

# **Pramati ISV Guide**

**Pramati ISV Kit: User Guide**

1230-008

November 2002

**Pramati Technologies**

301 White House, Begumpet

Hyderabad 500 016

India

**info@pramati.com**

**www.pramati.com**

Communications Design Group, Pramati Technologies.

Made in India.

Copyright 2002 Pramati Technologies. All rights reserved. Pramati is a registered trademark of Pramati Technologies in United States and India.

Java and all Java based marks are trademarks or registered trademarks of Sun Microsystems, Inc., in United States and other countries. WebLogic is a registered trademark of BEA Systems, Inc. Informix Cloudscape is a registered trademark of Informix Corporation. Oracle and Oracle 9i are registered trademarks of Oracle Corporation. ANT is a trademark of Apache Software Foundation. Red Hat is a registered trademark of Red Hat, Inc. Netscape is a registered trademark of Netscape Communications Corporation. Opera for Windows is the registered trademark of Opera Software, Inc. AIX is a trademark or registered trademark of IBM Corporation. UNIX is a registered trademark of the Open Group. Apple, Mac, Power Mac and Mac OS are registered trademarks of Apple Computer, Inc. Microsoft, Windows, Windows NT, Windows 2000 and Internet Explorer are Registered Trademarks of Microsoft Corporation. All other trademarks may be the registered trademarks of their respective owners.

No third party intellectual property right liability is assumed with respect to the information contained herein. While the information in this publication is believed to be accurate, Pramati Technologies makes no warranty of any kind to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Pramati Technologies shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material. Pramati Technologies assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of Pramati Technologies.

# Contents

<b>1</b>	<b>Typical ISV Requirements</b> .....	5
	Packaging and distributing the product bundle .....	5
	Typical scenario .....	6
	Server licensing .....	7
	Rebranding .....	8
	What's redistributable .....	9
	Documentation .....	9
	ISV Customer requirements .....	9
	Key terminology .....	10
<b>2</b>	<b>Silent Installation of Pramati Server</b> .....	13
	Requirements .....	13
	Installation .....	14
<b>3</b>	<b>Setting up Java Client</b> .....	17
	Requirement at client machines .....	17
<b>4</b>	<b>Customizing the Console</b> .....	19
	Levels of customization .....	19
	Creating Custom Views .....	19
	Customizing pages .....	21
<b>5</b>	<b>Creating and Configuring Pramati Server Instances Using XML</b> .....	23
	Introduction .....	23
	Understanding Pramati Server instances .....	23
	Key terminology .....	24
	Pramati Server configuration files .....	25
	Types of Server instances that can be configured .....	26
	Structure of Nodes Directory .....	27
	Configuration File Dependency .....	29
	Using the NodeCreator .....	30
	Custom node configuration .....	34
	Enabling embedded Message Server in a Standalone Server .....	37
	Restricting services in a Standalone J2EE Server .....	38
	Cluster Definition .....	40
	Creating a Cluster Instance (steps common to all types of clusters) .....	41
	Setting up a J2EE Cluster .....	43
	Setting up an HA Message Server Cluster .....	47
	Updating cluster configuration using XML (when DB is used for configuration) .....	48
	Overview .....	51
	Architecture .....	52
	Kernel .....	53
	Core Services .....	53
	Core Components .....	54
	Services framework features .....	58
<b>6</b>	<b>Using Pramati Server Node Creator Utility</b> .....	59
	Usage Objective .....	63
<b>7</b>	<b>Hosting Scenarios</b> .....	67
	Sample configuration layouts .....	71

<b>8</b>	<b>Deployment Descriptors</b> .....	73
	DTD for EJB 1.1 and 2.0 Deployment Descriptor .....	73
	DTD for Web Deployment Descriptor .....	75
	DTD for Pramati J2EE Server .....	76
	Pramati OR Map DTD .....	86
<b>9</b>	<b>Appendix 1: Installer Builder</b> .....	93
	Quick configuration .....	93
<b>10</b>	<b>Appendix 2: Extending the Installer Builder</b> .....	99
	Custom validation .....	99
	Setting up custom panels .....	100
	Building the installer .....	100
<b>11</b>	<b>Appendix 3: Developing an ISV Product Installer</b> .....	103
	Java requirement .....	103
	How to prepare the product bundle .....	103
	Directory structure at customer site .....	104
	Distributing license keys .....	104
	Customized administration .....	105
<b>12</b>	<b>Appendix 4: Sample ISV Product Installer</b> .....	107
	The foobank Sample .....	107
	Modifying the sample .....	107

# 1

## *Typical ISV Requirements*

---

With increasing popularity of J2EE as the *de facto* middleware standard, product vendors are coming to rely on this platform for delivering effective solutions. Bundling a J2EE application server with a solution ensures that an ISV's customers get exactly what they need, with minimum life-time investment in people and infrastructure.

### **Packaging and distributing the product bundle**

There are definite advantages for an ISV in embedding Pramati Server with its product:

#### **Single, manageable distribution unit**

Ship your complete solution as a single downloadable or CD image. The value perception of the solution provider stems from his ability to provide a complete solution and not a set of loosely coupled components that at best offer a patchy solution.

#### **Single installation process**

The need for a single, manageable distribution unit leads to the need for an integrated installation process. The latter ensures that your customers can quickly start using the application and reduces the degree of involvement of resources in the installation process.

#### **Pre-deployed application**

Server vendors provide proprietary tools for deploying an application. A bundled server that can be installed cleanly and quickly ensures that neither your field resources nor customers require in-depth knowledge of J2EE technologies to deploy applications. Fast deployment translates to more satisfied customers.

## Simple configuration

Customers can configure an application as well as the server through one simple, intuitive interface.

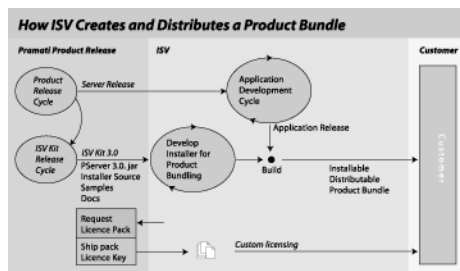
## Integrated license management

This is a crucial aspect in your business. The product will be sold under different licenses, so you should be able to get a configurable licensing framework from your application server. For example, you may distribute free evaluation copies of your product to prospective customers.

## Typical scenario

You may plan to distribute the product and the J2EE application widely. To ensure a great out-of-the-box experience for evaluators and customers, you need to minimize the pain in installing the application and server and in getting the application started.

When the application has been prepared, the ISV prepares an integrated, distributable product binary. This guide explains how to create such a bundle.



The software bundle can be shipped to customers over the Internet or via other media. The customer is required to enter the appropriate License Key and proceed with installing the product and application. The ISV can rebrand and customize the functionality of the Pramati Server Management Console to suit application requirements. Relevant management pages from the Console like JSPs can be embedded in the customer application.

## Server licensing

The benefit to ISVs lies in providing a complete infrastructure for solutions without the overhead of having to provide the customer with an application server license as well. This helps customers to remain focused on using the application rather than worry about managing the infrastructure. Pramati enables this with its "Limited License" facility that allows a end user to use the Server license only with the application provided by the ISV.

Pramati Server usage is controlled by a sixteen digit, case-sensitive, alphanumeric license key that must be supplied during installation so that it can be accessed at runtime.

The ISV's application is bundled with Pramati Server as an ISV product bundle that is marketed, priced and distributed as a single software unit. Pramati's offering to ISVs covers not only technological capability to deliver a single, off-the-shelf, shrink-wrapped product, but also correct licensing terms.

### How licensing works

Pramati licenses ISVs to reproduce its server software in binary form as part of the product bundle and to distribute the product bundle. This is enabled through the "Pramati Product Bundling and Reselling Agreement".

To enable ISVs to achieve volume sales with minimum operational difficulty, Pramati makes its Limited Licenses available in large packs (called Multipacks) as well as small units (Valupacks) so that ISVs can purchase from Pramati in bulk and then redistribute to their customers.

In addition to the License, Pramati also offers customized consulting services to help ISVs make the most of the ISV Kit. This includes extending the packaging framework to meet specific requirements. Pramati Server is licensed to end users through a standard Pramati End User License Agreement (Pramati EULA).

### **Ways of working with license keys**

To install and start Pramati Server, an end user requires a license key that is supplied with other application-specific keys.

When an ISV signs the product bundling agreement and places an order for a Multipack or Valuepack under that agreement, Pramati ships the ordered licenses to the ISV as a set of certificates, each bearing one License Key. During installation and at runtime, this license key must be specified.

### **Rebranding**

Pramati ISV Bundling Agreement permits rebranding of the installation process only and not Pramati Server.

Pramati Server provides a customisable installer that can be extended by the ISV to provide a custom look and feel. It also enables the deployer to provide installation-specific information, such as database location, password, security roles and user names.

Pramati Server also enables ISVs to wrap their installers with other installation software. In this mode, an ISV can perform a "silent install" of Pramati Server.

After an application has been deployed, end users can monitor it through Pramati Server's Web-based management console. The console can be customized to suit specific needs.

## What's redistributable

### Pramati Server binary

The Pramati Server installable binary Pserver30.jar is distributed whole. Contents of this JAR as listed in the table may not be distributed individually.

HTTP Server	For serving Web content
Servlet Engine	Dynamic web content and presentation layer component engines conforming to Servlets 2.2
Offline JSP Compiler	Tool that compiles JSP pages at the time of development to facilitate deployment
EJB Container	The business logic driver for serving EJB 2.0 components including Session Beans, Entity Beans and Message Driven Beans
J2EE Server	The complete set of J2EE-mandated services including Transaction Management, Persistence Management and Security Management
JMS Server	Full-feature JMS 1.0.2 implementation
Server Management Console	JMX-based, web-enabled management console for users to configure their application once deployed

## Documentation

Along with the embedded Server, an ISV needs to distribute the following documentation in electronic form:

- Pramati Server Getting Started Guide
- Pramati Server Deployment Guide
- Pramati Server Administration Guide
- Pramati Server Security Guide

## ISV Customer requirements

Some customers may require the JRE to be provided and some others may require the JDK. The ISV needs to prepare the deployable archive accordingly.

## JDK

If the J2EE application has raw JSP pages, which need to be compiled at runtime, JDK may be required on the host. This is a common scenario where only the deployment mapping XMLs are required to be in the EAR. If the ISV wants to permit re-mapping of deployment information at the customer site, then JDK is required to be on the host.

## JRE

If a customer wants to use JRE, pre-compiled generated code must be shipped by the ISV as a prepared archive (.par file) which can readily be deployed on Pramati Server. The Deploy Tool by default does not precompile the generated code when making a .par file. When JRE is used, deployment information cannot be modified at the customer site.

## Key terminology

ISV	Independent Software Vendor who is the developer and owner of the application to be bundled with the Server.
Customer	The ISV's Customer who wishes to license the application from the ISV and requires a J2EE Server to run the application.
Server	Pramati Server
Product Bundle	The ISV's bundle that includes the application, related files and Server
Application	The ISV's independent solution that is deployable on Server. To run the solution, a Server instance must first be configured.
Installation Program	The source for the ISV Product Installer built using generic Installer Builders.
ISV Product Installer	The distribution unit of the product bundle that is output by the Installer Builder. It runs as an installation wizard at the customer site to install and set up the ISV application.
Sample Installer	The Sample Installer shipped by Pramati. As a most basic configuration, the ISV may adapt this Installer to make it his own.
Pramati Installer Builder	Wizard that is run by the ISV to construct the Installer from the XMLs and sources provided with the Kit. Pramati provides a custom Installer Builder based on IzPack, the Java Software Installer Builder product that is used to distribute application files in single JAR files.

---

Installer Builder	Generic Installer Builders like IzPack, Zero G and InstallShield.
Rebranding	The process of modifying the look and feel of the installation and configuration of product bundle to deliver an ISV-branded offering to the customer. Pramati's name and brand can be made transparent during installation and configuration.
ISV Kit	A tool kit containing Pramati Server, tools, samples and documentation that help the ISV create an ISV Product Installer.
License Key	Pramati Server license key that may be required by the customer to install and run a Server instance.
Proprietary Key	The ISV's custom license key that it would process at installation time at customer's location. The ISV will build processing logic that encodes the License Key into the Proprietary Key.
Silent Installation	The process by which the Server installation happens transparently, reading all required installation information like directories and Java home from a specified XML file.

---



# 6

## *Silent Installation of Pramati Server*

---

This chapter provides information about installing the Server 'silently' under the covers as part of the ISV's product installation

### **Requirements**

Prior to installation, the ISV's installation program must ensure the system requirements to install Pramati server. Check the latest release notes for updated requirements.

#### **Hardware**

The hardware requirements are:

- Any hardware platform that supports Java with minimum 128 MB space
- RAM (256 recommended).
- Make sure that the temporary directory set by the Environment Variable TMP has a minimum disk space of 100 MB.

#### **Operating System**

You can install Server on any platform that supports JDK 1.3.1\_01. It has been tested on:

- Windows 2000™ Professional and Server
- Windows NT™ 4 with Service Pack 5 or higher versions
- RedHat Linux™ 6.2
- Sun Solaris™ 7 and 8

#### **JDK**

The Server requires JDK™ 1.3.1\_01 to install. To verify the JDK version on your machine, enter `java -version` at the command prompt. The value returned is the JDK version on your machine.

## JDBC

The Server supports JDBC 2.0 Standard Extension drivers. These drivers must be present in the directory `jdk1.3.1_01/jre/lib/ext`, where `jdk1.3.1_01` is the JDK installation directory on your machine. A complete list of JDBC drivers is available in <http://industry.java.sun.com/products/jdbc/drivers>.

### Database requirements

Data resource connections can be made with any RDBMS that has a JDBC 2.0 Standard Extension driver. The Server has been tested for the following database servers and drivers:

- Oracle 8i
- Oracle Thin Driver Version 8.1.6.0.1
- Merant DataDirect Oracle Driver 2.2
- Informix 7.30 TC3
- Informix JDBC Driver for Informix Dynamic Server 2.10.JC1N361
- Merant DataDirect Informix Driver 2.2
- SQL Server 7.0
- Merant DataDirect SQL Server Driver 2.2
- JTurbo SQL Server Driver 2.0
- i-net Sprinta 4.15 SQL Server Driver
- Cloudscape 4.0
- Cloudscape 4.0 JDBC Driver 4.0
- Cloudscape 4.0 RMI JDBC Driver 4.0

For a complete Platform Certification Report, refer the release notes for the Server.

## Installation

The following parameters are required to configure the server:

- 1 JDK Home
- 2 Installation Directory
- 3 Licensee Name
- 4 License Key

Create an <XML-FILE> file with the following code:

```
<AutomatedInstallation langpack="eng">
<com.pramati.CDKeyPanelAction>
<name><LICENSEE_NAME></name>
<value><LICENSEE_KEY></value>
</com.pramati.CDKeyPanelAction>
<com.pramati.TargetPanelAction>
<installpath><INSTALLATION_DIRECTORY></installpath>
<javaHome><JDK_HOME></javaHome>
</com.pramati.TargetPanelAction>
</AutomatedInstallation>
```

After generating the XML file, run the following command to install the Server.

```
java -jar Pserver30.jar -silent <XML-FILE>
```

The Server is installed in the installation directory specified in the <XML-FILE>.

