OpenSplice DDS

Version 4.1

Getting Started Guide

PRISMTECH

OpenSplice DDS

GETTING STARTED GUIDE

Part Number: OS-GSG

Doc Issue 26, 17 February 2009

Copyright Notice

Contents

Table of Contents

Preface

About the Getting Started Guide

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

The *Getting Started Guide* contains:

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

Intended Audience

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

Conventions

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


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

i
Item contains helpful hint or special information.

WIN
Information applies to Windows (e.g. NT, 2000, XP) only.

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

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 to other parts of the document, e.g. *Contacts* on page viii, behave as hypertext links: users can jump to that section of the document by clicking on the cross reference.

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

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

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

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

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

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

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

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

Contacts

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

Corporate Headquarters

PrismTech Corporation
6 Lincoln Knoll Lane
Suite 100
Burlington, MA
01803
USA

Tel: +1 781 270 1177
Fax: +1 781 238 1700

European Head Office

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

Tel: +44 (0)191 497 9900
Fax: +44 (0)191 497 9901

Web: *http://www.prismtech.com*
General Enquiries: *info@prismtech.com*

About OpenSplice DDS

CHAPTER

1 Why OpenSplice DDS

1.1 What is OpenSplice DDS?

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

1.2 Why Use It?

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

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

CHAPTER

2 Product Details

2.1 Key Components

OpenSplice DDS's include the key components listed here.

2.1.1 Services

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

2.1.2 Tools

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

2.2 Key Features

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

2.3 Language Bindings

OpenSplice DDS is available for the following languages:

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

- Java

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

The C++ language binding, referred to as *CORBA C Plus Plus* (CCPP) can only be used in combination with a C++ ORB. The ORB and compiler combination used to generate the default CCPP library (supplied with the OpenSplice release) is mentioned in the *Release Notes*. See Section 5.6, *Tailoring the C++ API*, for additional information.

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

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

2.4 Platforms

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

Using OpenSplice DDS

CHAPTER

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 instruction on running the OpenSplice examples. For managers, administrators, and developers to gain an initial understanding of the product, as well as for product installers for installing and administering OpenSplice.
Deployment Guide	A complete reference on how to configure and tune the OpenSplice service.
Tutorial Guide	A short course on developing applications with OpenSplice in C. Includes example code in C, C++ and Java.

OpenSplice DDS Documentation Set (Continued)

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

CHAPTER

4 Information Sources

4.1 Product Information

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

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

4.1.1 Knowledge Base

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

http://kb.prismtech.com

4.1.2 Additional Technical Information

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

http://www.prismtech.com

4.2 Support

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

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

Installation and Configuration

CHAPTER

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 Prerequisites

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

5.2 Installation for UNIX Platforms

Step 1: Install OpenSplice DDS

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

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

   where

   <version> - the OpenSplice DDS version number, for example *V2.0*
   <platform> - the platform architecture, for example *sparc* or *x86*
   <os> - the operating system, for example *solaris8* or *linux2.6*
   <E> - the environment, either *HDE* or *RTS*
   <ext> - the platform executable extension, either *bin* or *exe*

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

Step 2: Configure the OpenSplice DDS environment variables

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

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

This step performs all the required environment configuration.

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

5.3 Installation for Windows Platforms

Step 1: Install OpenSplice DDS

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

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

   where

   <version> - the OpenSplice DDS version number, for example *V2.0*
   <platform> - the platform architecture, for example *sparc* or *x86*
   <os> - the operating system, for example *solaris8* or *linux2.6*
   <E> - the environment, either *HDE* or *RTS*
   <ext> - the platform executable extension, either *bin* or *exe*

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

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

5.4 Configuration

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

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

The configuration file defines and configures the following OpenSplice services:

