



Sun GlassFish Enterprise Server v3 Add-On Component Development Guide



Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820-7727-10
December 2009

Copyright 2009 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Enterprise JavaBeans, EJB, GlassFish, J2EE, J2SE, Java Naming and Directory Interface, JavaBeans, Javadoc, JDBC, JDK, JavaScript, JavaServer, JavaServer Pages, JMX, JRE, JSP, JVM, MySQL, NetBeans, OpenSolaris, SunSolve, Sun GlassFish, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. or its subsidiaries in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2009 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs brevets américains ou des applications de brevet en attente aux États-Unis et dans d'autres pays.

Cette distribution peut comprendre des composants développés par des tierces personnes.

Certains composants de ce produit peuvent être dérivés du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux États-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, le logo Solaris, le logo Java Coffee Cup, docs.sun.com, Enterprise JavaBeans, EJB, GlassFish, J2EE, J2SE, Java Naming and Directory Interface, JavaBeans, Javadoc, JDBC, JDK, JavaScript, JavaServer, JavaServer Pages, JMX, JRE, JSP, JVM, MySQL, NetBeans, OpenSolaris, SunSolve, Sun GlassFish, Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc., ou ses filiales, aux États-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux États-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de cette publication et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes chimiques ou biologiques ou pour le nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des États-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Contents

Preface	7
1 Introduction to the Development Environment for Enterprise Server Add-On Components	15
Enterprise Server Modular Architecture and Add-On Components	15
OSGi Alliance Module Management Subsystem	16
Hundred-Kilobyte Kernel	16
Overview of the Development Process for an Add-On Component	17
Writing HK2 Components	17
Extending the Administration Console	17
Extending the asadmin Utility	18
Adding Container Capabilities	18
Packaging and Delivering an Add-On Component	18
2 Writing HK2 Components	19
HK2 Component Model	19
Services in the HK2 Component Model	20
HK2 Runtime	20
Scopes of Services	20
Instantiation of Components in HK2	21
HK2 Lifecycle Interfaces	21
Inversion of Control	22
Injecting HK2 Components	22
Extraction	23
Instantiation Cascading in HK2	24
Identifying a Class as an Add-On Component	24
Using the Apache Maven Build System to Develop HK2 Components	25

3	Extending the Administration Console	27
	Administration Console Architecture	28
	Implementing a Console Provider	28
	About Administration Console Templates	29
	About Integration Points	30
	Specifying the ID of an Add-On Component	30
	Adding Functionality to the Administration Console	31
	Adding a Node to the Navigation Tree	32
	Adding Tabs to a Page	34
	Adding a Task to the Common Tasks Page	37
	Adding a Task Group to the Common Tasks Page	38
	Adding Content to a Page	40
	Adding a Page to the Administration Console	42
	Adding Internationalization Support	43
	Changing the Theme or Brand of the Administration Console	43
	Creating an Integration Point Type	45
	▼ To Create an Integration Point Type	45
4	Extending the <code>asadmin</code> Utility	47
	About the Administrative Command Infrastructure of Enterprise Server	47
	Adding an <code>asadmin</code> Command	48
	Representing an <code>asadmin</code> Command as a Java Class	48
	Specifying the Name of an <code>asadmin</code> Command	48
	Ensuring That an <code>AdminCommand</code> Implementation Is Stateless	49
	Example of Adding an <code>asadmin</code> Command	49
	Adding Parameters to an <code>asadmin</code> Command	50
	Representing a Parameter of an <code>asadmin</code> Command	50
	Identifying a Parameter of an <code>asadmin</code> Command	50
	Specifying Whether a Parameter Is an Option or an Operand	51
	Specifying the Name of an Option	51
	Specifying the Acceptable Values of a Parameter	52
	Specifying the Default Value of a Parameter	52
	Specifying Whether a Parameter Is Required or Optional	53
	Example of Adding Parameters to an <code>asadmin</code> Command	53
	Adding Message Text Strings to an <code>asadmin</code> Command	54

Enabling an <code>asadmin</code> Command to Run	57
Setting the Context of an <code>asadmin</code> Command	57
Changing the Brand in the Enterprise Server CLI	57
Examples of Extending the <code>asadmin</code> Utility	59
5 Adding Container Capabilities	63
Creating a Container Implementation	63
Marking the Class with the <code>@Service</code> Annotation	63
Implementing the Container Interface	64
Adding an Archive Type	66
Implementing the <code>ArchiveHandler</code> Interface	67
Creating Connector Modules	68
Associating File Types with Containers Using the <code>Sniffer</code> Interface	68
6 Packaging, Integrating, and Delivering an Add-On Component	71
Packaging an Add-On Component	71
Integrating an Add-On Component With Enterprise Server	72
Delivering an Add-On Component Through Update Tool	72
A Integration Point Reference	73
Integration Point Attributes	73
<code>org.glassfish.admingui:navNode</code> Integration Point	74
<code>org.glassfish.admingui:rightPanel</code> Integration Point	75
<code>org.glassfish.admingui:rightPanelTitle</code> Integration Point	76
<code>org.glassfish.admingui:serverInstTab</code> Integration Point	76
<code>org.glassfish.admingui:commonTask</code> Integration Point	77
<code>org.glassfish.admingui:configuration</code> Integration Point	77
<code>org.glassfish.admingui:resources</code> Integration Point	78
<code>org.glassfish.admingui:customtheme</code> Integration Point	78
<code>org.glassfish.admingui:masthead</code> Integration Point	79
<code>org.glassfish.admingui:loginimage</code> Integration Point	79
<code>org.glassfish.admingui:loginform</code> Integration Point	80
<code>org.glassfish.admingui:versioninfo</code> Integration Point	80

Index	83
--------------------	-----------

Preface

This document explains how to use published interfaces of Sun GlassFish™ Enterprise Server to develop add-on components for Enterprise Server. This document explains how to perform *only* those tasks that ensure that the add-on component is suitable for Enterprise Server.

This document is for software developers who are developing add-on components for Enterprise Server. This document assumes that the developers are working with an Enterprise Server distribution or GlassFish community distribution. Access to the source code of the GlassFish project is *not* required to perform the tasks in this document. This document also assumes familiarity with the Java™ programming language.

This preface contains information about and conventions for the entire Sun GlassFish Enterprise Server (Enterprise Server) documentation set.

