSPL_URI` variable:

```
ospl start > osplstart.txt
```

A different configuration file can be specified as an additional parameter; for example:

```
ospl start "file://NAND Flash/OpenSpliceDDS/<OpenSpliceVersion>/
HDE/armv4i.wince/etc/config/ospl.xml" > osplstart.txt
```

ospl list

This command will list all the OpenSplice DDS configurations that are currently running on the node.

```
ospl list > ospllist.txt
```

ospl stop

This command will stop the OpenSplice DDS `splicedaemon` and OpenSplice DDS services specified within the configuration referred to by the `OSPL_URI` variable:

```
ospl stop > osplstop.txt
```

A different configuration to be stopped can be specified as an additional parameter; for example:

```
ospl stop "file://NAND Flash/OpenSpliceDDS/<OpenSpliceVersion>/
HDE/armv4i.wince/etc/config/ospl.xml" > osplstop.txt
```

11.6 Using the **mmstat** Diagnostic Tool on Windows CE

To run mmstat, use this command:

```
start mmstat > mmstat.txt
```

To see the full list of options, use this command:

```
start mmstat -h > mmstat-help.txt
```

The mechanism for terminating mmstat on Windows CE is different from other operating systems. All running instances of mmstat can be terminated with the following command:

```
start mmstat -q > mmstat-quit.txt
```

If there are multiple instances of mmstat running, a particular instance can be terminated by specifying the process identifier:

```
start mmstat -q -x <process id>  > mmstat-quit.txt
```

where <process id> is displayed in the output for the particular instance of mmstat.

PrismTech

11.7 **OpenSplice Examples**

Please refer to section 5.5, *Examples*, on page 27, for descriptions of the OpenSplice DDS examples.

11.7.1 **Building the examples**

There is a shortcut to load the examples into Microsoft Visual Studio which can be accessed from **Start > Programs > OpenSpliceDDS <OpenSpliceVersion> armv4i.wince HDE > Examples**.

Once the projects are open in Microsoft Visual Studio, click **Build/Rebuild Solution** at the appropriate level to build the required examples.

Copy the produced executable files to the OpenSplice DDS `bin` directory (*i.e.* `\NAND Flash\OpenSpliceDDS\<OpenSpliceVersion>\HDE\armv4i.winc e\bin`) on the Windows CE device. For the PingPong example the executable files are `Ping.exe` and `Pong.exe`. For the Tutorial example the files are `Chatter.exe`, `MessageBoard.exe`, and `UserLoad.exe`.

As an alternative to using the shortcut and to set up the environment for a new project perform the following steps:

Step 1: Run the OpenSplice command prompt from the **OpenSplice** entry under the Windows **Start** button:

Start > Programs > OpenSpliceDDS <OpenSpliceVersion> armv4i.wince HDE > OpenSpliceDDS command prompt

Step 2: Copy the Windows Microsoft Visual Studio environment variables to the new command prompt. To obtain these, right-click on the **Properties** for the **Visual Studio 2005 Command Prompt** entry located at **Start > Programs > Microsoft Visual Studio 2005 > Visual Studio Tools > Visual Studio 2005 Command Prompt**, and paste the **Shortcut Target** entry into the OpenSplice command prompt. For example this could be

```
%comspec% /k ""C:\Program Files\Microsoft Visual Studio 8\
VC\vcvarsall.bat"" x86
```

Step 3: Start Microsoft Visual Studio in this prompt:

```
devenv
```

Step 4: Open the solution file at

```
<OpenSpliceDDSInstallation>/examples/examples.sln
```

11.7.2 **Deploying the PingPong example**

Start OpenSplice DDS as described above. The Ping and Pong executables can then be started as follows:

```
start pong PongRead PongWrite > pong.txt
start ping 100 100 m PongRead PongWrite > ping.txt
```

The `ping.txt` file produced should contain the expected Ping Pong measurement statistics for 100 cycles. The `Pong` executable can be shut down by running the `ping shutdown` command:

```
start ping 1 1 t PongRead PongWrite > ping-shutdown.txt
```

11.7.3 Deploying the Tutorial example

Start OpenSplice DDS as described above. The Tutorial executables can then be started as follows:

```
start UserLoad > userload.txt
start MessageBoard > messageboard.txt
start Chatter 1 John > chatter.txt
```

The `messageboard.txt` file produced should contain the messages received from the `Chatter` executable. The `MessageBoard` executable can be terminated by running `Chatter` again with the `-1` option:

```
start Chatter -1 > chatter-shutdown.txt
```

CHAPTER

12 *PikeOS POSIX*

This chapter provides a brief description of how to deploy OpenSplice DDS on PikeOS.