### Providing the license key

If the ISV specific licensing is used, then ISV must build a mechanism to embed the Pramati's key into the ISV key. At install time, the installation program must extract the Pramati license key and include it in the XML as described in the previous section.

The Pramati Server reads the license key from the license.key file in the Server installation directory. On startup, the Server picks the license from this file and deletes the file.



# 7

## *Setting up Java Client*

---

If the ISV has a J2EE application with Java client, it needs to be bundled, so that his clients can install either the application's Server module or Client module.

### **Requirement at client machines**

The client side bundle consists of:

- 1 Pramati Server's client side jars
- 2 ISV application's Pramati specific client side jars, the stubs generated at the time of deployment under the archives
- 3 ISV application's application specific client side jars.
- 4 Common specifications and API classes including `ejb.jar`)

These are located in `[server_install]\server\nodes\StandAlone\default\archives\<application>\classes`

### **Generating installable archive**

The distributable bundle is generated during the archive preparation stage. Here, the deployment tool generates the required client side jars and places them in the specified directory. The ISV can then add the client side JARs to this directory to generate the installable archive.



# 8

## *Customizing the Console*

---

Pramati Server Console is composed of JSP pages called View Objects and JMX MBeans called Control objects. You can view these objects under Catalog in the Management Console.

From the Administration Service Page, you can connect to any Server instance, that is running and handles Server resources. After connecting to the Server, you can view the Server configuration details in the Management Console.

Administrators can use the Console to deploy resources and extract maximum performance on any deployment scenario. The console can be configured to communicate alerts and status information to WAP-enabled devices.

### **Levels of customization**

The console can be customized at various levels:

- Creating a custom view by selecting the pages that need to appear as hot links from the default view. Any page in the list, with specific input parameters, can be selected.
- Customizing the content/look-and-feel of any given page in a view.
- Identifying the relevant pages from a view and embedding them wherever required.

### **Creating Custom Views**

Custom admin-console (views) can be created and shipped to the customer along with the ISV's product bundle. A customized view allows for selecting hot links from an exhaustive list of pages, and by sub setting the chosen alerts.

Hotlinks allow direct navigation to the pages without getting lost while navigating to reach the desired page. A page to be hot linked can be selected from the exhaustive list of pages available from navigating to the catalog.

Any page viewed can be added as a hotlink, with a name, to the list. This option is available when any view other than "Default" is selected.

For example, a page showing the statistics of a particular data source named "Bank" can be configured as a hot link in the view. Then, the statistics for the bank resource can be viewed by clicking on the link directly at runtime.

To create a custom view:

- 1 To add a new view or modify the existing one, click on '+' or 'Edit' button displayed under View on the Console main page.
- 2 Enter the name of the view and description. Select a template for the view and click Save.
- 3 This lists a set of the view elements - Links, alerts and documents. Select the pages required to form the desired view.
- 4 The view can be created by selecting default JSP pages or alerts and configuring them according to the application standards or the existing pages may be selected, but the attribute names are modified to suit the requirement. Click on Add to add them to the view. This shows the view in the list of views in the main page.

#### **Distributing the view at the installation site**

Details of any view created/modified are persisted immediately. The views created locally can be recreated on a remote site by copying the persisted files to the remote site.

A file named "Views.xml" is updated for each view and a new file named "<view\_name>\_jspconfig.xml" would be created. These files are located at "<install-dir>/server/nodes/<StandAlone/Cluster>/<server-name>/archives/public\_html/admin/config" directory. If these files are copied to

the target destination the view created locally would reappear on the remote site.

"Views.xml" contains information on the hotlinks chosen, inputs for each of the hotlinks and alerts subclassed. Information contained in "<view\_name>\_jspconfig.xml" is explained in the next section.

## Customizing pages

Console pages are developed using intelligent custom JSP tags. Customizing is limited to the view selected. The various types are:

- Hiding/unhiding the components appearing in a page.
- Changing the display names of table headers and other labels appearing in the page.
- Providing default values for the components by accepting inputs appearing on the page.

To customize a page:

- 1 Click on the 'Edit' option next to the view displayed in the Console main page.
- 2 This lists a set of the view elements - Links, alerts and documents. The pages under these are marked and selecting the '+' button helps to view the attributes of the pages. The desired page can be clicked to set the attributes of every page.

### Distributing the customisation at a target site

The customisation information of pages in a view is stored in the file "<view\_name>\_jspconfig.xml" at "<install-dir> /server/nodes/<StandAlone/Cluster>/<server-name>/archives/public\_html/admin/config" directory. By copying this file to the target location the customisation can be recreated. It is required that the Views.xml is copied along with this file.

### Embedding Console pages in an application

The list of pages appearing in the Console are listed in the Page Catalog along with their URLs and descriptions. Pramati's Admin Console has a separate page for every logical component.

A page viewed on the browser can be made of three or four "sub" pages. The details of these sub pages are also listed in the catalog. The URLs for the pages carry information about the input parameters required.

By retyping this URL in a browser and supplying the input parameters, the IP Port name and the context name ("admin") the page can be viewed wherever the Pramati Server is running. These URLs can be a part of other pages specific to the ISV.

To view a page, the URLs require the input parameter. This selects the identified viewname for the session. If no values are given, the last view selected would be chosen.

Select the 'View' option under Catalog in the Console main page. This gives the list of views available.

- 1 Click on 'View Objects' tab to get a list of pages and their URLs.
- 2 Select the page desired and its corresponding URL.
- 3 Paste the URL preceded by `http://<host name>:<port name>`.
- 4 Example: The following URL displays the web container details:

```
/server/webc/edit.jsp?showsiblings=true
```

This is specified as : `http://localhost:8181//server/webc/edit.jsp?showsiblings=true`

# 9

## *Creating and Configuring Pramati Server Instances Using XML*

### **Introduction**

Pramati Server instances can be configured manually by modifying the configuration XML files. Server instances of type J2EE, Message Server and Cluster can be manually created with simple configuration in the XML files. This section describes the structure of a Server instance and discusses the procedure to create different types of Server instances in Standalone and Cluster scenarios.

Topics in this section are divided into:

- 1 Overview of Pramati Server instances and Pramati Server Cluster
- 2 Node directory structure and configuration with reference to the XML files.
- 3 Creating Server instances and Clusters.
- 4 Pramati Server architecture and the functional blocks of the Pramati Services Framework.

### **Understanding Pramati Server instances**

An instance of the application server, which some vendor term 'server node', can serve either as a Standalone server or a cluster server node. Pramati Server instance can either be a Standalone Server or Server Cluster Node.

Cluster is a logical grouping of two or more server instances running on a single machine or multiple machines and functioning as a single server. All servers in a cluster are transparent to the user. Pramati Server leverages this capability to provide scalability and high availability to applications with the ability to handle high workloads and reduce application downtimes due to failover.

When the server starts, an instance of the server is spawned as a separate process with its own JVM. Attributes of the instance—sizing, runtime behavior and performance—is defined by the configuration of the server and is specified in a set of server configuration files (XMLs).

## Key terminology

The following table provides a brief description for the various terms used.

Table 1: Terminology

Term	Description
Installation	Pramati Server product Installation
Install-Dir	The directory under which Pramati Server is installed
Nodes Directory	The directory under which all configured server instances exist
Server	The installed server with default settings
Server Instance	An instance of a Pramati Server- of any type (J2EE, Web cluster node, EJB cluster node, Message Server) will normally run in one VM.
Standalone Server	An instance that is a self contained J2EE Server running in a single VM
Instance Directory	The directory under the <i>Nodes Directory</i> that holds the configuration and other files/information for a Server Instance
Message Server	An instance which functions as a Standalone Message Server
Embedded Message Server	Full featured Message Server that runs <i>inside</i> a Standalone Server instance
JMS Adapter	JMS Adapter is setup in a J2EE Server instance determining the connectivity to a Message Server (Embedded or external Standalone Message Server or HA-Message Server)
Cluster	A logical server comprising a group of underlying nodes forming a single entity that internally works with a collection of nodes.
Cluster Node	Server instance that serves as a node of a cluster
J2EE Cluster	A cluster with both EJB and WEB Nodes
J2EE Node	A Server instance that serves as a node serving both Web and EJB requests. Can exist as a part of a J2EE Cluster
BOTH Node	Same as J2EE Node
Web Cluster	A cluster of Web Nodes
Web Node	A Server instance that serves as a node serving Web requests alone. Can either exist as part of a J2EE Cluster or a Web cluster
EJB Cluster	A cluster of EJB Nodes

Table 1: Terminology

Term	Description
EJB node	Server instance that serves as a node serving EJB requests alone. Can either exist as part of a J2EE Cluster or a Web cluster
Message Server Cluster	A cluster of Message Server Nodes
HA Message Server	Message Server Cluster that only supports failover, and not Load Balancing (as of 3.5, Message Server Cluster is only HA Message Server)
Message Server Node	A Server instance that serves as a node serving Message Server requests alone
Load Balancer	A request dispatcher for managing requests
Configuration	Modifying the default settings for the server

## Pramati Server configuration files

Configuration files required for setting up Pramati Server are located in the /templates directory under the installation root. These files, with their descriptions, are listed in the following table.

Table 2: Pramati Server configuration files

Configuration File	Description
server-config.xml	The main configuration file where all the details related to services bootstrapping is stored. When you create a node manually, make sure that you copy this file in the config directory from the template.
resource-config.xml	All the resource related information like Connection Factory, Data Source properties, Data Source Cluster, Mail Resource and Message Server Resource are stored here.
web-config.xml	Necessary configuration information for the Web Container Service is stored here.
deploy-config.xml	Contains configuration information for the deploy service.
logging-config.xml	Logging parameters like rotation type, max file size and rotation time are configured here.
security-config.xml	Contains information about security providers, security policy, certification manager and realm.
web-cache.xml	Contains dynamic cache configuration. Dynamic Cache service boosts the network throughput and access speed in the web container of Pramati Server.

Table 2: Pramati Server configuration files

Configuration File	Description
cluster-config.xml	Stores cluster related information including configuration details for individual nodes.
jms-config.xml	Information related to Message Server Queues and Topics are stored here.

## Types of Server instances that can be configured

There are three basic types of Pramati Server configurations:

- J2EE (serves EJB and web applications)
- Message Server (serves pure Message Server applications)
- Load Balancer (acts as the Pramati Request Dispatcher)

These are set up in the configuration XML, `server-config.xml`. The default instance is J2EE. In a J2EE configuration, the Server can be set up as a Standalone Server or a Server Cluster Node. In a Message Server configuration, the Server can be set up as a Standalone Server or a High Availability Node.

**Note:** In the J2EE configuration, the Server can run Message Server in an embedded mode.

From these three basic configuration types, the following Server instances can be set up:

### Standalone Server configurations derived from an instance

- Standalone Server
- Standalone Server with embedded Message Server
- Standalone Message Server

The most basic type of J2EE Server instance is the Standalone Server. It exists on a single machine, is not part of any cluster, and acts as an independent server. The Standalone Server is the default server type and can be configured to run embedded Message Server.

**Note:** By default embedded Message Server is checked as enabled in a J2EE server instance.

#### Server Cluster Node configurations derived from an instance

- J2EE Node (termed BOTH Node)
- Web Node
- HA Message Server Node
- EJB Node
- Load Balancer Node

The Server Cluster J2EE Node has both Web and EJB services enabled in `server-config.xml` file. A node can be configured to as a Standalone Message Server by disabling other services like EJB and Web—as long as there are no inter-service dependencies. The type of server instance is determined based on the specific services that are enabled through the Pramati Services Framework. Details are discussed in the following sections.

### Structure of Nodes Directory

Creating a Server instance (in the node directory) makes a directory each for Standalone Server and Server Cluster Node.

Each configured server instance, has an unique name by which it is identified in a cluster. Instances can be created and configured manually without the aid of the Console or Shell.

While creating server instances (in the node directory), there will be a DIR for each server instance (in case of Standalone servers) and one for each cluster-node in case of Cluster. The type of server instance is determined based on the specific services enabled. Details are discussed in the following sections.

#### Install Directory Structure

*/server*

*-bin*

(Server startup scripts are stored here)

*-lib*

(External and internal libraries used by the server are stored here)

*-templates*

(This directory contains various configuration XMLs and their DTDs along with the cloudscape DB property file)

*-nodes*

(Server Instances Directory containing node specific configuration information)

### Server Instance Directory Structure

*-<node-name>*

(A unique name which identifies the node in a cluster)

*-config*

(List of all configuration XML files which influences the runtime behavior of the node.)

*-logs*

(Stack traces and log messages for the node are stored here)

*-archives*

(This directory holds all extracted application

archives

*-temp*

(A temporary directory)

## Procedure for creating Pramati Server Instances

Server instances are created in the 'nodes' directory of a Pramati Server installation. There will be one DIR for each server instance (in case of Standalone servers) and one for each cluster-node in case of Cluster. The type of server instance is determined based on the specific services enabled. Details are discussed in the following sections.

The common steps involved in configuring a Server instance, of any type, are:

- 1 Under the nodes directory, create a sub-directory (henceforth referred to as *instance-directory*) for the required instance.
- 2 Name of the sub directory will be the name of the instance.

- 3 Create the following sub directory under this instance's directory
  - *config*
  - the other directories described in the 'Structure of Nodes Directory' above will be automatically created after the instance starts up.
- 4 Depending on the type of instance needed, copy the required config files from the <install\_dir>/server/templates to the configuration directory created above. The specific files needed are listed in the Table 3 (*Configuration files needed for Server Instances*) provided in the following section.
- 5 Edit the required configuration XML to setup the information as described in the following sections, relevant to the type of Instance being setup.
- 6 The instance can be started from the command line using the *-node* option, after running setup.bat /sh:

```
com.pramati.Server -node <node name>
```

#### *Configuration files:*

The configuration files determine the type of Server instance and the behavior of the server instance. For each server instance type a specific set of configuration files are required. The table below lists the config files required for each instance type. These will need to be copied from templates directory into the <instance>/config directory.

## Configuration File Dependency

Table 3: Configuration files needed for Server Instances

Configuration File	Standalone Server	J2EE (BOTH) Cluster Node	Web Cluster Node	Standalone Message Server	HA Message Server Node	EJB Cluster Node	Load Balancer
server-config.xml	Required	Required	Required	Required	Required	Required	Required
resource-config.xml	Required	Required	Can Have	Required	Required	Can Have	Required
web-config.xml	Required	Required	Required	-	-	-	Required
deploy-config.xml	Required	Required	Required	-	-	Required	Required

Table 3: Configuration files needed for Server Instances

Configuration File	Standalone Server	J2EE (BOTH) Cluster Node	Web Cluster Node	Standalone Message Server	HA Message Server Node	EJB Cluster Node	Load Balancer
logging-config.xml	Can Have	Can Have	Can Have	Can Have	Can Have	Can Have	Can Have
security-config.xml	Required	Required	Required	Required	Required	Required	Required
system-security.xml	Required	Required	Required	Required	Required	Required	Required
web-cache.xml	Required	Required	Required	-	-	-	Required
cluster-config.xml	-	Required	Required	-	Required	Required	Needed, if HA
jms-config.xml	Needed for embedded Message Server	-	-	Required	Required	-	Required
web-lbconfig.xml	-	-	-	-	-	-	Required

*Note: In most cases, the defaults used in the configuration file templates will be sufficient to start an instance.*

## Using the NodeCreator

The above mentioned XML files need not be created manually. Pramati provides an utility for creating these XML files. The NodeCreator is a utility for creating or removing nodes and clusters. You can configure and use the NodeCreator with both the command line and XMLs.