Enterprise Server v3 is developed through the GlassFish project open-source community at <https://glassfish.dev.java.net/>. The GlassFish project provides a structured process for developing the Enterprise Server platform that makes the new features of the Java EE platform available faster, while maintaining the most important feature of Java EE: compatibility. It enables Java developers to access the Enterprise Server source code and to contribute to the development of the Enterprise Server. The GlassFish project is designed to encourage communication between Sun engineers and the community.

The following topics are addressed here:

- “Enterprise Server Documentation Set” on page 8
- “Related Documentation” on page 9
- “Typographic Conventions” on page 10
- “Symbol Conventions” on page 10
- “Default Paths and File Names” on page 11
- “Documentation, Support, and Training” on page 12
- “Searching Sun Product Documentation” on page 12
- “Third-Party Web Site References” on page 12
- “Sun Welcomes Your Comments” on page 13

Enterprise Server Documentation Set

The Enterprise Server documentation set describes deployment planning and system installation. The Uniform Resource Locator (URL) for Enterprise Server documentation is <http://docs.sun.com/coll/1343.9>. For an introduction to Enterprise Server, refer to the books in the order in which they are listed in the following table.

TABLE P-1 Books in the Enterprise Server Documentation Set

Book Title	Description
<i>Release Notes</i>	Provides late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of the supported hardware, operating system, Java Development Kit (JDK™), and database drivers.
<i>Quick Start Guide</i>	Explains how to get started with the Enterprise Server product.
<i>Installation Guide</i>	Explains how to install the software and its components.
<i>Upgrade Guide</i>	Explains how to upgrade to the latest version of Enterprise Server. This guide also describes differences between adjacent product releases and configuration options that can result in incompatibility with the product specifications.
<i>Administration Guide</i>	Explains how to configure, monitor, and manage Enterprise Server subsystems and components from the command line by using the <code>asadmin(1M)</code> utility. Instructions for performing these tasks from the Administration Console are provided in the Administration Console online help.
<i>Application Deployment Guide</i>	Explains how to assemble and deploy applications to the Enterprise Server and provides information about deployment descriptors.
<i>Your First Cup: An Introduction to the Java EE Platform</i>	Provides a short tutorial for beginning Java EE programmers that explains the entire process for developing a simple enterprise application. The sample application is a web application that consists of a component that is based on the Enterprise JavaBeans™ specification, a JAX-RS web service, and a JavaServer™ Faces component for the web front end.
<i>Application Development Guide</i>	Explains how to create and implement Java Platform, Enterprise Edition (Java EE platform) applications that are intended to run on the Enterprise Server. These applications follow the open Java standards model for Java EE components and APIs. This guide provides information about developer tools, security, and debugging.
<i>Add-On Component Development Guide</i>	Explains how to use published interfaces of Enterprise Server to develop add-on components for Enterprise Server. This document explains how to perform <i>only</i> those tasks that ensure that the add-on component is suitable for Enterprise Server.

TABLE P-1 Books in the Enterprise Server Documentation Set (Continued)

Book Title	Description
<i>Embedded Server Guide</i>	Explains how to run applications in embedded Enterprise Server and to develop applications in which Enterprise Server is embedded.
<i>Scripting Framework Guide</i>	Explains how to develop scripting applications in languages such as Ruby on Rails and Groovy on Grails for deployment to Enterprise Server.
<i>Troubleshooting Guide</i>	Describes common problems that you might encounter when using Enterprise Server and how to solve them.
<i>Error Message Reference</i>	Describes error messages that you might encounter when using Enterprise Server.
<i>Reference Manual</i>	Provides reference information in man page format for Enterprise Server administration commands, utility commands, and related concepts.
<i>Domain File Format Reference</i>	Describes the format of the Enterprise Server configuration file, <code>domain.xml</code> .
<i>Java EE 6 Tutorial, Volume I</i>	Explains how to use Java EE 6 platform technologies and APIs to develop Java EE applications.
<i>Message Queue Release Notes</i>	Describes new features, compatibility issues, and existing bugs for Sun GlassFish Message Queue.
<i>Message Queue Administration Guide</i>	Explains how to set up and manage a Sun GlassFish Message Queue messaging system.
<i>Message Queue Developer's Guide for JMX Clients</i>	Describes the application programming interface in Sun GlassFish Message Queue for programmatically configuring and monitoring Message Queue resources in conformance with the Java Management Extensions (JMX).
<i>System Virtualization Support in Sun Java System Products</i>	Summarizes Sun support for Sun Java System products when used in conjunction with system virtualization products and features.

Related Documentation

The Java EE 6 Tutorial, Volume II (https://www.sun.com/offers/details/java_ee6_tutorial.xml) contains all the topics in *Java EE 6 Tutorial, Volume I* and adds advanced topics, additional technologies, and case studies. The document is available to registered users of Enterprise Server.

Javadoc™ tool reference documentation for packages that are provided with Enterprise Server is available as follows:

- The API specification for version 6 of Java EE is located at <http://java.sun.com/javase/6/docs/api/>.
- API documentation for packages that are specific to the Enterprise Server product is located at: <http://javadoc.glassfish.org/v3/apidoc/>.

Additionally, the following resources might be useful:

- The Java EE Specifications (<http://java.sun.com/javase/technologies/index.jsp>)
- The Java EE Blueprints (<http://java.sun.com/reference/blueprints/index.html>)

For information about creating enterprise applications in the NetBeans™ Integrated Development Environment (IDE), see <http://www.netbeans.org/kb/60/index.html>.

For information about the Java DB for use with the Enterprise Server, see <http://developers.sun.com/javadb/>.

The sample applications demonstrate a broad range of Java EE technologies. The samples are bundled with the Java EE Software Development Kit (SDK).

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P-2 Typographic Conventions

Typeface	Meaning	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name% you have mail.</code>
AaBbCc123	What you type, contrasted with onscreen computer output	<code>machine_name% su</code> Password:
<i>AaBbCc123</i>	A placeholder to be replaced with a real name or value	The command to remove a file is <i>rm filename</i> .
<i>AaBbCc123</i>	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the <i>User's Guide</i> . A <i>cache</i> is a copy that is stored locally. Do <i>not</i> save the file.