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

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

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

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

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

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

The network service may use separate channels, each with their own name and their own parameters (for example the port-number, the queue- size, and, if multicast enabled, the multicast address). Channels are either reliable (all data flowing through them is delivered reliably on the network level, regardless of QoS settings of the corresponding writers) or not reliable (all data flowing through them is delivered at most once, regardless of QoS settings of the corresponding writers). The idea 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=”0.5”>5.0</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>
            <NameSpace durabilityKind="Durable"
                alignmentKind="Initial_and_Aligner">
                <Partition>*</Partition>
            </NameSpace>
        </NameSpaces>
        <Persistent>
            <StoreDirectory>/tmp/Pdata</StoreDirectory>
        </Persistent>
    </DurabilityService>
</OpenSplice>
```

5.5 Examples

The way to build and run the examples is dependent on the Platform you are using. For Unix/Linux based systems there is a shell script called *BUILD* which is available for all supported languages. For Windows there is a similar batch file called ‘BUILD.bat’ for all supported languages. Be aware that you may have to tailor some of the environment variables in these batch files to point to the correct location of for example the Microsoft Platform SDK. As an alternative to these batch files, the Windows release also has Project and Solution files (Microsoft Visual Studio 8

compliant) available for the C and C++ languages (for each example there is a separate project file to compile it. Besides that, a single Solution file is available to compile all examples at the same time).

5.5.1 The PingPong Example

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

Step 1: Build the example applications

For the StandAlone C API (SAC):

1. Change to the `$OSPL_HOME/examples/dcps/standalone/C/PingPong` directory (on Windows use the command prompt).
2. Check the usage and functioning by inspecting the `ping.c` file.
3. Build the applications using `sh ./BUILD` on the command line on Linux or Solaris platforms or execute `BUILD.bat` from the command prompt on the Windows platform. Alternatively for Microsoft Visual Studio users, *double click* the provided Project file.
4. Run the applications running `sh ./RUN` on the command line on Linux or Solaris platforms or execute `RUN.bat` within the command prompt on the Windows platform.

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

1. Change to the `$OSPL_HOME/examples/dcps/CORBA/C++/OpenFusion/PingPong` directory (on Windows use the command prompt).
2. Check the usage and functioning by inspecting the `ping.cpp` file.
3. Build the applications using `sh ./BUILD` on the command line on Linux or Solaris platforms or execute `BUILD.bat` in a command prompt on the Windows platform. Alternatively for Microsoft Visual Studio users, double click the provided Project file.
4. Run the applications running `sh ./RUN` on the command line on Linux/Solaris platforms or execute `RUN.bat` in a command prompt on the Windows platform.

For the StandAlone Java API (SAJ)

1. Change to the `$OSPL_HOME/examples/dcps/standalone/Java/PingPong` directory (on Windows use the command prompt).

2. Check the usage and functioning by inspecting the *ping.java* file.
3. Build the applications using *sh ./BUILD* on the command line on Linux or Solaris platforms or execute *BUILD.bat* in a command prompt on Windows.
4. Run the applications running *sh ./RUN* on the command line on Linux or Solaris platforms or execute *RUN.bat* in a command prompt on Windows.

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

1. Change to the *$OSPL_HOME/examples/dcps/standalone/C++/PingPong* directory (on Windows use the command prompt).
2. Check the usage and functioning by inspecting the *ping.cpp* file.
3. Build the applications using *sh ./BUILD* on the command line on Linux/Solaris platforms or execute *BUILD.bat* in a command prompt on Windows. Alternatively for Microsoft Visual Studio users, *double click* the provided Project file.
4. Run the applications running *sh ./RUN* on the command line on Linux or Solaris platforms or execute *RUN.bat* in a command prompt on Windows.

5.5.2 The Tutorial Example

The tutorial example consists of three separate executables that constitute a very primitive Chatter application. The OpenSplice infrastructure should manually be started before running the Chatter applications executables (see Section 5.5.3, *Using the OpenSplice Tools*). Each Chatter executable can be started using different configurations provided each is running in their own, separate terminal window: this is to avoid having their screen outputs intermingled together in the same terminal window.

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

Chatter(.exe) [userID] [userName]
: Starts sending Chat messages for a user with the specified ID and name. Ensure the ID number is a unique and not used by any other user.

MessageBoard(.exe) [userID]
: Starts a MessageBoard that displays the Chat messages it receives from all users, except for the user with the specified ID.

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

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

Building the Tutorial examples is very similar to building the PingPong examples: for Unix/Linux based systems there is a *BUILD* script that compiles and links the executables; for Windows there are similar *BUILD.bat* files. Also, Project files are available for Microsoft Visual Studio for C and C++.

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

On Unix or Linux-based platforms use:

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

On Windows-based platforms use:

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

5.5.3 Using the OpenSplice Tools

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

Step 2: Manually start the OpenSplice infrastructure

1. Enter *ospl start* on the command line.[1] This starts the OpenSplice services.
2. The default configuration file that comes with OpenSplice, which is network enabled, is used.
   a) The default configuration uses UDP-broadcast.
   b) UDP-multicast can be used instead of UDP-broadcast, but requires that a multicast address is added in the configuration file (see Section 5.4, *Configuration*, on page 17).
3. These log files may be created in the current directory when OpenSplice is started:
   a) *ospl-info.log* - contains information and warning-reports

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

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

Step 3: Start the OpenSplice Tuner Tool

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

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

3. The OpenSplice system can now be monitored.

Step 4: Experiment with the OpenSplice tools and applications

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

Step 5: Manually stop the OpenSplice infrastructure

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

5.6 Tailoring the C++ API

The pre-compiled C++ API that is delivered with OpenSplice DDS only works in combination with a specific ORB and compiler combination, as stated previously. If another compiler version or another ORB is required, then a custom C++ API library can be built using the source code provided in the *`$OSPL_HOME/custom_lib/ccpp`* directory. Detailed instructions on how to create the custom API are provided in the *`README.txt`* file located in *`$OSPL_HOME/custom_lib/ccpp`*.

CHAPTER

6 Licensing OpenSplice

6.1 General

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

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

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

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

6.1.1 Development and Deployment Licenses

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

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

6.2 Installing the License File

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

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

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

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

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

6.3 Running the License Manager Daemon

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

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

To run the license manager, use the following command:

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

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

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

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

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

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

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

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

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

```
27001@ultra5
```

6.3.1 Utilities

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

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

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

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

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

```
%  lmutil lmhostid
```

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

```
8ac86d5
```

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

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