### Using the NodeCreator with the Command line

To run the NodeCreator:

Run NodeCreator.bat (for Windows) or NodeCreator.sh (for Unix) located at <install\_dir>/server/bin/.

Execute `java com.pramati.NodeCreator` class by passing the following values:

Table 4: Operations that can be performed using the Node Creator class

Operation	Description
-createj2ee	To create a Standalone J2EE node
-createj2eecluster	To create multiple J2EE nodes in a cluster
-createjms	To create a Message Server node
-createjmscluster	To create multiple Message Server nodes in a cluster

The various values (given as {parametername value} format) required to perform the above operations are:

Table 5: Values for the Node Creator class

Value	Description
-name	Name(s) of the nodes. In case of a cluster, the names are separated by a comma ','
-ip	The IP of the machine on which this node is being created. The default value is local host. In case of a cluster, the IPs are separated by a comma ','
-namingport	Naming port(s) of the nodes. In case of a cluster, the ports are separated by a comma ','
-httpport	HTTP port(s) of the nodes. In case of a cluster, the ports are separated by a comma ','
-clustername	In case of a cluster, the name of the cluster.

You can also specify the System Property 'install.root', if the program is not being run from the Server root.

#### Example for creating a J2EE Standalone node

```
java -Dinstall.root=. com.pramati.NodeCreator -createj2ee -
name my
node1 -namingport 9191 -httpport 8181
```

#### Example for creating a J2EE cluster

```
java -Dinstall.root=. com.pramati.NodeCreator -
createj2eecluster -name my
node1,mynode2 -namingport 9191,9292 -httpport 8181,8282 -
clustername mycluster
```

The help snippet that shows up on the command shell is as follows:

```
Usage: java com.pramati.NodeCreator [operation] [option value]
[flag]
```

### Using the NodeCreator with XMLs

You can create a node or cluster using the `config-topology-standalone-template.xml` and `config-topology-cluster-template.xml` located at `<install_dir>/server/templates/`.

Using these XML templates, you can convert existing node types, create nodes for a cluster, and configure persistence properties for the database, EJB or Web.

Converting a node type is useful if you want to convert an existing, say Standalone J2EE Server to an EJB cluster type. To do this, open the `config-topology-cluster-template.xml`:

```
<configuration-topology>
  <!-- Valid Types - a J2EE (EJB/BOTH/WEB) Cluster or a JMS
  Cluster-->
  <!-- SAMPLE CONFIGURATION, change node definitions as
  required -->
  <cluster name="MyCluster" type="J2EE">
    <convert-nodes>
<!-- These are nodes for conversion, the server "J2EESAS1" of
the given ip in the <source-node> tag would be converted to
the cluster node type given in the node definition. The tar-
get nodenames can be different in which case the directories
will be copied.-->
      <node name="EJB1" type="EJB1-cluster" naming-
      port="9000" http-port="8000">
        <source-node ip="localhost" name="J2EESAS1" />
      </node>
    </convert-nodes>
  </cluster>
</configuration-topology>
```

Table 6: Description of attributes for `<convert-nodes>` tag

Attributes	Description
node name	Name(s) of the target node. That is the node that is to be created.
type	The type of the target node that is to be created.
naming-port	Naming port(s) of the target node.
http-port	HTTP port(s) of the target node.
source-node ip	The IP of the source machine on which this node is started. The default value is local host.
name	Name of the source node.

If you wish to create a J2EE Cluster, locate the `<create-nodes>` tag in the `config-topology-cluster-template.xml`:

```
<create-nodes>
  <node name="nodename1" type="both">
    <ip>localhost</ip>
    <naming-port>9292</naming-port>
    <http-port> 8282</http-port>
  </node>
  <node name="nodename2" type="both">
    <ip>localhost</ip>
    <naming-port>9293</naming-port>
    <http-port> 8283</http-port>
  </node>
</create-nodes>
```

Table 7: Description of attributes for `<create-nodes>` tag to create a cluster

Attributes	Description
node name	Name of the cluster node.
type	The type of the cluster - EJB, Web, or BOTH.
ip	The IP of the machine on which this node is being created. The default value is local host.
naming-port	Naming port of the node.
http-port	HTTP port of the node.

To define the properties for persistence, modify the `<configuration-persistence>` tag in the `config-topology-cluster-template.xml`:

```
<!-- use "db" for DB persistence -->
<configuration-persistence type="file">
  <property name="driver" value=""/>
  <property name="url" value=""/>
  <property name="user" value=""/>
  <property name="password" value=""/>
  <property name="tablename" value=""/>
</configuration-persistence>
<!--the default type here is "in-memory", use "db" for DB
persistence -->
<ejb-cluster-persistence >
```

```

    <property name="driver" value="" />
    <property name="url" value="" />
    <property name="user" value="" />
    <property name="password" value="" />
    <property name="table-name" value="" />
  </ejb-cluster-persistence>
  <!--the default type here is "in-memory", use "db" for DB
persistence -->
  <web-cluster-persistence >
    <property name="url" value="" />
    <property name="url" value="" />
    <property name="user" value="" />
    <property name="password" value="" />
    <property name="table-name" value="" />
  </web-cluster-persistence>
  <!-- the upload-config allows pre-configuring the created
cluster.e.g.The configuration for nodename1 would be uploaded
to the DB and this configuration would be synced up to the
other nodes.-->
  <upload-config-source-node name="nodename1" />
</cluster>
</configuration-topology>

```

## Custom node configuration

You can use the `-configuration` flag with the `nodecreator` utility and pass the name of a configuration file as a parameter. For example:

```
com.pramati.NodeCreator -configuration config.xml
```

The `config.xml` file can have the following structure:

```

<server-configuration>
  <configuration service-name="ResourceService">
    <datasource name="myDS" connection-factory="myCF" transac-
tion-participation="true" description="No Description">
      <pool-properties min-pool-size="10" max-pool-size="20"
initial-pool-size="2" refresh-interval-seconds="" con-
nection-request-timeout-seconds="" idle-timeout-sec-
onds="" cache-size="" />
      <connection-validation sql="" class="" />
    </datasource>
  </configuration>
</server-configuration>

```

```

<connection-factory name="myCF" description="" class-
name="COM.cloudscape.core.RmiJdbcDriver" url="jdbc:cloud-
scape:rmi:D:\CTS\j2eects\db_cs40\DB1" authorized-
by="Container">
  <login-parameters user="" password="" mask-pass-
word="false"/>
</connection-factory>

<-jms-adapter name="myAdapter" description="No Description"
interface-class="com.pramati.jms.client.JMSProviderImpl">
  <properties>
    <property name="com.pramati.naming.cacheLookups"
value="false"/>
    <property name="java.naming.factory.initial"
value="com.pramati.naming.PramatiLocalContextFactory"/>
    <property name="java.naming.provider.url" value="rmi://
localhost:9090"/>
    <property name="com.pramati.force.standalone.ctx"
value="false"/>
  </properties>
</jms-adapter-->
</configuration>
</server-configuration>

```

As an example, the above will add the mentioned resources in the newly created nodes - the relevant entries will be added to the `resource-config.xml` files.

The supported resources for this tag are:

1. JDBC Connection Factories
2. Datasources
3. JMS Adapters

## Creating a Standalone Server

### *General Instructions:*

- Under the nodes directory, create a sub-directory for the required instance.
- Name of the sub-directory will be the name of the instance.
- Create the following sub-directory under this instance's directory

- *config*

- the other directories described in the '*Structure of Nodes Directory*' above will be automatically created after the instance starts up.

- Depending on the type of instance needed, copy the required config files from the INSTALL-DIR/server/templates to the configuration directory created above. The specific files needed are listed in Table 2 provided in the following section.
- Edit the configuration XML to setup the information as described in the following sections.
- Make sure that the cluster service is disabled in `server-config.xml`

```
<service name="ClusterService" enabled="false"
        class-name="com.pramati.cluster.PramatiClusterService">
```
- The instance can be started using *-node* option.

A Instance can be created by just creating a directory with an unique name under *server/nodes* directory. Once the directory is created, create another directory called *config* and copy `server-config.xml` and other configuration files required for the Standalone Server (as described in Table2) from the templates directory to *config*. Once the node directory is created, you can activate the Instance using *-node* command.

The same process can be followed for all the instance types including Message Server instances.

Once the Standalone J2EE Server is setup, based on the type selected, setup the basic attributes of the Web and EJB Containers. In the `server-config.xml`, change the values as needed:

```
<service name="WebContainer" enabled="true"
        class-name="com.pramati.web.WebServer">
    <config-file>web-config.xml</config-file>
    <requires always="NamingService,DeployService" />
    <property name="http-port" value="8181" />
    <property name="https-port" value="443" />
    <property name="ssl-enabled" value="false" />
</service>
```

Default value of HTTP port, and HTTPS port can be set now.

For EJB, in `server-config.xml`, the default pool size and session timeout value can be modified.

```

<service name="EJBContainer" enabled="true"
        class-name="com.pramati.ejb.EJBContainer">
    <requires always="NamingService,TransactionService,
        DeployService" />
    <property name="default-entity-min-pool-size"
value="40" />
    <property name="default-entity-max-pool-size"
value="1000" />
    <property name="default-session-min-pool-size"
value="40" />
    <property name="default-session-max-pool-size"
value="1000" />
    <property name="default-mdb-min-pool-size"
value="40" />
    <property name="default-mdb-max-pool-size"
value="1000" />
    <property name="default-low-activity-interval"
value="60" />
    <property name="default-session-timeout"
value="1000" />
    <property name="smart-code-generation"
value="true" />
</service>

```

## Enabling embedded Message Server in a Standalone Server

For enabling embedded Message Server for a Standalone Server follow the general instructions as given above for creating a Standalone Server. Add the following:

```

<service name="JMSService" enabled="true"
        class-name="com.pramati.jms.server.PramatiMessageService">
    <config-file>jms-config.xml</config-file>
    <requires always="NamingService,ResourceService"
        if-enabled="TransactionService" />
</service>

```

If it is not already present in your `server-config.xml` file. By default the standalone server is configured to start embedded Message Server service on services framework startup. '`enabled=true`' lets the server start the Message Server Service. For an embedded Message Server, both EJB and Web container can be enabled in `server-config.xml`.

```
<service name="WebContainer" enabled="true"
    class-name="com.pramati.web.WebServer">
    service name="EJBContainer" enabled="true"
    class-name="com.pramati.ejb.EJBContainer">
```

## Restricting services in a Standalone J2EE Server

Based on the Pramati services framework, the services can be disabled or enabled. The services include the core containers (Web and EJB) and other services such as Resource and Transactions. A Standalone J2EE Server by default has both Web and EJB Container services enabled:

```
<service name="WebContainer" enabled="true"
    class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="true"
    class-name="com.pramati.ejb.EJBContainer">
```

For a Web only J2EE Server, enable the Web Container Service and disable the EJB Service.

```
<service name="WebContainer" enabled="true"
    class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="false"
    class-name="com.pramati.ejb.EJBContainer">
```

For an EJB only J2EE Server, enable the Web Container Service and disable the EJB and Message Server Services.

```
<service name="WebContainer" enabled="false"
    class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="true"
    class-name="com.pramati.ejb.EJBContainer">
```

## Creating Standalone Message Server

For a Standalone Message Server, enable the JMS Service in the `server-config.xml`:

```
<service name="JMSService" enabled="true"
```

```

        class-name="com.pramati.jms.server.PramatiMessageService">
        <config-file>jms-config.xml</config-file>
        <requires always="NamingService,ResourceService"
        if-enabled="TransactionService" />
    </service>

```

Disable the Web Container and EJB Container services from `server-config.xml`:

```

    <service name="WebContainer" enabled="false"
        class-name="com.pramati.web.WebServer">
    <service name="EJBContainer" enabled="false"
        class-name="com.pramati.ejb.EJBContainer">

```

## Understanding Pramati Cluster Configuration

Pramati Server features a powerful clustering solution. A cluster is defined as a group of nodes. Pramati Clustering solution offers transparent, self-organizing, homogenous web based configuration and management allowing easy EJB and Web clustering. When an application is deployed on any one of the cluster nodes, it is automatically replicated across all running cluster nodes. For a client, this translates to a single server view of the cluster.

Server clustering is self-organizing. Adding or removing nodes does not require restarting of the cluster. When a node is added, the load is automatically distributed across all the nodes based on their weight.

All the J2EE Services like Naming, Resource, Security and Transaction are automatically replicated across all the nodes across the cluster. Non-J2EE services like locking, application synchronization, load balancing, and failover are designed such that all complex cluster mechanics are hidden ensuring a single server view of the client.

Server clustering provides both EJB and Web clustering. Each node has its own set of replicated and non-replicated services. Pramati Cluster supports the Cluster-node types: Web, EJB and BOTH.

Types of Clusters supported are:

- J2EE Clustering
- Message Server Clustering

## Cluster Definition

Cluster definition involves:

- Defining a logical cluster (this information is available with all the nodes of the cluster)
- Defining the nodes of the cluster

The initial definition of the cluster is made in the XML of the first node of the cluster. Once the basic cluster attributes are defined in the `cluster-config.xml`.

The cluster type is defined by the Cluster-node definition in the `cluster-config.xml`

```
<node>
    <name>node</name>
    <computer-name>127.0.0.1</computer-name>
    <naming-port>9191</naming-port>
    <http-port>8181</http-port>
    <node-type>BOTH</node-type>
    <node-id>001</node-id>
</node>
```

Here, when node-type is J2EE, enabling the cluster service would make it a J2EE Cluster. A J2EE Cluster can be of type BOTH (both ejb and web) or just EJB/WEB.

A J2EE Cluster cannot start with Message Server in an embedded mode and a Message Server cluster cannot have EJB or Web containers enabled. If node-type is "JMS", then it becomes a HA Message Server cluster.

Once cluster is enabled in `server-config.xml`, the cluster topology and properties are defined in the `cluster-config.xml`. Cluster nodes, topology, persistence (EJB/Web) and loadbalancer will be specified here.

J2EE clustering can involve both EJB and Web clustering. However Web and Message Server instances cannot exist in a single cluster. The Cluster solution is based on a common Cluster backbone (called Cluster service, which also runs on the Pramati Services framework).

Setting up a cluster involves a cluster definition and defining cluster nodes. The cluster definition essentially comprises defining the common configuration and state persistence information. Pramati Cluster being a self-organizing non master-slave homogenous cluster, does not have a separate entity for cluster

housekeeping or processing. Based on the shared persistence store, the cluster runtime (cluster service) in each cluster node jointly perform all the housekeeping and cluster operational tasks.