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P-3 Symbol Conventions

Symbol	Description	Example	Meaning
[]	Contains optional arguments and command options.	ls [-l]	The -l option is not required.
{ }	Contains a set of choices for a required command option.	-d {y n}	The -d option requires that you use either the y argument or the n argument.
\${ }	Indicates a variable reference.	\${com.sun.javaRoot}	References the value of the com.sun.javaRoot variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

TABLE P-4 Default Paths and File Names

Placeholder	Description	Default Value
<i>as-install</i>	Represents the base installation directory for Enterprise Server. In configuration files, <i>as-install</i> is represented as follows: \${com.sun.aas.installRoot}	Installations on the Solaris™ operating system, Linux operating system, and Mac operating system: <i>user's-home-directory/glassfishv3/glassfish</i> Windows, all installations: <i>SystemDrive:\glassfishv3\glassfish</i>
<i>as-install-parent</i>	Represents the parent of the base installation directory for Enterprise Server.	Installations on the Solaris operating system, Linux operating system, and Mac operating system: <i>user's-home-directory/glassfishv3</i> Windows, all installations: <i>SystemDrive:\glassfishv3</i>
<i>domain-root-dir</i>	Represents the directory in which a domain is created by default.	<i>as-install/domains/</i>

TABLE P-4 Default Paths and File Names (Continued)

Placeholder	Description	Default Value
<i>domain-dir</i>	<p>Represents the directory in which a domain's configuration is stored.</p> <p>In configuration files, <i>domain-dir</i> is represented as follows:</p> <pre> \${com.sun.aas.instanceRoot} </pre>	<i>domain-root-dir/domain-name</i>

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- [Documentation \(http://www.sun.com/documentation/\)](http://www.sun.com/documentation/)
- [Support \(http://www.sun.com/support/\)](http://www.sun.com/support/)
- [Training \(http://www.sun.com/training/\)](http://www.sun.com/training/)

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.comSM web site, you can use a search engine by typing the following syntax in the search field:

```
search-term site:docs.sun.com
```

For example, to search for “broker,” type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use `sun.com` in place of `docs.sun.com` in the search field.

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to <http://docs.sun.com> and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 820-7727.

Introduction to the Development Environment for Enterprise Server Add-On Components

Sun GlassFish™ Enterprise Server enables an external vendor such as an independent software vendor (ISV), original equipment manufacturer (OEM), or system integrator to incorporate Enterprise Server into a new product with the vendor's own brand name. External vendors can extend the functionality of Enterprise Server by developing add-on components for Enterprise Server. Enterprise Server provides interfaces to enable add-on components to be configured, managed, and monitored through existing Enterprise Server tools such as the Administration Console and the `asadmin` utility.

The following topics are addressed here:

- [“Enterprise Server Modular Architecture and Add-On Components” on page 15](#)
- [“OSGi Alliance Module Management Subsystem” on page 16](#)
- [“Hundred-Kilobyte Kernel” on page 16](#)
- [“Overview of the Development Process for an Add-On Component” on page 17](#)

Enterprise Server Modular Architecture and Add-On Components

Enterprise Server has a modular architecture in which the features of Enterprise Server are provided by a consistent set of components that interact with each other. Each component provides a small set of functionally related features.

The modular architecture of Enterprise Server enables users to download and install only the components that are required for the applications that are being deployed. As a result, start-up times, memory consumption, and disk space requirements are all minimized.

The modular architecture of Enterprise Server enables you to extend the basic functionality of Enterprise Server by developing add-on components. An *add-on component* is an encapsulated definition of reusable code that has the following characteristics:

- The component provides a set of Java classes.

- The component offers services and public interfaces.
- The component implements the public interfaces with a set of private classes.
- The component depends on other components.

Add-on components that you develop interact with Enterprise Server in the same way as components that are supplied in Enterprise Server distributions.

You can create and offer new or updated add-on components at any time. Enterprise Server administrators can install add-on components and update or remove installed components after Enterprise Server is installed. For more information, see [Chapter 10, “Extending Enterprise Server,”](#) in *Sun GlassFish Enterprise Server v3 Administration Guide*.

OSGi Alliance Module Management Subsystem

To enable components to be added when required, Enterprise Server provides a lightweight and extensible kernel that uses the module management subsystem from the [OSGi Alliance](#). Any Enterprise Server component that plugs in to this kernel must be implemented as an OSGi bundle. To enable an add-on component to plug in to the Enterprise Server kernel in the same way as other components, package the component as an OSGi bundle. For more information, see [“Packaging an Add-On Component”](#) on page 71.

The default OSGi module management subsystem in Enterprise Server is the [Apache Felix OSGi framework](#). However, the Enterprise Server kernel uses only the [OSGi Service Platform Release 4 API](#). Therefore, Enterprise Server supports other OSGi module management subsystems that are compatible with the OSGi Service Platform Release 4 API.

Hundred-Kilobyte Kernel

The [Hundred-Kilobyte Kernel \(HK2\)](#) is the lightweight and extensible kernel of Enterprise Server. HK2 consists of the following technologies:

- **Module subsystem.** The HK2 module subsystem provides isolation between components of the Enterprise Server. The HK2 module subsystem is compatible with existing technologies such as the OSGi framework.
- **Component model.** The HK2 component model eases the development of components that are also services. Enterprise Server discovers these components automatically and dynamically. HK2 components use injection of dependencies to express dependencies on other components. Enterprise Server provides two-way mappings between the services of an HK2 component and OSGi services.

For more information, see [Chapter 2, “Writing HK2 Components.”](#)

Overview of the Development Process for an Add-On Component

To ensure that an add-on component behaves identically to components that are supplied in Enterprise Server distributions, the component must meet the following requirements:

- If the component generates management data or monitoring data, it must provide that data to other Enterprise Server components in the same way as other Enterprise Server components.
- If the component generates management data or monitoring data, it must provide that data to users through Enterprise Server administrative interfaces such as Administration Console and the `asadmin` utility.
- The component must be packaged and delivered as an OSGi bundle.

To develop add-on components that meet these requirements, follow the development process that is described in the following sections:

- [“Writing HK2 Components” on page 17](#)
- [“Extending the Administration Console” on page 17](#)
- [“Extending the `asadmin` Utility” on page 18](#)
- [“Adding Container Capabilities” on page 18](#)
- [“Packaging and Delivering an Add-On Component” on page 18](#)

Writing HK2 Components

The Hundred-Kilobyte Kernel (HK2) is the lightweight and extensible kernel of Enterprise Server. To interact with Enterprise Server, add-on components plug in to this kernel. In the HK2 component model, the functions of an add-on component are declared through a contract-service implementation paradigm. An *HK2 contract* identifies and describes the building blocks or the extension points of an application. An *HK2 service* implements an HK2 contract.