12.1 How to Build for PikeOS

For this target, OpenSplice must be configured in the single-process mode and executables must be statically linked. Also, to avoid requiring filesystem support, the `ospl.xml` configuration file is built into the executable. A tool named `osplconf2c` is provided to generate the code required for this.

When executed with no arguments, `osplconf2c` takes the configuration file specified by `OSPL_URI` (only `file:` URIs are supported) and generates a C source file named `ospl_config.c`. The URI and filename can be specified using the `-u` and `-o` options respectively. The generated code should be compiled and linked with each executable. It comprises an array representing the configuration file and also the entry points which allow the configured services to be started as threads.

OpenSplice is built against the BSD POSIX support in PikeOS (`bposix`), and also uses the `lwIP` networking facility. When linking an executable for PikeOS deployment it is necessary to specify the system libraries as follows:

```
-llwip4 -lsbuf -lm -lc -lp4 -lvm -lstand
```

The OpenSplice libraries for each configured service (networking, cmsoap, *etc.*) also have to be linked in.

12.2 Deployment Notes

When setting up an integration project in which to deploy an OpenSplice executable the following items should be configured:

- Networking should be enabled using the `LwIP` stack.
- The amount of available memory will generally need to be increased. We recommend a minimum of 48MB for OpenSplice partitions.

In CODEO:

Step 1: Create a new integration project based on the *devel-posix* template.

Step 2: Edit `project.xml.conf`:

1. Configure networking for the `muxa` in the service partition, if required.

2. Enable `LwIP` in the POSIX partition, set its device name and IP addresses.

3. Add a dependency in the POSIX partition on the network driver file provider.

Step 3: Edit `vmit.xml`:

> In the POSIX partition, set the `SizeBytes` parameter in *`Memory Requirement->RAM_[partition]`* to at least `0x03000000`.

Step 4: Copy your OpenSplice executable into the project's target directory and build the project.

12.3 **Limitations**

Multicast networking is not supported on PikeOS POSIX partitions. Consequentially, the DDSI networking service is not supported. The native networking service is supported in broadcast mode.

12.4 **PikeOS on Windows Hosts**

Developing with OpenSplice for PikeOS on Windows differs from OpenSplice on other Windows-hosted platforms. This is because PikeOS uses Cygwin to present a Unix-like environment for cross-development. OpenSplice development therefore follows similar practice to the Unix-hosted editions; for example, you would start a session by entering a `bash` shell and sourcing the `release.com` file.

However, the OpenSplice tools (`idlpp`, `osplconf2c`, `ospltun`, *etc.*) are built as Windows executables or will be running in Java for Windows and so need to be run with Windows-style pathnames. This is handled using the `cygpath` command as seen in the `release.com` script and the example `makefiles`. Tuner and Configurator may be launched from the **Start** menu as usual.

12.4.1 **Building the examples**

This proceeds as would be expected in a Unix environment. Assuming that OpenSplice and PikeOS are installed in the default locations:

• Start Cygwin `bash`

1. `declare -x PIKEOS_HOME=/opt/pikeos-3.1`
2. `declare -x PATH=$PATH:$PIKEOS_HOME/cdk/ppc/oea/bin`
3. `. /opt/OpenSpliceDDS-V6.2.3/HDE/ppc.pikeos3/release.com`
4. `cd $OSPL_HOME/examples`
5. `make`

12.4.2 Using a custom LwIP

The example `makefiles` are set up by default to link against the build of *LwIP* distributed with PikeOS; however, in some circumstances it may be useful to rebuild *LwIP*. In this case, set the environment variable `LWIP_HOME` to a directory containing `liblwip4.a`, `lwipconfig.o` and `lwipopts.h` prior to building the examples.

CHAPTER

13 *ELinOS*

This chapter provides notes about deploying OpenSplice DDS on ELinOS.

13.1 Deployment notes

OpenSplice may be deployed in an ELinOS system.

The following ELinOS features should be enabled:

- Kernel support for System-V IPC
- Use full `shmem` filesystem
- `tmpfs`

The following system libraries are required:

- `stdc++`
- `pthread`
- `rt`
- `dl`
- `z`
- `m`

⚠ We do not recommend running with less than 32M of system memory.

13.2 Limitations

⚠ ELinOS partitions within PikeOS are supported, but DDSI networking will not function as it requires a multicast-capable interface.

The native networking service is supported in a broadcast configuration.