### Uploading the configuration to the DB (When DB is used for configuration persistence)

In case of db persistence (for cluster configuration), after initial setup of cluster node, to upload the configuration into the database, start the node for the first time with the following commandline option:

```
com.pramati.Server -node <NODENAME> -uploadconfig
```

The rest of the nodes would automatically pick up the configuration and do not need to be started with this option. After any change in the configuration, to update the database, start the cluster node with the following startup option:

```
com.pramati.Server -node <NODENAME> -updatelocalconfig
```

This option would update all of the configuration files. Only this node needs this option and the rest would automatically use updated configuration. To update specific configuration changed manually, start cluster node with the following startup option:

```
com.pramati.Server -node <NODENAME> -updatelocalconfigfor  
<SERVICENAME>[,<SERVICENAME>]
```

Only this node needs this option and the rest would automatically use updated configuration.

### Creating a Cluster Instance (steps common to all types of clusters)

The cluster definition is essentially 'defined' along with the first cluster node, and the same common persistence information and cluster details (including name) is to be setup in all the other nodes that will form a part of this cluster.

The steps involved in creating a cluster are:

- The DIR and the config files have to be setup just as one would do for a Standalone instance
- Define the name of the cluster

Open the `server-config.xml` file. Add a declaration as follows:

```
<belongs-to-cluster> MyCluster </belongs-to-cluster>
```

- Enable the cluster service

This cluster name is a logical representation for a group of nodes enabling them to communicate with each other. Though at present the nodes interact with each other through a repository of IP and port information, it is important that the nodes identify themselves with a common name. Once the node is identified as part of a cluster, the service definition for the cluster can be added in the `server-config.xml` file as follows:

```
<service name="ClusterService" enabled="true"
  class-name="com.pramati.cluster.PramatiClusterService">
  <config-file>cluster-config.xml</config-file>
  <requires always="NamingService" />
</service>
```

Set the 'enabled' attribute to true, so the cluster service is started when the services framework is started. As you can infer from the above code snippet, the actual configuration file for the cluster is stored by every node in `cluster-config.xml` file in the config directory.

- Setup persistence

The services configuration in a node can be stored **either in a file or a database**. These information can be specified in the `server-config.xml` file's `<configuration-persistence>` option. By default all the configuration information are persisted to the file.

```
<configuration-persistence use="file">
  <persistence type="file" />
  <persistence type="db">
    <property name="driver" value="" />
    <property name="url" value="" />
    <property name="user" value="" />
    <property name="password" value="" />
    <property name="tablename" value="" />
  </persistence>
</configuration-persistence>
```

Persistence can be configured either at the file level or DB level. If you are planning to have a DB level persistence, connection parameters like driver, connection URL, username, password and table name can be specified in the configuration file which are persisted. A driver can be any valid JDBC driver. The driver varies for different databases. The 'tablename' attribute is used to create a specific table where the configuration details for all the services are stored.

If you are setting up a new cluster, you can opt for a file level persistence, where the DB is not used for aiding a sync up. The new version of the server supports an 'On Demand sync up', wherein the the configuration information for each node are synced up with every other node and persisted in the DB.

- Setup the cluster configuration file that would define all nodes that form part of this cluster. `cluster-config.xml` file captures the information on the nodes that forms a cluster. Initially when creating the cluster this information is to be provided in this file. This file will be identical in ALL the nodes that will form the cluster- with information on each one of them.
- After the first startup of the server, the information gets persisted in the DB/ File persistence store of the cluster, and a read-only version of the file is always available (synced up from the DB on every 'start' of the cluster node). Copy the `cluster-config.xml` file from the templates directory to the `nodes/<node-name>/config` directory. Now add the node related information or topology for each node.

```
<node>
    <name>N1</name>
    <computer-name>128.128.77.77</computer-name>
    <naming-port>9191</naming-port>
    <http-port>8181</http-port>
    <node-type>BOTH</node-type>
    <node-id>001</node-id>
</node>
<node>
    <name> N2 </name>
    <computer-name> 128.128.77.78 </computer-name>
    <naming-port> 9190 </naming-port>
    <http-port> 8180 </http-port>
    <node-type> BOTH </node-type>
    <node-id> 002 </node-id>
</node>
```

## Setting up a J2EE Cluster

Follow the General instruction for setting up a Cluster node ( Refer to '*Creating a Cluster Instance—common to all cluster types*' ), summarized here:

- Create the Node Directory and copy the configuration files required for the J2EE Node as mentioned in Table 2.

- Add cluster declaration in `server-config.xml`. Make sure that EJB and Web or EJB/Web are enabled and Message Server is disabled.

```
<service name="WebContainer" enabled="true"
  class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="true"
  class-name="com.pramati.ejb.EJBContainer">
<service name="JMSService" enabled="false"
  class-name="com.pramati.jms.server.PramatiMessageService">
```

- Enable Cluster Service.

```
<service name="ClusterService" enabled="true"
  class-name="com.pramati.cluster.PramatiClusterService">
```

- Setup Persistence.
- Setup Cluster topology in `cluster-config.xml`.

*Note: Make sure the node types mentioned in the node definitions in `cluster-config.xml` are of the type BOTH, Web or EJB*

### J2EE Cluster node types

The Cluster node types for both the nodes are indicated as BOTH, which means both the Web and EJB Services are running on that node. If you specify BOTH, it is required that you enable both EJB and Web services from the `server-config.xml` file. Other valid options are EJB and Web.

- *BOTH* – If a cluster node is defined as BOTH, both EJB and Web containers are started. Select this option when the application has both enterprise and web components.

A snippet from `server-config.xml`:

```
<service name="WebContainer" enabled="true"
  class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="true"
  class-name="com.pramati.ejb.EJBContainer">
```

- *Web* – If a cluster node is defined as Web, only the Web container is started. EJB components are not available on this node. Select this option when the application has only JSP pages and other web components. A Standalone web node can act as an EJB Load Balancer.

A snippet from `server-config.xml`:

```
<service name="WebContainer" enabled="true"
```

```

        class-name="com.pramati.web.WebServer">
    <service name="EJBContainer" enabled="false"
        class-name="com.pramati.ejb.EJBContainer">

```

- **EJB** – If a cluster node is defined as EJB, only the EJB container is started. Web components are not available on this node. Select this option when the application consists of only enterprise beans.

A snippet from `server-config.xml`:

```

    <service name="WebContainer" enabled="false"
        class-name="com.pramati.web.WebServer">
    <service name="EJBContainer" enabled="true"
        class-name="com.pramati.ejb.EJBContainer">

```

To summarize:

Table 8: J2EE Cluster Node Type and required services

J2EE Node Type	Web Container	EJB Container
BOTH	Required	Required
EJB	-	Required
Web	Required	-

### Advanced Configuration

Typically a J2EE Cluster node means a 'BOTH' node. But a J2EE Cluster can also be configured as a Web only cluster, EJB only cluster or a combination of Web and EJB nodes.

### Setting up Web Replication and Persistence

You can set up a Web cluster following the same procedure as shown in '*Creating a Cluster Instance-common to all cluster types*'. Make sure that you enable the Web container service for all the nodes. This can be done from the individual `server-config.xml` files.

Additionally, you need to add these information in your `cluster_config.xml` file.

```

    <web-cluster>
        <enable>true</enable>
        <web-session-persistence>
            in-memory
        </web-session-persistence>
        <replication-on-all-nodes>

```

```

        true
    </replication-on-all-nodes>
    <web-cluster-persist-info>
        <url />
        <driver />
        <user />
        <password />
        <table-name />
    </web-cluster-persist-info>
</web-cluster>

```

Backup nodes can be setup for a Web cluster only when the web session persistence is set to In-Memory and replication on all nodes is set to false. Enable the Web cluster by setting '*enabled*' as '*true*'. There can be two kinds of Web session persistence:

- In-memory Persistence
- Database Persistence

When you select your Web session persistence to be in-memory, the HTTP session objects are persisted in memory locally for nodes in the same VM and persisted through RMI calls for nodes in different VM.

Whenever there is any state change for Web sessions, the information is replicated across all other nodes. This may may prove expensive on memory and hence it is better to set '*replication-on-all-nodes*' to '*false*'. When the '*replication-on-all-nodes*' are set to '*false*', only the backup nodes are notified for replication. The Web cluster persistence information can be stored in a DB.

### Setting up EJB Persistence

EJB session persistence for J2EE Cluster could be in-memory or database based. In case of database- based persistence, provide all the database information like URL, table name, etc. The following snippet is taken from

cluster\_config.xml:

```

<ejb-session-persistence>in-memory</ejb-session-persistence>
    <persist-info>
        <url />
        <driver />
        <user />
        <password />
        <table-name />
    </persist-info>

```

```
</persist-info>
```

The above example uses In-memory EJB session persistence.

## Setting up an HA Message Server Cluster

General instructions for setting up a cluster node (Refer to '*Creating a Cluster Instance*):

- 1 Create the Node Directory and copy the configuration files required for a Message Server node as mentioned in Table 2.
- 2 Add cluster declaration in `server-config.xml`. Make sure the JMS Service is also enabled and Web and EJB are disabled.

```
<service name="WebContainer" enabled="false"
  class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="false"
  class-name="com.pramati.ejb.EJBContainer">
<service name="JMSService" enabled="true"
  class-name="com.pramati.jms.server.PramatiMessageService">
```

- 3 Enable cluster service.

```
<service name="ClusterService" enabled="true"
  class-name="com.pramati.cluster.PramatiClusterService">
```

- 4 Setup persistence.

- 5 Setup cluster topology in `cluster-config.xml`.

**Note:** Make sure the node types mentioned in the node definitions in `cluster-config.xml` are only of the type `jms-cluster`.

A sample node configuration for a Message Server cluster:

```
<node>
  <name>node</name>
  <computer-name>jms1</computer-name>
  <naming-port>2099</naming-port>
  <node-type>jms-cluster</node-type>
  <node-id>001</node-id>
</node>
```

The above snippet is from `cluster-config.xml`.

**Note:** A Message Server Cluster cannot have a Web/EJB node. Hence make sure that the Web/EJB container services are disabled for all the Message Server nodes.

## Updating cluster configuration using XML (when DB is used for configuration)

Once cluster is enabled in `server-config.xml`, the cluster topology and properties are defined in the `cluster_config.xml`. Cluster nodes, topology, persistence (EJB/Web) and Load Balancer will be specified here. After the initial configuration of the cluster, the cluster configuration may be modified using the Console or the XMLs.

In a file based persistence, once the config XMLs are modified in any of the nodes, and the cluster is restarted, the new configuration takes effect and is synched-up with all other nodes.

When DB is used for config persistence, the following steps need to be followed to update the configuration in the DB, using local XMLs to provide the information:

- The cluster must be stopped
- In any of the nodes, update the required config XML
- Start the node, where the XMLs were modified:
  - After any change in the configuration, to update the database, start cluster node with the following option: `com.pramati.Server -node <NODENAME> -updateLocalconfig`  
This updates all of the configuration files.
  - To update specific configuration changed manually, start cluster node with the following startup option: `com.pramati.Server -node <NODENAME> -updateLocalconfigfor <SERVICENAME>[<SERVICENAME>]`

Only this node needs this option and the rest would automatically use updated configuration.

## Configuring JMS Adapters in J2EE Servers

JMS Adapters are setup in a J2EE Server instance determining the connectivity to a Message Server (Embedded, Standalone or HA-Message Server). JMS Adapters can be configured in `resource-config.xml` file of the J2EE Server (Standalone or cluster-node).

In case of non-embedded Message Server, that is when the JMS Service is not running in the same VM as the EJBServices, specify

```
com.pramati.naming.client.PramatiClientContextFactory for
java.naming.factory.initial. Also specify the IP and port of the VM where
```

the Message Server is running in `resource-config.xml`. In case of HA Message Server, provide the comma separated list of URLs of HA Message Server nodes.

Sample configuration for the local JMS Adapter is shown below. The sample configuration is valid only for a Message Server running in embedded mode:

```
<jms-adapters>
  <jms-adapter name="default" description="nodesc" inter-
    face-class="com.pramati.jms.client.JMSProviderImpl">
    <properties>
      <property name="java.naming.factory.initial"
        value="com.pramati.naming.PramatiLocalContextFac-
        tory"/>
      <property name="java.naming.provider.url"
        value="rmi://localhost:9191"/>
    </properties>
  </jms-adapter>
</jms-adapters>
```

Sample configuration for the external JMS Adapter (non-HA) is shown below. The sample configuration is valid only for a Message Server not running in embedded mode. `jmshost` is the host where the Standalone Message Server is running on Port 2099:

```
<jms-adapters>
  <jms-adapter name="default" description="nodesc" inter-
    face-class="com.pramati.jms.client.JMSProviderImpl">
    <properties>
      <property name="java.naming.factory.initial"
        value="com.pramati.naming.PramatiClientContextFac-
        tory"/>
      < property name="java.naming.provider.url" value="rmi://
        jmshost:2099"/>
    </properties>
  </jms-adapter>
</jms-adapters>
```

Sample configuration for the external JMS Adapter (HA) is shown below. The sample configuration is valid only for an HA-Message Server cluster. `jms2`, `jms1` are part of HA Message Server configuration.

```
<jms-adapters>
```

```

<jms-adapter name="default" description="nodesc" inter-
face-class="com.pramati.jms.client.JMSProviderImpl">
<properties>
  <property name="java.naming.factory.initial"
  value="com.pramati.naming.PramatiClientContextFac-
  tory"/>
  <property name="java.naming.provider.url"
  value="rmi://jms2:9191,rmi://jms1:9191"/>
</properties>
</jms-adapter>
</jms-adapters>

```

*Note: For HA Message Server node or a Standalone Message Server node, the resource-config.xml should not contain any JMS Adapters.*

## Web Load Balancer

A Load Balancer is a type of Server instance that works as a web-request dispatcher, distributing the web requests across a set of Web/J2EE Cluster nodes. In advanced configurations, it distributes the web requests even across non-cluster server instances and multiple vendor's servers. Load Balancing for EJB clusters is done at the container level and does not need any external configuration.

A Load Balancer is a variant of Web node. The steps involved in setting up a Load Balancer are:

- 1 The DIR and the config files have to be setup just as one would do for a Standalone instance.
- 2 As defined in Table 2 above, copy the `server-config.xml` and `web-lbconfig.xml` into the config DIR
- 3 Enable the Load Balancer in `web-config.xml`  
`<request-dispatcher enabled="true" name="web-lbconfig.xml"/>`
- 4 Configure the Load Balancer properties in `web-lbconfig.xml`. The nodes are referenced by their name:

```

<node name="first" type="pramati" host="localhost" web-
port="8484">
  <naming-port>9494</naming-port>
  <socket-pool min="20" max="50" idle-time-out="1000"/>
</node>

```

You can also configure node chooser and node group information from `web-lbconfig.xml`. For more information on node chooser, node group and failover refer to the *Technical Reference* document.

## Customizing Server Footprint

The Server framework allows customizing a server instance to the extent of choosing the specific set of services required by the application. This will result in just those services being started. The binary files (jar) for the remaining services can even be removed from the Server install directory. Services can be enabled and disabled by changing configuration in `server-config.xml`. More details are dealt in the following section.

## Server Architecture

### Overview

The Server framework is a system composed of a set of independent services to provide extensibility and enable assembling of server flavors such as Web-only server or EJB-only server. The Integration Service allows developers to decompose an application and enables reusable design. To address the need to separate the implementation of Services and eliminate crossdependencies resulting in compilation issues, the framework has been designed to enable parallel development, pluggability and modularity to suit custom requirements.