For more information, see [Chapter 2, “Writing HK2 Components.”](#)

Extending the Administration Console

The Administration Console is a browser-based tool for administering Enterprise Server. It features an easy-to-navigate interface and online help. Extending the Administration Console enables you to provide a graphical user interface for administering your add-on component. You can use any of the user interface features of the Administration Console, such as tree nodes, links on the Common Tasks page, tabs and sub-tabs, property sheets, and JavaServer™ Faces pages. Your add-on component implements a marker interface and provides a configuration file that describes how your customizations integrate with the Administration Console.

For more information, see [Chapter 3, “Extending the Administration Console.”](#)

Extending the `asadmin` Utility

The `asadmin` utility is a command-line tool for configuring and administering Enterprise Server. Extending the `asadmin` utility enables you to provide administrative interfaces for an add-on component that are consistent with the interfaces of other Enterprise Server components. A user can run `asadmin` commands either from a command prompt or from a script. For more information about the `asadmin` utility, see the `asadmin(1M)` man page.

For more information, see [Chapter 4, “Extending the `asadmin` Utility.”](#)

Adding Container Capabilities

Applications run on Enterprise Server in containers. Enterprise Server enables you to create containers that extend or replace the existing containers of Enterprise Server. Adding container capabilities enables you to run new types of applications and to deploy new archive types in Enterprise Server.

For more information, see [Chapter 5, “Adding Container Capabilities.”](#)

Packaging and Delivering an Add-On Component

Packaging an add-on component enables the component to interact with the Enterprise Server kernel in the same way as other components. Integrating a component with Enterprise Server enables Enterprise Server to discover the component at runtime. If an add-on component is an extension or update to existing installations of Enterprise Server, deliver the component through Update Tool.

For more information, see [Chapter 6, “Packaging, Integrating, and Delivering an Add-On Component.”](#)

Writing HK2 Components

The Hundred-Kilobyte Kernel (HK2) is the lightweight and extensible kernel of Enterprise Server. To interact with Enterprise Server, add-on components plug in to this kernel. In the HK2 component model, the functions of an add-on component are declared through a contract-service implementation paradigm. An *HK2 contract* identifies and describes the building blocks or the extension points of an application. An *HK2 service* implements an HK2 contract.

The following topics are addressed here:

- “HK2 Component Model” on page 19
- “Services in the HK2 Component Model” on page 20
- “HK2 Runtime” on page 20
- “Inversion of Control” on page 22
- “Identifying a Class as an Add-On Component” on page 24
- “Using the Apache Maven Build System to Develop HK2 Components” on page 25

HK2 Component Model

The Hundred-Kilobyte Kernel (HK2) provides a module system and component model for building complex software systems. HK2 forms the core of Enterprise Server's architecture.

The module system is responsible for instantiating classes that constitute the application functionality. The HK2 runtime complements the module system by creating objects. It configures such objects by:

- Injecting other objects that are needed by a newly instantiated object
- Injecting configuration information needed for that object
- Making newly created objects available, so that they can then be injected to other objects that need it

Services in the HK2 Component Model

An HK2 service identifies the building blocks or the extension points of an application. A service is a plain-old Java object (POJO) with the following characteristics:

- The object implements an interface.
- The object is declared in a JAR file with the `META-INF/services` file.

To clearly separate the contract interface and its implementation, the HK2 runtime requires the following information:

- Which interfaces are contracts
- Which implementations of such interfaces are services

Interfaces that define a contract are identified by the `org.jvnet.hk2.annotation.Contract` annotation.

```
@Retention(RUNTIME)
@Target(TYPE)
public @interface Contract {
}
```

Implementations of such contracts should be identified with an `org.jvnet.hk2.annotations.Service` annotation so that the HK2 runtime can recognize them as `@Contract` implementations.

```
@Retention(RUNTIME)
@Target(TYPE)
public @interface Service {
    ...
}
```

For more information, see [Service](#).

HK2 Runtime

Once Services are defined, the HK2 runtime can be used to instantiate or retrieve instances of services. Each service instance has a scope, specified as singleton, per thread, per application, or a custom scope.

Scopes of Services

You can specify the scope of a service by adding an `org.jvnet.hk2.annotations.Scoped` annotation to the class-level of your `@Service` implementation class. Scopes are also services, so they can be custom defined and added to the HK2 runtime before being used by other services.

Each scope is responsible for storing the service instances to which it is tied; therefore, the HK2 runtime does not rely on predefined scopes (although it comes with a few predefined ones).

```
@Contract
public abstract class Scope {
    public abstract ScopeInstance current();
}
```

The following code fragment shows how to set the scope for a service to the predefined Singleton scope:

```
@Service
public Singleton implements Scope {
    ...
}

@Scope(Singleton.class)
@Service
public class SingletonService implements RandomContract {
    ...
}
```

You can define a new Scope implementation and use that scope on your @Service implementations. You will see that the HK2 runtime uses the Scope instance to store and retrieve service instances tied to that scope.

Instantiation of Components in HK2

Do not call the new method to instantiate components. Instead, retrieve components by using the ComponentManager instance. The simplest way to use the ComponentManager instance is through a `getComponent(Class<T> clazz)` call:

```
public <T> T getComponent(Class<T> clazz) throws ComponentException;
```

More APIs are available at [ComponentManager](#).

HK2 Lifecycle Interfaces

Components can attach behaviors to their construction and destruction events by implementing the `org.jvnet.hk2.component.PostConstruct` interface, the `org.jvnet.hk2.component.PreDestroy` interface, or both. These are interfaces rather than annotations for performance reasons.

The `PostConstruct` interface defines a single method, `postConstruct`, which is called after a component has been initialized and all its dependencies have been injected.

The `PreDestroy` interface defines a single method, `preDestroy`, which is called just before a component is removed from the system.

EXAMPLE 2-1 Example Implementation of `PostConstruct` and `PreDestroy`

```
@Service(name="com.example.container.MyContainer")
public class MyContainer implements Container, PostConstruct, PreDestroy {
    @Inject
    Logger logger;
    ...
    public void postConstruct() {
        logger.info("Starting up.");
    }

    public void preDestroy() {
        logger.info("Shutting down.");
    }
}
```

Inversion of Control

Inversion of control (IoC) refers to a style of software architecture where the behavior of a system is determined by the runtime capabilities of the individual, discrete components that make up the system. This architecture is different from traditional styles of software architecture, where all the components of a system are specified at design-time. With IoC, discrete components respond to high-level events to perform actions. While performing these actions, the components typically rely on other components to provide other actions. In an IoC system, components use injection to gain access to other components, and extraction to make component variables available to the system.