Pramati Services framework is the core services sandbox for Pramati Server, which enables services to be plugged in, and removed seamlessly with minor configuration changes. Pramati Services framework was conceived as a flexible component framework designed to facilitate the development of sophisticated enterprise solutions from a bunch of independent and interdependent services. The services framework helps greatly in reuse of services and thus enabling the decomposition of the application into a set of manageable services.

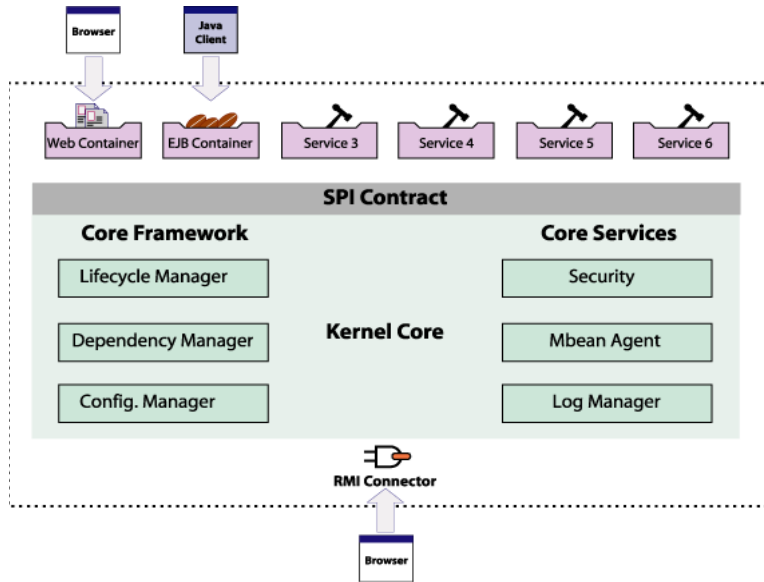


Fig Services Framework

## Architecture

Pramati Services framework comprises of the following core components:

- Life Cycle Manager
- Dependency Manager
- Configuration Manager

Pramati core services are

- Security
- JMX
- Logging

The services are classified into:

- *Services (Normal)* - These are services, which cannot be configured.
- *Configurable Services* - These are services, which can be configured using an external configuration file.
- *Re-configurable Services* – These are the services, which can be re-configured after successful deployment in the server framework.

Some of the default service configurations which comes with Pramati Services framework are:

- Naming Service
- Transaction Service
- Deploy Service
- Resource Service
- Web Container
- EJB Container
- JMS Service
- Management Service
- Cluster Service

## **Kernel**

Pramati Services framework (PSF) architecture comprises of the Kernel, which provides arbitration for all the framework components and takes care of all services level contracts. It creates and manages the Management Agent, the services repository and other global components required for the function of the services. PSF Kernel is responsible for starting and managing core framework components like the Lifecycle Manager, Dependency Manager and Configuration Manager. The Kernel provides the management and service registration capabilities and also logging, security, packaging, configuration and deployment functionalities.

Pramati Server creates, maintains and destroys the Kernel instance. Upon Kernel startup, the Dependency Manager invokes the Dependency Analyzer, which reads the list of services and their dependencies and categorizes them according to their types, that is Enabled, Circular, Disabled and Invalid. The types of dependencies will be discussed later in this chapter.

## **Core Services**

PSF Kernel comprises of three core services

- MBean Server
- Security
- Logging

## Core Components

### Lifecycle Manager

Based on the lifecycle of each service, the functionality of the Lifecycle Manager can be broadly classified as:

Table 9: Functionalities of a Lifecycle Manager

Function	Description
Loading the service class	The service class mentioned in the configuration file is loaded into the Kernel VM and instantiated of type Service. All classes which need to be started up as a service by the Lifecycle Manager should implement the Service Interface – <i>com.pramati.services.framework.Service</i> Class serviceClass = Class.forName(serviceClassName);
Call the no-args constructor of the service	The service implementation class, which needs to be started, should have a constructor with no argument signature. This is mandatory for starting the service as the Lifecycle Manager does not assume or try to obtain the initialization parameters of the service.
Initialize the service data	Every configurable service will have service data associated with them. These are service specific information, which needs to be taken care of when the service is deployed. The service data is initialized for identifying the service specific configuration.
Set Service context	Once the service data is initialized, the service context or the runtime environment for each service is setup, so the service can communicate with the kernel through the service context.
Configure service (Optional)	The next step after setting the service context is to configure the service. The services are configured only if they are set as 'configurable' or 're-configurable'.
Start service	After the service is configured, the service is started by instantiating the service implementation class file. Service serv = (Service) serviceClass.newInstance();
Reconfigure service	When a service is started as a re-configurable service, the service can be re-instantiated whenever its configuration file gets modified. This can be achieved by setting up a service specific listener which tarps all the configuration modification events. And in the event of the service modification, the life cycle manager reconfigures the service.
Stop service	The service can be stopped or disabled even when it is plugged in and running.
Destroy service	The service can be destroyed are unplugged from the services framework at anytime. Note that the service implementation class file remains undeleted but starting the service follows the entire initial lifecycle process including configuration and setting up new service context.

## Configuration Manager

The configuration manager reads and loads the service specific configuration into kernel memory. The configuration manager performs the following actions:

Table 10: Functionalities of a Configuration Manager

Function	Description
Get Node Information	Retrieves the Server Instance information which includes default login information like username, password and realm, RMI properties like socket time out, export port, SSL port, SSL protocol state and class file server port number.
Get Configuration	Reads the global configuration from the server-config.xml and also reads configuration of individual services. The Configuration object has a timestamp associated with it.
Save Service Properties	The properties of the services are saved in kernel memory space for faster processing. This holds true for services like the EJB Container, which has properties like min-pool size, pool size, low-activity interval, session-timeout and smart-code-generation.
Save Configuration	The default XML manager picks up the configuration DOM and writes to a location. Note that if there is any configuration, which has to be synced up, cluster wide cannot be done in the service config.
Add Configuration Listener	Any listener would be monitored at the frequency specified in the config. file. If there is no such entry, the default monitoring frequency of 30 sec is registered.
Register Configuration	Registers all external configurations, which are not part of Pramati Services Framework. A third party load balancer might send a configuration handler and a registration key to be registered with the framework.
Get Persistent Mode	The configuration can either be persisted in a file or a database. The configuration Manager reads the content of the <configuration-persistence> tag from the server-config.xml which includes persistent type, and information like driver name, url, username and password in case of db persistence.

## Dependency Manager

Dependencies can be classified as:

- Regular (Always)
- Optional

If a dependency is stated as 'always' in the configuration file of a service, then the dependent service should be started before starting the service. But if the dependency is optional, the dependent service need not be started before the service. Based on the rules the service can have the following dependency states:

- *Enabled*- In 2 services dependency, if service S1 depends on S2, S2 should be enabled for the dependency to reach this state.

- *Disabled*- In 2 services dependency, if service S1 depends on S2, and if S2 is disabled, the Dependency Manager throws an Exception and stops the server.
- *Invalid*- If a service has specified an unknown or non-configured service as the dependent service, the Dependency Manager throws an Exception and stops the server.
- *Circular*- Circular dependency state happens when two services point to each other as their dependent service. Hence the server would not be able to start any of the services.

The dependency information for services is specified in the `server-config.xml` file. The code snippet below indicates the service definition of the cluster service.

```
<service name="ClusterService" enabled="false"
  class-name="com.pramati.cluster.PramatiClusterService">
  <config-file>cluster-config.xml</config-file>
  <requires always="NamingService" />
</service>
```

Here 'name' indicates the service name, 'class-name' points to the service implementation file, 'config-file' holds the cluster specific configuration information, and 'requires' sets the dependency for the service. In the case of the cluster service, naming service should have been started before. This is an instance of regular dependency.

## Inter Service Dependency

The inter service dependency of the core services is depicted in the following

Table 11: Inter Service Dependency

Service	Startup Class	Dependency	Configuration
EJB Container	com.pramati.ejb.EJBContainer	NamingService, TransactionService, DeployService	None
Web Container	com.pramati.web.WebServer	Naming Service, Deploy Service	web-config.xml
Naming	com.pramati.naming.PramatiNamingServiceJRMPS	None	None
Transaction	com.pramati.jta.PramatiJTATransactionService	Naming Service	None
Deploy	com.pramati.j2ee.deploy.PramatiDeployService	Naming Service, Cluster Service (If enabled)	deploy-config.xml
Resource	com.pramati.resource.PramatiResourceService	Naming Service, Transaction Service, Deploy Service (If enabled).	resource-config.xml
JMS	com.pramati.jms.server.PramatiMessageService	NamingService, ResourceService, TransactionService (If enabled)	jms-config.xml
Management	com.pramati.admin.base.ManagementServiceImpl	NamingService, WebContainer (If enabled)	management-config.xml
Cluster	com.pramati.cluster.PramatiClusterService	NamingService	cluster-config.xml

table:

## Log Manager

The services framework supports an advanced logging mechanism providing an ability to monitor the services running in the framework. The logging itself comprises of the actual mechanism in which the messages are logged and the functionality of the Log Manager. The Log Manager forms the core component of the framework, which provides an interface between the services framework and the logging mechanism. It abstracts the logging mechanism from direct access. The Log Manager performs the following tasks:

- Get Logging properties

- Get Trace Admin
- Get Rotation Type
- Get File Size
- Set Rotation Type
- Set File Size
- Set Rotation time

Rotation is the process of saving the backup of the log in some location (db or file) and refreshing the log file to start as a new process. There can be two kinds of rotation 1. Rotation based on duration 2. Rotation based on size.

Log Manager is capable of setting the rotation type and managing the log files for each node in the cluster.

### **Services framework features**

- Implementation of a service is abstracted from all other services enabling ease of maintenance
- Contracts are defined upfront enabling cleaner separation
- No compile-time dependencies enabling loosely-coupled coexistence of services through SPI contracts
- Subsettable and Extensible, enabling adding and starting services in the same VM
- Modular, enabling for reduction in footprint and overhead
- Modular and Manageable

# Using Pramati Server Node Creator Utility

The NodeCreator is a utility for creating or removing nodes and clusters. You can configure and use the NodeCreator with both the command line and XMLs.

## Using NodeCreator with Command line

To run the NodeCreator:

Run NodeCreator.bat (for Windows) or NodeCreator.sh (for Unix) located at <install\_dir>/server/bin/.

Execute `java com.pramati.NodeCreator` class by passing the following values:

Table 12: Operations that can be performed using the Node Creator class

Operation	Description
-createj2ee	To create a Standalone J2EE node
-createj2eecluster	To create multiple J2EE nodes in a cluster
-createjms	To create a Message Server node
-createjmscluster	To create multiple Message Server nodes in a cluster

The various values (given as {parametername value} format) required to perform the above operations are:

Table 13: Values for the Node Creator class

Value	Description
-name	Name(s) of the nodes. In case of a cluster, the names are separated by a comma ','
-ip	The IP of the machine on which this node is being created. The default value is local host. In case of a cluster, the IPs are separated by a comma ','
-namingport	Naming port(s) of the nodes. In case of a cluster, the ports are separated by a comma ','
-httpport	HTTP port(s) of the nodes. In case of a cluster, the ports are separated by a comma ','
-clustername	In case of a cluster, the name of the cluster.

You can also specify the System Property 'install.root', if the program is not being run from the Server root.

Example for creating a J2EE Standalone node

```
java -Dinstall.root=. com.pramati.NodeCreator -createj2ee -name my
node1 -namingport 9191 -httpport 8181
```

## Example for creating a J2EE cluster

```
java -Dinstall.root=. com.pramati.NodeCreator -createj2eecluster -name my
node1,mynode2 -namingport 9191,9292 -httpport 8181,8282 -clustername myclus-
ter
```

The help snippet that shows up on the command shell is as follows:

```
Usage: java com.pramati.NodeCreator [operation] [option value] [flag]
```

## Using NodeCreator with XMLs

You can create a node or cluster using the `config-topology-standalone-template.xml` and `config-topology-cluster-template.xml` located at `<install_dir>/server/templates/`.

Using these XML templates, you can convert existing node types, create nodes for a cluster, and configure persistence properties for the database, EJB or Web.

Converting a node type is useful if you want to convert an existing, say Standalone J2EE Server to an EJB cluster type. To do this, open the `config-topology-cluster-template.xml`:

```
<configuration-topology>
  <!-- Valid Types - a J2EE (EJB/BOTH/WEB) Cluster or a JMS Cluster-->
  <!-- SAMPLE CONFIGURATION, change node definitions as required -->
  <cluster name="MyCluster" type="J2EE">
    <convert-nodes>
      <!-- These are nodes for conversion, the server "J2EESAS1" of the given ip in
      the <source-node> tag would be converted to the cluster node type given in the
      node definition. The target nodenames can be different in which case the
      directories will be copied.-->
      <node name="EJB1" type="EJB1-cluster" naming-port="9000" http-
      port="8000">
        <source-node ip="localhost" name="J2EESAS1" />
      </node>
    </convert-nodes>
  </cluster>
</configuration-topology>
```

Table 14: Description of attributes for `<convert-nodes>` tag

Attributes	Description
node name	Name(s) of the target node. That is the node that is to be created.
type	The type of the target node that is to be created.
naming-port	Naming port(s) of the target node.
http-port	HTTP port(s) of the target node.
source-node ip	The IP of the source machine on which this node is started. The default value is local host.
name	Name of the source node.

If you wish to create a J2EE Cluster, locate the `<create-nodes>` tag in the `config-topology-cluster-template.xml`:

```
<create-nodes>
  <node name="nodename1" type="both">
    <ip>localhost</ip>
    <naming-port>9292</naming-port>
    <http-port> 8282</http-port>
  </node>
  <node name="nodename2" type="both">
    <ip>localhost</ip>
    <naming-port>9293</naming-port>
    <http-port> 8283</http-port>
  </node>
</create-nodes>
```

Table 15: Description of attributes for `<create-nodes>` tag to create a cluster

Attributes	Description
node name	Name of the cluster node.
type	The type of the cluster - EJB, Web, or BOTH.
ip	The IP of the machine on which this node is being created. The default value is local host.
naming-port	Naming port of the node.
http-port	HTTP port of the node.