Injecting HK2 Components

Services usually rely on other services to perform their tasks. The HK2 runtime identifies the `Contract` implementations required by a service by using the org.jvnet.hk2.annotations.Inject annotation. `Inject` can be placed on fields or setter methods of any service instantiated by the HK2 runtime. The target service is retrieved and injected during the calling service's instantiation by the component manager.

The following example shows how to use `@Inject` at the field level:

```
@Inject
ConfigService config;
```

The following example shows how to use `@Inject` at the setter level:

```
@Inject
public void set(ConfigService svc) {...}
```

Injection can further qualify the intended injected service implementation by using a name and scope from which the service should be available:

```
@Inject(Scope=Singleton.class, name="deploy")
AdminCommand deployCommand;
```

Extraction

Although all services are automatically placed into a scope for later retrieval, a component may need to extract more than itself. One practical way of doing so is to use a factory service. For simplicity, however, the HK2 runtime extracts all fields or getter methods annotated with the [org.jvnet.hk2.annotations.Extract](#) annotation.

The following example shows how to use `@Extract` at the field level:

```
@Extract
ConfigService config;
```

The following example shows how to use `@Extract` at the getter level:

```
@Extract
public ConfigService getConfigService() {...}
```

Extraction, like injection, can also use the name and scope annotation fields to further qualify the extracted `Contract` implementation.

Extracted fields and properties are made available to other service instances by exporting them to the `org.jvnet.hk2.component.Habitat` instance. `Habitat` instances can be injected into other components, and the components can then extract and use the data contained in the `Habitat` instance.

```
@Inject
protected Habitat habitat;
...
public void doSomething(String name) {
    ...
    ConfigService config = habitat.getComponent(ConfigService.class);
    ...
}
```

Instantiation Cascading in HK2

Injection of instances that have not been already instantiated triggers more instantiation. You can see this as a component instantiation cascade where some code requests for a high-level service will, by using the `@Inject` annotation, require more injection and instantiation of lower level services. This cascading feature keeps the implementation as private as possible while relying on interfaces and the separation of contracts and providers.

EXAMPLE 2-2 Example of Instantiation Cascading

The following example shows how the instantiation of `DeploymentService` as a `Startup` contract implementation will trigger the instantiation of the `ConfigService`.

```
@Contract
public interface Startup {...}

Iterable<Startup> startups;
startups = componentMgr.getComponents(Startup.class);

@Service
public class DeploymentService implements Startup {
    @Inject
    ConfigService config;
}

@Service
public class ConfigService implements ... {...}
```

Identifying a Class as an Add-On Component

Enterprise Server discovers add-on components by identifying Java programming language classes that are annotated with the `org.jvnet.hk2.annotation.Service` annotation.

To identify a class as an implementation of an Enterprise Server service, add the `org.jvnet.hk2.annotation.Service` annotation at the class-definition level of your Java programming language class.

```
@Service
public class SamplePlugin implements ConsoleProvider {
    ...
}
```

The `@Service` annotation has the following elements. All elements are optional.

name

The name of the service. The default value is an empty string.

scope

The scope to which this service implementation is tied. The default value is `org.jvnet.hk2.component.PerLookup.class`.

factory

The factory class for the service implementation, if the service is created by a factory class rather than by calling the default constructor. If this element is specified, the factory component is activated, and `Factory.getObject` is used instead of the default constructor. The default value of the factory element is `org.jvnet.hk2.component.Factory.class`.

EXAMPLE 2-3 Example of the Optional Elements of the `@Service` Annotation

The following example shows how to use the optional elements of the `@Service` annotation:

```
@Service (name="MyService",
         scope=com.example.PerRequest.class,
         factory=com.example.MyCustomFactory)
public class SamplePlugin implements ConsoleProvider {
    ...
}
```

Using the Apache Maven Build System to Develop HK2 Components

If you are using Maven 2 to build HK2 components, invoke the auto-depends plug-in for Maven so that the `META-INF/services` files are generated automatically during build time.

EXAMPLE 2-4 Example of the Maven Plug-In Configuration

```
<plugin>
  <groupId>com.sun.enterprise</groupId>
  <artifactId>hk2-maven-plugin</artifactId>
  <configuration>
    <includes>
      <include>com/sun/enterprise/v3/**</include>
    </includes>
  </configuration>
</plugin>
```

EXAMPLE 2-5 Example of `META-INF/services` File Generation

This example shows how to use `@Contract` and `@Service` and the resulting `META-INF/services` files.

The interfaces and classes in this example are as follows:

EXAMPLE 2-5 Example of META-INF/services File Generation (Continued)

```
package com.sun.v3.annotations;
@Contract
public interface Startup {...}

package com.wombat;
@Contract
public interface RandomContract {...}

package com.sun.v3;
@Service
public class MyService implements Startup, RandomContract, PropertyChangeListener {
    ...
}
```

These interfaces and classes generate this META-INF/services file with the MyService content:

```
com.sun.v3.annotations.Startup
com.wombat.RandomContract
```

Extending the Administration Console

The Administration Console is a browser-based tool for administering Enterprise Server. It features an easy-to-navigate interface and online help. Extending the Administration Console enables you to provide a graphical user interface for administering your add-on component. You can use any of the user interface features of the Administration Console, such as tree nodes, links on the Common Tasks page, tabs and sub-tabs, property sheets, and JavaServer™ Faces pages. Your add-on component implements a marker interface and provides a configuration file that describes how your customizations integrate with the Administration Console.

This chapter refers to a simple example called `console-sample-ip` that illustrates how to provide Administration Console features for a hypothetical add-on component. Instructions for obtaining and using this example are available at the [example's project page](http://wiki.glassfish.java.net/Wiki.jsp?page=V3SampleIpProject) (<http://wiki.glassfish.java.net/Wiki.jsp?page=V3SampleIpProject>). When you check out the code, it is placed in a directory named `glassfish-samples/v3/plugin/adminconsole/console-sample-ip/` in your current directory. In this chapter, path names for the example files are relative to this directory.

The following topics are addressed here:

- “Administration Console Architecture” on page 28
- “About Administration Console Templates” on page 29
- “About Integration Points” on page 30
- “Specifying the ID of an Add-On Component” on page 30
- “Adding Functionality to the Administration Console” on page 31
- “Adding Internationalization Support” on page 43
- “Changing the Theme or Brand of the Administration Console” on page 43
- “Creating an Integration Point Type” on page 45

Administration Console Architecture

The Administration Console is a web application that is composed of OSGi bundles. These bundles provide all the features of the Administration Console, such as the Web Applications, Update Center, and Security content. To provide support for your add-on component, create your own OSGi bundle that implements the parts of the user interface that you need. Place your bundle in the `modules` directory of your Enterprise Server installation, along with the other Administration Console bundles.

To learn how to package the Administration Console features for an add-on component, go to the `modules` directory of your Enterprise Server installation and examine the contents of the files named `console-componentname-plugin.jar`. Place the `console-sample-ip` project bundle in the same place to deploy it and examine the changes that it makes to the Administration Console.

The Administration Console includes a Console Add-On Component Service. The Console Add-On Component Service is an HK2 service that acts as a façade to all the Administration Console add-on components. The Console Add-On Component Service queries the various console providers for integration points so that it can perform the actions needed for the integration (adding a tree node or a new tab, for example). The interface name for this service is `org.glassfish.api.admingui.ConsolePluginService`.

For details about the Hundred-Kilobyte Kernel (HK2) project, see [“Hundred-Kilobyte Kernel” on page 16](#) and [“HK2 Component Model” on page 19](#).

Each add-on component must contain a console provider implementation. This is a Java class that implements the `org.glassfish.api.admingui.ConsoleProvider` interface and uses the `HK2@Service` annotation. The console provider allows your add-on component to specify where your integration point configuration file is located. This configuration file communicates to the Console Add-On Component Service the customizations that your add-on component makes to the Administration Console.

Implementing a Console Provider

The `org.glassfish.api.admingui.ConsoleProvider` interface has one required method, `getConfiguration`. The `getConfiguration` method returns the location of the `console-config.xml` file as a `java.net.URL`. If `getConfiguration` returns `null`, the default location, `META-INF/admingui/console-config.xml`, is used. The `console-config.xml` file is described in [“About Integration Points” on page 30](#).

To implement the console provider for your add-on component, write a Java class that is similar to the following example.

EXAMPLE 3-1 Example ConsoleProvider Implementation

This example shows a simple implementation of the `ConsoleProvider` interface:

EXAMPLE 3-1 Example ConsoleProvider Implementation (Continued)

```
package org.glassfish.admingui.plugin;

import org.glassfish.api.admingui.ConsoleProvider;
import org.jvnet.hk2.annotations.Service;

import java.net.URL;

@Service
public class SamplePlugin implements ConsoleProvider {

    public URL getConfiguration() { return null; }
}
```

This implementation of `getConfiguration` returns `null` to specify that the configuration file is in the default location. If you place the file in a nonstandard location or give it a name other than `console-config.xml`, your implementation of `getConfiguration` must return the URL where the file can be found.

You can find this example code in the file `project/src/main/java/org/glassfish/admingui/plugin/SamplePlugin.java`.

About Administration Console Templates

Enterprise Server includes a set of templates that make it easier to create JavaServer Faces pages for your add-on component. These templates use [Templating for JavaServer Faces Technology](https://jsftemplating.dev.java.net/) (<https://jsftemplating.dev.java.net/>), which is also known as JSFTemplating.

Examples of JSFTemplating technology can be found in the following sections of this chapter:

- “Creating a JavaServer Faces Page for Your Node” on page 33
- “Creating JavaServer Faces Pages for Your Tabs” on page 36
- “Creating a JavaServer Faces Page for Your Task” on page 38
- “Creating a JavaServer Faces Page for Your Task Group” on page 39
- “Creating a JavaServer Faces Page for Your Page Content” on page 41
- “Adding a Page to the Administration Console” on page 42

About Integration Points

The integration points for your add-on component are the individual Administration Console user interface features that your add-on component will extend. You can implement the following kinds of integration points:

- Nodes in the navigation tree
- Elements on the Common Tasks page of the Administration Console
- JavaServer Faces pages
- Tabs and sub-tabs

Specify all the integration points in a file named `console-config.xml`. In the example, this file is in the directory `project/src/main/resources/META-INF/adingui/`. The following sections describe how to create this file.

In addition, create JavaServer Faces pages that contain JSF code fragments to implement the integration points. In the example, these files are in the directory `project/src/main/resources/`. The content of these files depends on the integration point you are implementing. The following sections describe how to create these JavaServer Faces pages.

For reference information on integration points, see [Appendix A, “Integration Point Reference.”](#)

Specifying the ID of an Add-On Component

The `console-config.xml` file consists of a `console-config` element that encloses a series of integration-point elements. The `console-config` element has one attribute, `id`, which specifies a unique name or ID value for the add-on component.

In the example, the element is declared as follows:

```
<console-config id="sample">
    ...
</console-config>
```

You will also specify this ID value when you construct URLs to images, resources and pages in your add-on component. See [“Adding a Node to the Navigation Tree” on page 32](#) for an example.

For example, a URL to an image named `my.gif` might look like this:

```
<sun:image url="/resource/sample/images/my.gif" />
```

The URL is constructed as follows:

- `/resource` is required to locate any resource URL.
- `sample` is the add-on component ID. You must choose a unique ID value.
- `images` is a folder under the root of the add-on component JAR file.

Adding Functionality to the Administration Console

The `integration-point` elements in the `console-config.xml` file specify attributes for the user interface features that you choose to implement. The example file provides examples of most of the available kinds of integration points at this release. Your own add-on component can use some or all of them.

For each `integration-point` element, specify the following attributes.

`id`

An identifier for the integration point.

`parentId`

The ID of the integration point's parent.

`type`

The type of the integration point.

`priority`

A numeric value that specifies the relative ordering of integration points for add-on components that specify the same `parentId`. A lower number specifies a higher priority (for example, 100 represents a higher priority than 400). The integration points for add-on components are always placed after those in the basic Administration Console. You might need to experiment to place the integration point where you want it. This attribute is optional.

`content`

The content for the integration point, typically a JavaServer Faces page. In the example, you can find the JavaServer Faces pages in the directory `project/src/main/resources/`.

Note – The order in which these attributes are specified does not matter, and in the example `console-config.xml` file the order varies. To improve readability, this chapter uses the same order throughout.