To define the properties for persistence, modify the `<configuration-persistence>` tag in the `config-topology-cluster-template.xml`:

```
<!-- use "db" for DB persistence -->
<configuration-persistence type="file">
  <property name="driver" value=""/>
  <property name="url" value=""/>
  <property name="user" value=""/>
  <property name="password" value=""/>
  <property name="tablename" value=""/>
</configuration-persistence>
<!--the default type here is "in-memory", use "db" for DB persistence -->
<ejb-cluster-persistence >
  <property name="driver" value=""/>
  <property name="url" value=""/>
  <property name="user" value=""/>
  <property name="password" value=""/>
```

```

        <property name="table-name" value=""/>
    </ejb-cluster-persistence>
    <!--the default type here is "in-memory", use "db" for DB persistence -->
    <web-cluster-persistence >
        <property name="url" value=""/>
        <property name="url" value=""/>
        <property name="user" value=""/>
        <property name="password" value=""/>
        <property name="table-name" value=""/>
    </web-cluster-persistence>
    <!-- the upload-config allows pre-configuring the created cluster.e.g.The
configuration for nodename1 would be uploaded
        to the DB and this configuration would be synced up to the other nodes.-
->
        <upload-config-source-node name="nodename1" />
</cluster>
</configuration-topology>

```

## Using - configuration flag for custom node configuration

You can use the `-configuration` flag with the NodeCreator utility and pass the name of a configuration file as a parameter. For example: `com.pramati.NodeCreator -configuration config.xml`

The `config.xml` file can have the following structure:

```

<server-configuration>
  <configuration service-name="ResourceService">
    <datasource name="myDS" connection-factory="myCF" transaction-participa-
tion="true" description="No Description">
      <pool-properties min-pool-size="10" max-pool-size="20" initial-pool-
size="2" refresh-interval-seconds="" connection-request-timeout-sec-
onds="" idle-timeout-seconds="" cache-size="" />
      <connection-validation sql="" class="" />
    </datasource>

    <connection-factory name="myCF" description="" classname="COM.cloud-
scape.core.RmiJdbcDriver" url="jdbc:cloud-
scape:rmi:D:\CTS\j2eects\db_cs40\DB1" authorized-by="Container">
      <login-parameters user="" password="" mask-password="false"/>
    </connection-factory>

    <-jms-adapter name="myAdapter" description="No Description" interface-
class="com.pramati.jms.client.JMSProviderImpl">

```

```
<properties>
  <property name="com.pramati.naming.cacheLookups" value="false"/>
  <property name="java.naming.factory.initial" value="com.pramati.nam-
    ing.PramatiLocalContextFactory"/>
  <property name="java.naming.provider.url" value="rmi://localhost:9090"/>
  <property name="com.pramati.force.standalone.ctx" value="false"/>
</properties>
</jms-adapter-->
</configuration>
</server-configuration>
```

As an example, the above will add the mentioned resources in the newly created nodes - the relevant entries will be added to the `resource-config.xml` files.

The supported resources for this tag are:

1. JDBC Connection Factories
2. Datasources
3. JMS Adapters

## Cluster Configuration using NodeCreator

Cluster configuration can be performed using the NodeCreator utility as shown in the following example:

### Usage Objective

To have a cluster with four nodes (a1, a2, a3 and a4) on four different machines (n1, n2, n3 and n4) respectively.

*Inputs:* A sample standalone node containing all the resources and configurations. An XML file giving details of the cluster conversion. The template of the XML file can be found in `$INSTALL_DIR/templates/config-topology-cluster-template.xml`.

*Requirements:* To restore all the configuration of the sample node and modify some of the configuration as per requirement.

### Steps Involved:

- 1 Start the Admin Service on each of the node where cluster nodes is required to be made. In this case (n1,n2,n3,n4). Admin Service can be started by using `runstartupsvc.bat` in `$INSTALL_DIR/server/bin` directory.
- 2 Use the NodeCreator utility to configure a cluster configuration. NodeCreator utility could be found in `$INSTALL_DIR/server/bin` directory. The command to use the nodecreator utility is  

```
nodecreator -config_topology { $path_of_config_xml } -configuration  
{ $path_of_config_xml } -writetopology
```

The writetopology option creates the topology XML, which can be used to launch the configured cluster nodes. The topology XML is generated in the same directory as the config-topology file. Use the nodecreator command without any parameters to see the help.

3 Use the node-creator utility to remove the configured cluster nodes. The command to use is `nodecreator -remove -topology { $path_of_topology_xml }`.

The topology XML is created while configuring the cluster if *-writetopology* option is used.

## Configuring J2EE Server Cluster using NodeCreator

You can configure a J2EE Cluster using the NodeCreator class. To run the NodeCreator Utility:

Run `NodeCreator.bat` (for Windows) or `NodeCreator.sh` (for Unix) located at `<install_dir>/server/bin/`.

Execute `java com.pramati.NodeCreator` class by passing the following values:

Table 16: Operations that can be performed using the Node Creator class

Operation	Description
-createj2ee	To create a Standalone J2EE node
-createj2eecluster	To create multiple J2EE nodes in a cluster
-createjms	To create a Message Server node
-createjmscluster	To create multiple Message Server nodes in a cluster

The various values (given as {parametername value} format) required to perform the above operations are:

Table 17: Values for the Node Creator class

Value	Description
-name	Name(s) of the nodes. In case of a cluster, the names are separated by a comma ','.
-ip	The IP of the machine on which this node is being created. The default value is local host. In case of a cluster, the IPs are separated by a comma ','.
-namingport	Naming port(s) of the nodes. In case of a cluster, the ports are separated by a comma ','.
-httpport	HTTP port(s) of the nodes. In case of a cluster, the ports are separated by a comma ','.
-clustername	In case of a cluster, the name of the cluster.

You can also specify the System Property 'install.root', if the program is not being run from the Server root.

Example for creating a J2EE cluster

```
java -Dinstall.root=. com.pramati.NodeCreator -createj2eecluster -name my
```

```
node1,mynode2 -namingport 9191,9292 -httpport 8181,8282 -clustername mycluster
```

The help snippet that shows up on the command shell is as follows:

```
Usage: java com.pramati.NodeCreator [operation] [option value] [flag]
```

## Creating LB Node using NodeCreator

Load Balancer node can be created using the NodeCreator utility as shown below:

```
java -Dinstall.root=. com.pramati.NodeCreator -createlb -name nodeA -namingport 9191 -httpport 8080 -topology <topology.xml>
```

In the above example a node named nodeA will be created and the node configuration is picked up from the topology.xml file created in the same directory as the config-topology-cluster-template.xml. The attributes are explained in the following table:

Table 18: Attributes for creating LB Nodes

Attribute	Description
-name	Name of the cluster node.
-namingport	Naming port(s) of the nodes. In case of a cluster, the ports are separated by a comma ','.
-httpport	HTTP port for the node.
-topology	Location to pickup the cluster topology and configuration.



A typical Pramati installation involves a Standalone server, Administration service, Web clients and Java Clients. The Server and Administration Service are installed on one machine and Java Clients the Pramati Client setup are on client machines or workstations.

#### **Pramati Server Installable/Configurable Components**

**Administration Service** Start with the Administration Service to configure, connect to and manage servers on the network. Through the Service, you can work with Pramati Server, Pramati Message Server or a Web/EJB node of a configured Pramati Cluster. The Administration Service needs to be started on every machine where the user wants to configure, start or manage servers. This service allows remote accessibility through communication with the remote administration services. Every Administration Service is attached to an authorized username and password. This service can communicate only with those administration services that have the same user name and password. The Service starts by discovering Pramati servers running on the LAN. The discovered servers are displayed in the browser along with their URL to connect.

**Standalone J2EE Server** Pramati server is a Java implementation of the Java2 Enterprise Edition specifications. It is developed to simplify the process of deploying Enterprise Java Beans beans and other J2EE applications.

**Cluster** A Cluster is a group of server nodes that work together to provide a more powerful, reliable application platform. The Cluster is transparent to the user that abstracts a single processing server for server operations such as configuration, tuning, deployment and monitoring. Cluster nodes could be of three types- EJB or Web or Both.

**Cluster BOTH node** While configuring a cluster node, Pramati administration service provides the option to select EJB or Web or both.

The Both option is selected when the application has beans, Java clients and web components. This options can be enabled while configuring a new node.

**Cluster EJB Node** in an EJB node, the EJB container alone gets started. This is advantageous when the application contains only beans or is a Java client.

**Cluster Web node** The web option is selected when the application contains JSPs and other web components. A standalone web node can act as a node balancer.

**Embedded JMS server** This JMS server runs in the same VM as the J2EE server. The embedded JMS server is configured and started by default along with the Pramati Server.

**Console** Pramati Server Management Console, JMX-based server management framework and Web-based console enables to manage and provide J2EE Solution. The Console provides a customizable infrastructure for enterprise server administrators. Through dynamic monitoring, tuning and control of applications, system administrators can optimally deploy resources and extract maximum performance from various deployment scenarios.

**Java Client** Clients that access EJB applications deployed on Pramati Server or other services provided by the server, by making either an HTTP request through a standard web browser or a standard Java Client that is compatible with the Remote Method Invocation (RMI) specification.

Pramati Server can be configured to run as a standalone Server or as a Node of a Cluster. The application components can be deployed on the Server in one of the following ways:

- **Locally (without load balance)** The application is deployed individually on one or more independent Servers on the enterprise network. An HTTP Server directs requests to each Server.

- **Clustered (with load balance)** Multiple Servers function as a single Server to balance application request loads. Console replicates parts or all of the application on the Server machines.

**Standalone JMS server** Pramati Message Server is a pure Java implementation of the Java Message Service™ (JMS™) 1.0.2 specification published by Sun Microsystems. It is designed for maximizing the portability of client applications without requiring the developer to learn all the functionalities of the Server.

### Installable components and Relevance to ISV Embedding

Component	Where	Install	Configure	ISV Kit
Admin Service	Server machine	Automatically installed as part of Server installation (PServer30.exe or jar)	Will need a ONE TIME setup of the service/daemon using install_services.bat (on unix: prs) scripts under the Services DIR.	
Pramati Server (stand-alone)	Server machine	Install PServer30.exe (or .jar)	Configured using the Pramati Server console as a StandAlone server. In a single Pramati Server installation multiple servers/nodes may be configured (each identified by a unique name).	
Pramati Cluster	Server/Cluster-node machine	Install PServer30.exe (or .jar)	Configured using the Pramati Server console as a cluster. In a single Pramati Server installation multiple clusters can be configured(each identified by a unique name).	
Cluster BOTH node	Server/Cluster-node machine	Install PServer30.exe (or .jar)	Configured using the Pramati Server console as a cluster node. In a single Pramati Server installation multiple nodes can be configured with a unique name and can be started along with the server at the same time.	
Cluster EJB Node	Server/cluster-node machine	Install PServer30.exe (or .jar)	Configured using the Pramati Server console as a cluster node. In a single Pramati Server installation multiple nodes can be configured with a unique name and can be started along with the server at the same time.	

Component	Where	Install	Configure	ISV Kit
Cluster Web node	Server/Cluster-node machine	Install PServer30.exe (or .jar)	Configured using the Pramati Server console as a cluster node. In a single Pramati Server installation multiple nodes can be configured with a unique name and can be started along with the server at the same time.	
Java Clients	Client workstations	No separate installation. From the installation on the Server machine, need to copy the 'lib' folder onto the client machine.	Setup jndi.props for server connectivity information.	
Embedded JMS Server	Server machine	Automatically installed as part of Server installation (PServer30.exe or .jar)	By default Embedded JMS is started. Option available to disable this as part of Server Configuration, from the Server Console.	
Standalone JMS Server	Server machine	Installed as part of Server installation (PServer30.exe or .jar)	Configured using the Pramati Server console as a standalone JMS server. In a single Pramati Server installation multiple nodes of the JMS server can be configured.	

## Sample configuration layouts

Some of the common configuration scenarios are shown in this section.

### Scenario A: All installations on the host machine

Nothing exists in the De-militarized Zone (DMZ) behind the firewall. Pramati Server is on the host machine. Pramati Web Container handles all HTTP requests.

### Scenario B: Pramati Web Container in DMZ

Pramati Web Container and EJB Container are configured in two different zones: the Web Container is in the DMZ and handles all the

HTTP requests. The EJB Container is on the host machine and serves EJBs.

### **Scenario C: Third-party HTTP server**

An existing HTTP server is configured with Pramati Server. While Pramati Server serves EJBs and JSP pages (dynamic content), HTTP server serves static content. In this scenario, Administrators must install Pramati WebGate, the HTTP server plug-in facility, for Pramati Server to work with other HTTP servers.

### **Scenario D: Third-party HTTP server + Servlet engine**

HTTP server with Servlet engine serves HTTP requests from the DMZ, while Pramati EJB Container serves EJBs from the host machine. In this scenario, administrators must place Pramati Server-specific client.jar in the classpath of the Servlet engine.

Hosting applications on multiple independent Servers is the simplest of the Server configurations. Each Server is installed on the local host or a remote host. In this configuration, you deploy the complete application on each Server machine. To do this, you use the Deploy Tool in the Console to repeatedly deploy the application on each running Server.

When you deploy an application this way, the session is neither backed up nor shared between two or more Servers. This implies that load balancing and fail over is not available to the application.

To deploy applications across multiple Pramati Servers while sharing a common external HTTP server, install Pramati WebGate, the HTTP Server Plug-in that ships with the Server.

Deployment descriptors are standard text files, formatted using XML notations, that are packaged in the J2EE applications. These deployment descriptors are used to define components and operating parameters for applications. This chapter discusses the following DTDs:

- EJB 1.1 and EJB 2.0
- Servlets 2.2 and Servlets 2.3
- Pramati J2EE Server

These files are placed under `pramati_util.jar`, which is located in `<install_dir>/server/lib/pramati`. The contents of the DTD elements are case sensitive. The comments in the DTDs specify additional requirements of syntax and semantics that are not expressed by the DTD mechanism.

### **DTD for EJB 1.1 and 2.0 Deployment Descriptor**

The deployment descriptor is a part of the `ejb-jar` file producer and consumer. This covers the passing of enterprise beans from the Bean Provider to the Application Assembler, and from the Application Assembler to the Deployer.

An `ejb-jar` file produced by the Bean Provider contains one or more enterprise beans and does not contain application assembly instructions. An `ejb-jar` file produced by an Application Assembler contains one or more enterprise beans, plus application assembly information describing how the enterprise beans are combined into a single application deployment unit.

*The J2EE specification defines how enterprise beans and other application components contained in multiple `ejb-jar` files can be assembled into an application. Server supports both EJB 2.0 and 1.1 applications, and ships with both the DTDs.*

The role of the deployment descriptor is to capture the declarative information (i.e. information that is not included directly in the enterprise beans' code) that is intended for the consumer of the ejb-jar file.

There are two basic kinds of information in the deployment descriptor:

- **Enterprise beans' structural information** - This describes the structure of an enterprise bean and declares the bean's external dependencies. Providing structural information in the deployment descriptor is mandatory for the ejb-jar file producer. The structural information cannot be changed as it breaks the enterprise bean's function.
- **Application assembly information** - Application assembly information describes how the enterprise bean in the ejb-jar file is composed into a larger application deployment unit. Providing assembly information in the deployment descriptor is optional for the ejb-jar file producer. Assembly level information can be changed without breaking the enterprise bean's function, although it may alter the behavior of an assembled application.

The following types of configuration and deployment information are supported in Server for EJB application deployment descriptors:

- Enterprise Bean's Name, Class and Type
- Enterprise Bean's Remote Home and Local Home Interface (EJB 2.0 Beans only)
- Enterprise Bean's Remote and Local Interface
- Re-entrancy indication
- Session bean's state management type, transaction demarcation type
- Entity bean's persistence management, primary key class, prim-key-class element, abstract schema name, Container-managed fields
- Container-managed relationships (EJB 2.0 Beans only)
- Finder and select queries. Select queries for EJB 2.0 Beans only
- Environment entries

- Resource manager connection factory references
- Resource environment references, EJB references
- EJB local references (EJB 2.0 Beans only)
- Security role information
- Message-driven bean's destination, message selector, acknowledgment mode
- Method Permission Information
- Transaction Attributes

*To view the XML DTDs shipped with the server, refer `ejb-jar_1_1.dtd.xml` and `ejb-jar_2_0.dtd.xml` located in `pramati_util.jar`*

## DTD for Web Deployment Descriptor

The Web deployment descriptors convey the elements and configuration information of web applications between application developers, application assemblers, and deployers. For backward compatibility of applications written to the 2.2 version of the API, web containers are also required to support the 2.2 version of the deployment descriptor.