The following topics are addressed here:

- “Adding a Node to the Navigation Tree” on page 32
- “Adding Tabs to a Page” on page 34
- “Adding a Task to the Common Tasks Page” on page 37
- “Adding a Task Group to the Common Tasks Page” on page 38

- [“Adding Content to a Page” on page 40](#)
- [“Adding a Page to the Administration Console” on page 42](#)

Adding a Node to the Navigation Tree

You can add a node to the navigation tree, either at the top level or under another node. To add a node, use an integration point of type `org.glassfish.admingui:navNode`. Use the `parentId` attribute to specify where the new node should be placed. Any tree node, including those added by other add-on components, can be specified. Examples include the following:

`tree`

At the top level

`applicationServer`

Under the Enterprise Server node

`applications`

Under the Applications node

`resources`

Under the Resources node

`configuration`

Under the Configuration node

`webContainer`

Under the Web Container node

`httpService`

Under the HTTP Service node

Note – The `webContainer` and `httpService` nodes are available only if you installed the web container module for the Administration Console (the `console-web-gui.jar` OSGi bundle).

If you do not specify a `parentId`, the new content is added to the root of the integration point, in this case the top level node, `tree`.

EXAMPLE 3-2 Example Tree Node Integration Point

For example, the following `integration-point` element uses a `parentId` of `tree` to place the new node at the top level.

```
<integration-point
  id="sampleNode"
  parentId="tree"
  type="org.glassfish.admingui:treeNode">
```

EXAMPLE 3-2 Example Tree Node Integration Point (Continued)

```

        priority="200"
        content="sampleNode.jsf"
    />

```

This example specifies the following values in addition to the `parentId`:

- The `id` value, `sampleNode`, specifies the integration point ID.
- The `type` value, `org.glassfish.admingui:treeNode`, specifies the integration point type as a tree node.
- The `priority` value, `200`, specifies the order of the node on the tree.
- The `content` value, `sampleNode.jsf`, specifies the JavaServer Faces page that displays the node.

The example `console-config.xml` file provides other examples of tree nodes under the Resources and Configuration nodes.

Creating a JavaServer Faces Page for Your Node

A JavaServer Faces page for a tree node uses the tag `sun:treeNode`. This tag provides all the capabilities of the Project Woodstock tag `webui.jsf:treeNode`.

EXAMPLE 3-3 Example JavaServer Faces Page for a Tree Node

In the example, the `sampleNode.jsf` file has the following content:

```

<sun:treeNode
    id="treeNode1"
    text="SampleTop"
    url="/sample/page/testPage.jsf?name=SampleTop"
    imageURL="/resource/sample/images/sample.png"
>
    <sun:treeNode
        id="treeNodeBB"
        text="SampleBB"
        url="/sample/page/testPage.jsf?name=SampleBB"
        imageURL="resource/sample/images/sample.png" />
</sun:treeNode>

```

This file uses the `sun:treeNode` tag to specify both a top-level tree node and another node nested beneath it. In your own JavaServer Faces pages, specify the attributes of this tag as follows:

id

A unique identifier for the tree node.

text

The node name that appears in the tree.

url

The location of the JavaServer Faces page that appears when you click the node. In the example, most of the integration points use a very simple JavaServer Faces page called `testPage.jsf`, which is in the `src/main/resources/page/` directory. Specify the integration point `id` value as the root of the URL; in this case, it is `sample` (see [“Specifying the ID of an Add-On Component” on page 30](#)). The rest of the URL is relative to the `src/main/resources/` directory, where `sampleNode.jsf` resides.

The `url` tag in this example passes a `name` parameter to the JavaServer Faces page.

imageUrl

The location of a graphic to display next to the node name. In the example, the graphic is always `sample.png`, which is in the `src/main/resources/images/` directory. The URL for this image is an absolute path, `/resource/sample/images/sample.png`, where `sample` in the path is the integration point `id` value (see [“Specifying the ID of an Add-On Component” on page 30](#)).

Adding Tabs to a Page

You can add a tab to an existing tab set, or you can create a tab set for your own page. One way to add a tab or tab set is to use an integration point of type `org.glassfish.admingui:serverInstTab`, which adds a tab to the tab set on the main Enterprise Server page of the Administration Console. You can also create sub-tabs. Once again, the `parentId` element specifies where to place the tab or tab set.

EXAMPLE 3-4 Example Tab Integration Point

In the example, the following integration-point element adds a new tab on the main Enterprise Server page of the Administration Console:

```
<integration-point
  id="sampleTab"
  parentId="serverInstTabs"
  type="org.glassfish.admingui:serverInstTab"
  priority="500"
  content="sampleTab.jsf"
/>
```

This example specifies the following values:

- The `id` value, `sampleTab`, specifies the integration point ID.
- The `parentId` value, `serverInstTabs`, specifies the tab set associated with the server instance. The Enterprise Server page is the only one of the default Administration Console pages that has a tab set.
- The `type` value, `org.glassfish.admingui:serverInstTab`, specifies the integration point type as a tab associated with the server instance.
- The `priority` value, `500`, specifies the order of the tab within the tab set. This value is optional.
- The `content` value, `sampleTab.jsf`, specifies the JavaServer Faces page that displays the tab.

EXAMPLE 3-5 Example Tab Set Integration Points

The following integration-point elements add a new tab with two sub-tabs, also on the main Enterprise Server page of the Administration Console:

```
<integration-point
  id="sampleTabWithSubTab"
  parentId="serverInstTabs"
  type="org.glassfish.admingui:serverInstTab"
  priority="300"
  content="sampleTabWithSubTab.jsf"
/>

<integration-point
  id="sampleSubTab1"
  parentId="sampleTabWithSubTab"
  type="org.glassfish.admingui:serverInstTab"
  priority="300"
  content="sampleSubTab1.jsf"
/>

<integration-point
  id="sampleSubTab2"
  parentId="sampleTabWithSubTab"
  type="org.glassfish.admingui:serverInstTab"
  priority="400"
  content="sampleSubTab2.jsf"
/>
```

These examples specify the following values:

- The `id` values, `sampleTabWithSubTab`, `sampleSubTab1`, and `sampleSubTab2`, specify the integration point IDs for the tab and its sub-tabs.
- The `parentId` of the new tab, `serverInstTabs`, specifies the tab set associated with the server instance. The `parentId` of the two sub-tabs, `sampleTabWithSubTab`, is the `id` value of this new tab.
- The `type` value, `org.glassfish.admingui:serverInstTab`, specifies the integration point type for all the tabs as a tab associated with the server instance.
- The `priority` values specify the order of the tabs within the tab set. This value is optional. In this case, the `priority` value for `sampleTabWithSubTab` is `300`, which is higher than the value for `sampleTab`. That means that `sampleTabWithSubTab` appears to the left of `sampleTab` in the Administration Console. The `priority` values for `sampleSubTab1` and `sampleSubTab2` are `300` and `400` respectively, so `sampleSubTab1` appears to the left of `sampleSubTab2`.
- The `content` values, `sampleTabWithSubTab.jsf`, `sampleSubTab1.jsf`, and `sampleSubTab2.jsf`, specify the JavaServer Faces pages that display the tabs.