*Server ships with DTDs for applications written to both Servlet 2.2 and 2.3 version of the API.*

The following types of configuration and deployment information are supported in Server for the Web application deployment descriptors:

- ServletContext Init Parameters
- Session Configuration
- Servlet Declaration
- Servlet Mappings
- Application Lifecycle Listener classes
- Filter Definitions and Filter Mappings
- MIME Type Mappings

- Welcome File list
- Error Pages
- Security Information
- Taglib
- Syntax for looking up JNDI objects (env-entry, ejb-ref, ejb-local-ref, resource-ref, resource-env-ref)

*To view the XML DTDs shipped with the server, refer web-app\_2\_2.dtd.xml and web-app\_2\_3.dtd.xml located in pramati\_util.jar*

## DTD for Pramati J2EE Server

The Server supports J2EE 1.3 and previous versions of DTDs. Supporting the previous version ensures that applications written in previous versions of the specification can be deployed on products supporting the current version of this specification.

It also ensures that there are no restrictions in mixing versions of deployment descriptors within a single application; any combination of valid deployment descriptor versions are also supported.

```
<!--  
Copyright 2000-2001 Pramati Technologies Private Ltd , #  
301 White House,  
Begumpet, HYDERABAD 5000016, INDIA. All rights reserved.  
This product or document is protected by copyright and  
distributed  
under licenses restricting its use, copying, distribu-  
tion, and  
decompilation. No part of this product or documentation  
may be  
reproduced in any form by any means without prior written  
authorization  
of Pramati and its licensors, if any.  
-->  
<!--
```

This is the XML DTD for pramati-j2ee-server.xml. The xml must include

a DOCTYPE of the following form:

```
<!DOCTYPE pramati-j2ee-server PUBLIC "-//Pramati Technologies //DTD Pramati J2ee Server 3.0//EN" 'http://www.pramati.com/dtd/pramati-j2ee-server_3_0.dtd'>
```

```
-->
```

```
<!--
```

Element auto-start specifies whether the application, if prepared, should be started

on the server start time. Valid values are "TRUE" and "FALSE".

Used in:pramati-j2ee-server

```
-->
```

```
<!ELEMENT auto-start (#PCDATA)>
```

```
<!--
```

This element represents the external connection factory that is used in mapping

Message Driven bean destinations

Example:<conn-factory>myConnFactory<conn-factory>

myConnFactory is the name of the connection factory on the external JMS Server

Used in:destination-mapping

```
-->
```

```
<!ELEMENT conn-factory (#PCDATA)>
```

```
<!--
```

The description element is used to provide text describing the parent

element

```
-->
```

```
<!ELEMENT description (#PCDATA)>
```

```
<!--
```

This element represents the external destination(queue/topic)that is used in mapping

Message Driven bean destinations

Example:<destination-link>myQueue<destination-link>

myQueue is the name of the destination on the external JMS Server

Used in:destination-mapping

-->

<!ELEMENT destination-link (#PCDATA)>

<!--

This element provides the external mapping information related

to the external JMS server for MessageDrivenBean destinations

Used in:ejb

-->

<!ELEMENT destination-mapping (destination-link, conn-factory)>

<!--

This element represents an EJB inside a deployable ejb module

for an application.Contains information for its creation and maintenance

like pool sizes,jndi-name - by which the server identifies the bean and so on.

Used in:ejb-module

-->

<!ELEMENT ejb (name, max-pool-size?, min-pool-size?, low-activity-interval?, is-secure, jndi-name?, query-mapping\*, session-timeout?, ejb-ref\*, ejb-local-ref\*, resource-mapping\*, resource-env-ref\*, server-session?, thread-pool?, destination-mapping?, run-as-principal?)>

<!-- ejb-link element is ejb jndi name or ejb local name based on whether it is ejb-ref or ejb-local-ref.

Used in : ejb-local-ref, ejb-ref

-->

```
<!ELEMENT ejb-link (#PCDATA)>
<!--
This element represents a ejb-local-ref for the EJB
Used in:ejb,web-module
-->
<!ELEMENT ejb-local-ref (ejb-ref-name, ejb-link)>
<!--
This element contains the information for an ejb-module
for the application
This module contains related information for the ejb com-
ponents mostly related to
the mapping information for them.
Used in:pramati-j2ee-server
-->
<!ELEMENT ejb-module (name, ejb+)>
<!--
This element represents a ejb-local-ref for an EJB/web
module
Used in:ejb,web-module
-->
<!ELEMENT ejb-ref (ejb-ref-name, ejb-link)>
<!--
This element represents the name of the ejb reference
Used in:ejb-ref,ejb-local-ref
-->
<!ELEMENT ejb-ref-name (#PCDATA)>
<!--
This element specifies whether access to the bean methods
is secure or not
Used in:ejb
Valid Values :true,false
-->
```

```
<!ELEMENT is-secure (#PCDATA)>
<!--
This element represents the name given to the EJB which
will be unique in the server name space
Used in:ejb
-->
<!ELEMENT jndi-name (#PCDATA)>
<!--
This element represents the low activity interval for the
pools
used by the server
Used in:ejb
-->
<!ELEMENT low-activity-interval (#PCDATA)>
<!--
This element represents the maximum messages for a
server session pool for a MessageDrivenBean
Used in:server-session
-->
<!ELEMENT max-messages (#PCDATA)>
<!--
This element represents the max pool size for the pools
used by the server
Used in:ejb,server-session,thread-pool
-->
<!ELEMENT max-pool-size (#PCDATA)>
<!-- represents the name of the finder method as read by
loading the bean home class.
Used in : query-mapping
-->
<!ELEMENT method-name (#PCDATA)>
```

```
<!-- represents the name of the parameter of the finder
method.
Used in : query-mapping
-->
<!ELEMENT method-param (#PCDATA)>
<!--
This element represents the min pool size for the pools
used for the EJB
Used in:ejb,server-session,thread-pool
-->
<!ELEMENT min-pool-size (#PCDATA)>
<!--
This element represents the name of the deployable module
Used in:ejb-module,web-module,role-mapping
-->
<!ELEMENT module-name (#PCDATA)>
<!--
The element name specifies the name of the conerned
entity
Used in :ejb-module,web-module,ejb
-->
<!ELEMENT name (#PCDATA)>
<!--
The pramati-j2ee-server element is the root element of
the pramati-j2ee-server
xml document. This encapsulates all the information
required for an application to be deployed on to Pramati
Server.
as an example resource mapping information like mapping a
resource onto
an actual resource on the server or mapping a security
role onto an actual server role .
```

All this information is persisted in this file and is picked up along with the application whenever Pramati Server uses this application

-->

```
<!ELEMENT pramati-j2ee-server (description?, vhost-name, auto-start, realm-name?, (ejb-module | web-module)+, role-mapping*)>
```

<!-- query-mapping element appears if the ejb is an ejb1.1 bean. It is used to map the query-name in the queries.props

to the finder method-name. It may also contain some method-param elements

Used in : ejb

-->

```
<!ELEMENT query-mapping (method-name, query-name, method-param?)>
```

<!-- represents the name of query which is present in the queries.props and have to be mapped to the method-name in the query-mapping element.

Used in : query-mapping

-->

```
<!ELEMENT query-name (#PCDATA)>
```

<!--

This element specifies the security realm on which the application is deployed

the default realm is 'system'

Used in:pramati-j2ee-server

-->

```
<!ELEMENT realm-name (#PCDATA)>
```

<!--

This element gives the details for a resource env reference

Used in:ejb,web-module

-->

```
<!ELEMENT resource-env-ref (resource-env-ref-name,  
resource-env-ref-type, resource-env-ref-link)>  
<!--  
This element specifies the link for the resource env ref  
declared  
Used in:resource-env-ref  
-->  
<!ELEMENT resource-env-ref-link (#PCDATA)>  
<!--  
This element specifies the name for the resource env ref  
declared  
Used in:resource-env-ref  
-->  
<!ELEMENT resource-env-ref-name (#PCDATA)>  
<!--  
This element specifies the type of the resource env ref  
declared  
Used in:resource-mapping  
-->  
<!ELEMENT resource-env-ref-type (#PCDATA)>  
<!--  
This element specifies the resource link in the server  
in the case of resource-mapping.This represents the  
actual resource name in the server namespace  
Used in:resource-mapping  
-->  
<!ELEMENT resource-link (#PCDATA)>  
<!--  
This element gives the details about the ejb resource  
mapping onto  
a resource on the server specifying the name and type of  
the resource  
Used in:ejb,web-module
```

```
-->
<!ELEMENT resource-mapping (resource-name, resource-type,
resource-link)>
<!--
This element gives the name of the resource used in
resource mapping
for an EJB.this name is used in the application code.
Used in:resource-mapping
-->
<!ELEMENT resource-name (#PCDATA)>
<!--
This element specifies the type of the resource used in
resource mapping
Used in:ejb
-->
<!ELEMENT resource-type (#PCDATA)>
<!--
This element specifies the actual role on the server to
which the role in
the module maps on to
Used in:role-mapping
-->
<!ELEMENT role-link (#PCDATA)>
<!--
This element specifies the role mapping information for
mapping
module level roles onto actual server roles
Used in:pramati-j2ee-server
-->
<!ELEMENT role-mapping (module-name, role-name, role-
link)>
<!--
```

This element specifies the role name declared in the module which would

be mapped onto an actual role in the server

Used in:role-mapping

-->

```
<!ELEMENT role-name (#PCDATA)>
```

<!--

This element specifies the principal/identity that would be used to

invoke methods on the EJB

Used in:ejb

-->

```
<!ELEMENT run-as-principal (#PCDATA)>
```

<!--

This element specifies the server-session pool details for a MessageDrivenBean

Used in:ejb

-->

```
<!ELEMENT server-session (max-messages, min-pool-size, max-pool-size)>
```

<!-- session-timeout element is an optional element which appears if the ejb is

a stateful session bean.

Used in : ejb

-->

```
<!ELEMENT session-timeout (#PCDATA)>
```

<!--

This element specifies the details for the thread pool used for a MessageDrivenBean

for an EJB

Used in:ejb

-->

```
<!ELEMENT thread-pool (min-pool-size, max-pool-size)>
```

```
<!--
```

This element specifies the virtual host on which the application is deployed

Example:

```
<vhost-name>www1.pramati.com</vhost-name>
```

by default 'default' is the value given and all applications are deployed using the same

Used in:pramati-j2ee-server

```
-->
```

```
<!ELEMENT vhost-name (#PCDATA)>
```

```
<!--
```

This element contains the information for a web-module for the application

This module contains related information for the web components.

Used in:pramati-j2ee-server

```
-->
```

```
<!ELEMENT web-module (name, module-name, ejb-ref*, ejb-local-ref*, resource-mapping*, resource-env-ref*)>
```

## Pramati OR Map DTD

This OR Map DTD, pramati-or-map\_3\_0.dtd, located under <install\_dir>/server/templates/dtds is a proprietary DTD used to perform OR mapping before the applications are deployed.

```
<!--
```

Copyright 2000-2001 Pramati Technologies Private Ltd ,  
# 301 White House,  
Begumpet,Hyderabad-5000016, India.

All rights reserved.

This product or document is protected by copyright and distributed

under licenses restricting its use, copying, distribution, and

decompilation. No part of this product or documentation may be

reproduced in any form by any means without prior written authorization of Pramati and its licensors, if any.

-->

<!--

This is the XML DTD for the Pramati O-R mapping document. All Pramati O-R

mapping documents must include a DOCTYPE of the following form:

```
<!DOCTYPE or-mapping PUBLIC "-//Pramati Technologies //
DTD Pramati OR Map 3.0//EN" 'http://www.pramati.com/dtd/
pramati-or-map_3_0.dtd'>
```

-->

<!--

The following conventions apply to all Pramati O-R mapping document

elements unless indicated otherwise.

- In elements that contain PCDATA, leading and trailing whitespace in the data may be ignored.

- In elements whose value is an "enumerated type", the value is

  - case sensitive.

-->

<!-- cmp-field element represents the name of the container managed field of the

entity bean.

Used in : field-mapping

-->

```
<!ELEMENT cmp-field (#PCDATA)>
```

<!-- Deprecated tag. Kept to have backward compatibility of pramati-or-mapping.xml.

Now the functionality corresponding to this has been replaced by isolation-level tag.

Used in : ejb

-->

```
<!ELEMENT concurrency-type (#PCDATA)>
```

<!-- datasource-name element specifies the data source which contains the table to which the bean is mapped.

Used in : ejb

-->

```
<!ELEMENT datasource-name (#PCDATA)>
```

<!-- db-field specifies the database field name corresponding to the cmp-field element value in the same field-mapping element of an ejb.

Used in : field-mapping, deferred-key

-->

```
<!ELEMENT db-field (#PCDATA)>
```

<!-- deferred-key specifies the database field name corresponding the deferred primary key of the bean.

Used in : ejb

-->

```
<!ELEMENT deferred-key (db-field)>
```

<!-- ejb element represent the Enterprise Java Bean of type entity.

Used in : ejb-jar

-->

```
<!ELEMENT ejb (concurrency-type?, exclusion-type, ejb-name, datasource-name, table-name, isolation-level?, schema-name?, field-mapping*, deferred-key*)>
```

<!-- ejb-jar element represents ejb jar in the enterprise archive

Used in : or-mapping

-->

```
<!ELEMENT.ejb-jar (jar-name, (ejb | relationship)+)>
<!--.ejb-name represents the ejb name of the bean as specified in the ejb-jar.xml.
Used in :.ejb
-->
<!ELEMENT.ejb-name (#PCDATA)>
<!--.exclusion-type element can have two values
    1.exclusive - represents that the bean exclusively accesses and updates the table
        and optimizations can be made based on the assumption that no other source
        modifies the database table.
    2.non-exclusive - represents that the database table can be accessed by some other
        sources than just the bean.
Used in :.ejb
-->
<!ELEMENT.exclusion-type (#PCDATA)>
<!--.field element represents the values of lhs or rhs table field in the join condition.
Used in :.lhs, .rhs
-->
<!ELEMENT.field (#PCDATA)>
<!--.field-mapping element represents a field mapping node consisting of a cmp-field
vs db-field mapping.
Used in :.ejb
-->
<!ELEMENT.field-mapping (cmp-field, db-field)>
<!--.isolation-level element represents the isolation level to be followed
for accessing and updating the bean. For ejb1.1 beans currently supported value
```

for isolation level are repeatable-read, optimistic-repeatable-read, read-committed.

For ejb2.0 the only supported vales for isolation-level is repeatable-read.Please

refer to Pramati Server Documentation for details.

Used in : ejb

-->

```
<!ELEMENT isolation-level (#PCDATA)>
```

<!-- jar-name element represents the name of a jar in the enterprise archive.

Used in : ejb-jar

-->

```
<!ELEMENT jar-name (#PCDATA)>
```

<!-- join element encapsulates a set of join conditions for the given relationship.

Used in : relationship

-->

```
<!ELEMENT join (join-condition+)>
```

<!-- join-condition element represents one join condition for a relationship.

Used in : join

-->

```
<!ELEMENT join-condition (lhs, rhs)>
```

<!-- lhs element represents the left hand side part of the join condition. It consists of a

table name and a field name to be equated to the right hand side counterpart.

Used in : join-condition

-->

```
<!ELEMENT lhs (table, schema?, field)>
```

<!-- or-mapping element is the root element of Pramati O-R mapping (pramati-or-map.xml)

document.

```
-->
<!ELEMENT or-mapping (ejb-jar+)>
<!-- relationship element represents a relationship node
with one or more
join conditions encapsulated in a join node.
Used in : ejb-jar
-->
<!ELEMENT relationship (relationship-name, join+)>
<!-- relationship-name represents the name of the rela-
tionship.
Used in : relationship
-->
<!ELEMENT relationship-name (#PCDATA)>
<!-- rhs element represents the right hand side of the
equality in the
join condition. It consists of a table name and a table
field.
Used in : join-condition
-->
<!ELEMENT rhs (table, schema?, field)>
<!-- schema element represents the values of lhs or rhs
schema in the join condition.
Used in : rhs, lhs
-->
<!ELEMENT schema (#PCDATA)>
<!-- schema-name element represents the name of the
schema which contains the table to which
the bean is mapped to for its field mapping.
Used in : ejb
-->
<!ELEMENT schema-name (#PCDATA)>
<!-- table element represents the values of lhs or rhs
table in the join condition.
```

Used in : rhs, lhs

-->

```
<!ELEMENT table (#PCDATA)>
```

<!-- table-name element represents the name of the table to which the bean is mapped to for its field mapping.

Used in :.ejb

-->

```
<!ELEMENT table-name (#PCDATA)>
```

# 2

## *Appendex 1: Installer Builder*

---

For support on downloading and using Pramati Installer Builder, contact [support@pramati.com](mailto:support@pramati.com). The Installer Builder is a downloadable zip file consisting of the Installation Program, Pramati utility classes, API, JARs, compilation scripts, templates, and documents. The extracted files are organized in a directory structure.

### **Quick configuration**

To get started with configuring the Product Installer:

- include a deployable ISV application EAR
- include ISV documentation
- include database setup, drivers and DDL scripts
- include resource, security, and Server configuration details
- rebrand the installer with ISV images

### **Step 1: Creating a project directory**

Create a project directory under the ISV Kit installation directory similar to the one already provided for fooBank: `<isvkit_root>/applications/samples/fooBank`.

This directory, hereafter referred to as `/project_dir`, should contain the following application-related files:

- EAR. For example, `fooBank/app`
- docs. For example, `fooBank/docs`
- resources such as drivers and scripts. For example, `fooBank/resource/db`
- application configuration XMLs. For example, `fooBank`

## Step 2: Modifying the XMLs

The /project\_dir should contain the following XMLs:

- assembly.xml, which assembles the application with Pramati Server
- config.xml, which contains server, resource, and security related information
- installer\_messages.xml, which contains ISV messages that would be displayed to end users during the final product installation.

The XML templates are provided with default values in the following location: <isvkit\_root>/pramati\_wizard\_components/templates

### Specifying server configuration

The embedded Server installs with certain default settings specified by the ISV in config.xml. End users can modify some of these settings, such as port numbers, from the installation wizard.

The ISV specifies the following server configuration properties:

- server name
- port number
- HTTP port
- HTTPS port

The following tag in config.xml refers to server configuration:

```
<server-configure>
    <server-name>default</server-name>
    <port-number>9191</port-number>
    <httpport>8181</httpport>
    <httpsport>443</httpsport>
</server-configure>
```

### Specifying the application in config.xml

The application's EAR and documentation is placed in /project\_dir. The following tag in config.xml refers to the EAR:

```
<application>foobank\App\FooBank.ear</application>
```

Note that `config.xml` does not explicitly refer to the location of application-related documentation.

#### Specifying data resource

When an end user installs the ISV's product, Pramati Server installs silently. It is pre-configured according to the resource configuration data provided in `config.xml`.

The driver name and DDL script required to create a data resource can be given any name. The following code shows the data resource tag used by the foobank sample. An ISV should adapt this by substituting application-specific values.

```
<data-resource>
    <name>demo</name>
    <!--JAR below is copied into java_home/jre/lib/ext on host-->
    <driver_jar_location>
        foobank/resource/db/classes12.zip
    </driver_jar_location>
    <!-- Below SQL script creates tables and populates them -->
    <ddl_script_location>
        foobank/resource/db/oracle.sql
    </ddl_script_location>
    <!-- Below resource info is copied into server-config.xml-->
    <url>jdbc:oracle:thin:@db1:1521:db1</url>
    <driver>oracle.jdbc.driver.OracleDriver</driver>
    <user>scott</user>
    <password>tiger</password>
</data-resource>
```

#### Specifying mail resource

An ISV's application may also use mail resources, which are also specified in `config.xml`. A sample mail resource:

```
<mail-source>
    <user>john</user>
    <!-- password is subsequently encrypted -->
    <password>john123</password>
    <name>mail</name>
```

```

    <host>mail.company.com</host>
    <from>john@company.com</from>
</mail-source>

```

### Specifying user information

Role names are specified in `server-config.xml`, which can be later added to a user group from the Server Management Console by the application administrator.

### Step 3: Rebranding the installer

The ISV can rebrand the Product Installer by modifying values in the `assembly.xml` and `installer_messages.xml` files.

The following parameters can be configured in `assembly.xml`:

Title	Title of the application
Left image	Image that appears on the left panel
Frame Icon	Icon for the frame
Pack Name description	Description provided within the pack name

The following parameters can be configured in `installer_messages.xml`:

CDKeyPanel	Contact information
ServerSummaryPanel	Messages that appear after the server has started

### Step 4: Assembling and compiling the installable binary

After all required files have been placed in `/project_dir`, the ISV Product Installer can be built as shown:

- 1 Open the `compile.bat` file in an editor from the location `<isvkit_root>/Pramati/src/pramati_wizard_components`.
- 2 Edit this file to specify the classpath as follows:

```

SET IZPACK_ROOT = ISVKit_root\izpack
SET CLASSPATH=
%IZPACK_ROOT%\lib\installer.jar;%IZPACK_ROOT%\lib\frontend.jar;
%IZPACK_ROOT%\lib\compiler.jar;%IZPACK_ROOT%\lib\uninstaller.jar;
javac -d
IZPACK_ROOT%\bin\classes com/pramati/isvkit/util/*.java

```

- 3 Run the compile.bat file from the command prompt.
- 4 The application has to be compiled and stored as an installable JAR. To do this, go to <isvkit\_root>/izpack/bin and execute:

```
compile ISVKit_root\installer_builder\JBank\assembly.xml -b  
ISVKit_root -o <location>/<jar-name>.jar
```

**EXAMPLE**

```
C:\ISVTesting\ISVKit_root\izpack\bin>compile  
C:\ISVTesting\ISVKit_root\installer_builder\JBank\assembly.xml -b  
C:\ISVTesting\ISVKit_root -o c:\temp\japp.jar
```

This compiles and stores an installable JAR in the specified location.

- 5 Edit /isvkit\_root/izpack/bin/pack.bat. Replace fooBank.jar with your application-specific JAR. For example, c:\temp\japp.jar.
- 6 The japp.jar is the final end user JAR. This JAR starts the server and deploys the application. Execute this JAR using the command `java -jar japp.jar`.



# 3

## *Appendix 2: Extending the Installer Builder*

---

Pramati Installer Builder can be extended to provide more power and flexibility to the ISV Product Installer, such as custom validation in panels, custom setup after installing the product at customer site, and new panels.

### **Custom validation**

#### **Extending logic in panels**

The default panels present in the ISV Kit can be extended. The following code segment shows how to extend the logic in the default panels:

```
public class CustomPanel extends IzPanel
{
public CustomPanel (InstallerFrame parent, InstallData idata)
{
super(parent,idata);
// Write your UI code here
GridBagLayout gb=new GridBagLayout();
GridBagConstraints gbc = new GridBagConstraints();
gbc.gridx=0;
gbc.gridy = 0;
gbc.weightx = 0.0;
gbc.weighty=0;
gbc.anchor=GridBagConstraints.NORTHWEST;
gbc.insets = new Insets( 15,10,2,0);
gbc.fill=GridBagConstraints.NONE;
add(new JLabel("CustomLabel",gbc);
}
public void show()
{
super.show()
//Write the validation required to show the panel in wizard.
}
```

```
public boolean isValidated()
{
//Give the Validation to navigate to the another panel here.
// If the validation returns false, you can't navigate further.
}
}
```

This class must be located in the folder <ISVKit\_root>/installer\_builder/installerJavaSources.

## Setting up custom panels

The following code shows how to build a custom panel:

```
public CustomPanel (InstallerFrame parent, InstallData idata)
{
super(parent,idata);
// The UI code should be written here.
}
public void show()
{
super.show()
//Write the validation required here }
public boolean isValidated()
{
//Write the validation required for navigation here.
}
}
```

The panel name must be included in the panel section of the sample application's assembly.xml file located in <ISVKit\_root>/installer\_builder/sample.

## Building the installer

To compile the panels, select compile.bat located in <ISVKit\_root>/installer\_builder/wizard\_src/.

The compiled classes created are placed in <ISVKit\_root>/izpack/bin/panels. Recompile these classes to build the installer by running the compile.bat located in <ISVKit\_root>/izpack/bin.



## *Appendix 3: Developing an ISV Product Installer*

---

Pramati Installer Builder acts as a reference for the operations described in this guide. The installer builder, based on IzPack is used to distribute application files as single JAR files. However, the ISV can work with any other generic Installer Builder like Zero G, InstallShield, or write an own Installer.

This section describes the steps to be performed when the Server is bundled with the application an a single JAR file.

### **Java requirement**

Pramati Server configuration APIs are in Java. So any Java-based Installer Builder such as Zero G and InstallShield can be used.

### **How to prepare the product bundle**

#### **Prepare the application**

Pramati Deploy Tool is used to prepare the deployable archive with deployment mappings provided in the packaged XMLs. The application can be deployed at the customer site only if the schema name and the table name matches with that of the ISV's application.

#### **Prepare the Installer**

The Installer defines the installation process to install the ISV components and the Server. It sets up a server instance, the required resources, and deploys the application. At the end of installation:

- The ISV Product Installer is installed
- The 'prepared' application is extracted
- Pramati Server is installed
- A Server instance is configured

While deploying the applications from the Installer:

- Resources and roles are setup
- Application is deployed
- Application is started

## Directory structure at customer site

When the customer installs the ISV product bundle, the following directory structure is created under /isvproduct\_root:

\PramatiServer

<Server dir>

\ISVApplication

\DBSchema

\Docs

---

## Distributing license keys

ISV applications provide secured access to the applications through license keys. The ISV has three options to select the license keys:

### Using Pramati License Key

Pramati license key is used. The ISV Product Installer prompts for this key. The access security is solely through Server licencing. The Pramati Installer Builder supports this mode only.

### Using proprietary license key

Application can be secured through a proprietary license key. The Server also requires Pramati License Key. So, the ISV must embed this key in the PLK. The customer has a single key, and the ISV Product Installer embeds the Pramati Server licence key. This key must be provided by the ISV Product Installer while installing the Server.

**Provide separate keys**

ISVs can provide two separate keys to customers—one for the ISV application and the other for Pramati Server. The Server key must be provided by the Product Installer while installing the Server.

**Customized administration**

ISVs provide the customer with specific management functions based on the ISV's application. Pramati Server provides a Web-based, customizable Pramati Server Management Console that can be reconfigured or reprogrammed to provide custom functionalities.



# 4

## *Appendix 4: Sample ISV Product Installer*

---

### **The foobank Sample**

The foobank sample that is shipped with the ISV Kit contains the application EAR, foobank.ear, and the following configurable XMLs and documentation used by the product installation wizard.

- assembly.xml
- installer\_messages.xml
- config.xml
- documentation and images

### **Modifying the sample**

The foobank sample is placed in a folder foobank under <install-dir>. It contains:

app	Folder that holds the EAR - foobank.ear
docs	Folder that contains documentation required for the application
Configurable XMLs	assembly.xml, installer_messages.xml, and config.xml
Licence.txt	Text file explaining the licensing mechanisms
Images	Images used by the ISV Product Installer, sample.gif and welcome.gif
resources	Folder that contains oracle.sql script

---

### **Data resource**

The sample is bound with a default datasource, demo. The following tag in config.xml shows the configuration. This is can be replaced with a user-defined datasource.

```
<Data-Resource>  
    <name>demo</name>  
</Data-Resource>
```

### DDL scripts

The sample has the database script `oracle.sql` located under the `ISVKit_root\installer_builder\Sample_fooBank\database`. This helps to run the DDL script. This can be set in the following tags in the `config.xml`:

```
<ddl_script_location>database/oracle.sql</ddl_script_location>
```

The following tag refers to the driver class that is required to create a `datasource` (using, for example, `classes12.zip`):

```
<driver_jar_location> fooBank/resource/db/classes12.zip <location>
```

### Specifying single roles

The `config.xml` consists of the default single roles. The Role-Name may be replaced with a name created by the user in the following tag:

```
<security>
<role>pramati</role>
</security>
```

### Server configuration

The sample consists of the configured server default with the naming port number, HTTP port and HTTPS port as 9191, 8181 and 443 respectively. These parameters can take different values, as shown in the code sample:

```
<server-configure>
  <server-name>default</server-name>
  <port-number>9191</port-number>
  <httpport>8181</httpport>
  <httpsport>443</httpsport>
</server-configure>
```

## Branding information

The XMLs, `assembly.xml` and `installer_messages.xml`, contain the following set of configurable branding parameters:

Title	Title of the application.
Left image	Image that appears on the left panel.
Frame Icon	Icon for the frame.
Pack Name description	description provided within the pack name.
CDKeyPanel	The contact information
ServerSummaryPanel	The messages that appear after the server is started

## Rebuilding the sample

To rebuild the sample, the following instructions are to be followed:

- 1 Run the `classpath.bat` file located in `ISVKit_root\installer_builder\installerJavaSources` from the command prompt to set the classpath.

- 2 The application has to be compiled and stored as an installable jar. To do this, navigate to `<ISVKit_root>\izpack\bin` and execute:

```
compile ISVKit_root\installer_builder\JBank\assembly.xml -b
ISVKit_root\installer_builder -o <location>/<jar-name>.jar
```

### EXAMPLE:

```
C:\ISVTesting\ISVKit_root\izpack\bin> compile
C:\ISVTesting\ISVKit_root\installer_builder\JBank\assemble -b
C:\ISVTesting\ISVKit_root\installer_builder -o c:\temp\japp.jar
```

Compiling this outputs the installable JAR in the location specified.

- 3 The JAR can be deployed by navigating to the JAR's location and specifying `java -jar <jar-name>.jar` in the command prompt.
- 4 This starts the server. Click on next to extract the server files.
- 5 Click on Next to add the resource details. Select next to proceed. Then, select 'Deploy application' to deploy the application.