Creating JavaServer Faces Pages for Your Tabs

A JavaServer Faces page for a tab uses the tag `sun:tab`. This tag provides all the capabilities of the Project Woodstock tag `webuijsf:tab`.

EXAMPLE 3-6 Example JavaServer Faces Page for a Tab

In the example, the `sampleTab.jsf` file has the following content:

```
<sun:tab id="sampleTab" immediate="true" text="Sample First Tab">
  <!--command
    setSessionAttribute(key="serverInstTabs" value="sampleTab");
    gf.redirect(page="#{request.contextPath}/page/tabPage.jsf?name=Sample%20First%20Tab");
  /-->
</sun:tab>
```

Note – In the actual file there are no line breaks in the `gf.redirect` value.

In your own JavaServer Faces pages, specify the attributes of this tag as follows:

id

A unique identifier for the tab, in this case `sampleTab`.

immediate

If set to `true`, event handling for this component should be handled immediately (in the Apply Request Values phase) rather than waiting until the Invoke Application phase.

text

The tab name that appears in the tab set.

The JSF page displays tab content differently from the way the page for a node displays node content. It invokes two handlers for the command event: `setSessionAttribute` and `gf.redirect`. The `gf.redirect` handler has the same effect for a tab that the `url` attribute has for a node. It navigates to a simple JavaServer Faces page called `tabPage.jsf`, in the `src/main/resources/page/` directory, passing the text “Sample First Tab” to the page in a name parameter.

The `sampleSubTab1.jsf` and `sampleSubTab2.jsf` files are almost identical to `sampleTab.jsf`. The most important difference is that each sets the session attribute `serverInstTabs` to the base name of the JavaServer Faces file that corresponds to that tab:

```
setSessionAttribute(key="serverInstTabs" value="sampleTab");

setSessionAttribute(key="serverInstTabs" value="sampleSubTab1");

setSessionAttribute(key="serverInstTabs" value="sampleSubTab2");
```

Adding a Task to the Common Tasks Page

You can add either a single task or a group of tasks to the Common Tasks page of the Administration Console. To add a task or task group, use an integration point of type `org.glassfish.admingui:commonTask`.

See “[Adding a Task Group to the Common Tasks Page](#)” on page 38 for information on adding a task group.

EXAMPLE 3-7 Example Task Integration Point

In the example `console-config.xml` file, the following `integration-point` element adds a task to the Deployment task group:

```
<integration-point
    id="sampleCommonTask"
    parentId="deployment"
    type="org.glassfish.admingui:commonTask"
    priority="200"
    content="sampleCommonTask.jsf"
/>
```

This example specifies the following values:

- The `id` value, `sampleCommonTask`, specifies the integration point ID.
- The `parentId` value, `deployment`, specifies that the task is to be placed in the Deployment task group.
- The `type` value, `org.glassfish.admingui:commonTask`, specifies the integration point type as a common task.
- The `priority` value, `200`, specifies the order of the task within the task group.
- The `content` value, `sampleCommonTask.jsf`, specifies the JavaServer Faces page that displays the task.

Creating a JavaServer Faces Page for Your Task

A JavaServer Faces page for a task uses the tag `sun:commonTask`. This tag provides all the capabilities of the Project Woodstock tag `webui.jsf:commonTask`.

EXAMPLE 3-8 Example JavaServer Faces Page for a Task

In the example, the `sampleCommonTask.jsf` file has the following content:

```
<sun:commonTask
  text="Sample Application Page"
  tooltip="Sample Application Page"
  onClick="return admingui.woodstock.commonTaskHandler('treeForm:tree:applications:ejb',
    '#{request.contextPath}/sample/page/testPage.jsf?name=Sample%20Application%20Page');">
</sun:commonTask>
```

Note – In the actual file, there is no line break in the `onClick` attribute value.

This file uses the `sun:commonTask` tag to specify the task. In your own JavaServer Faces pages, specify the attributes of this tag as follows:

- | | |
|----------------------|---|
| <code>text</code> | The task name that appears on the Common Tasks page. |
| <code>tooltip</code> | The text that appears when a user places the mouse cursor over the task name. |
| <code>onClick</code> | Scripting code that is to be executed when a user clicks the task name. |

Adding a Task Group to the Common Tasks Page

You can add a new group of tasks to the Common Tasks page to display the most important tasks for your add-on component. To add a task group, use an integration point of type `org.glassfish.admingui:commonTask`.

EXAMPLE 3-9 Example Task Group Integration Point

In the example `console-config.xml` file, the following `integration-point` element adds a new task group to the Common Tasks page:

```
<integration-point
    id="sampleGroup"
    parentId="commonTasksSection"
    type="org.glassfish.admingui:commonTask"
    priority="500"
    content="sampleTaskGroup.jsf"
/>
```

This example specifies the following values:

- The `id` value, `sampleGroup`, specifies the integration point ID.
- The `parentId` value, `commonTasksSection`, specifies that the task group is to be placed on the Common Tasks page.
- The `type` value, `org.glassfish.admingui:commonTask`, specifies the integration point type as a common task.
- The `priority` value, `500`, specifies the order of the task group on the Common Tasks page. The low value places it at the end of the page.
- The `content` value, `sampleTaskGroup.jsf`, specifies the JavaServer Faces page that displays the task.

Creating a JavaServer Faces Page for Your Task Group

A JavaServer Faces page for a task group uses the tag `sun:commonTasksGroup`. This tag provides all the capabilities of the Project Woodstock tag `webui.jsf:commonTasksGroup`.

EXAMPLE 3-10 Example JavaServer Faces Page for a Task Group

In the example, the `sampleTaskGroup.jsf` file has the following content:

```
<sun:commonTasksGroup title="My Own Sample Group">
  <sun:commonTask
    text="Go To Sample Resource"
    tooltip="Go To Sample Resource"
    onClick="return admingui.woodstock.commonTaskHandler('form:tree:resources:treeNode1',
      '#{request.contextPath}/sample/page/testPage.jsf?name=Sample%20Resource%20Page');">
  </sun:commonTask>
  <sun:commonTask
    text="Sample Configuration"
    tooltip="Go To Sample Configuration"
    onClick="return admingui.woodstock.commonTaskHandler('form:tree:configuration:sampleConfigNode',
```

EXAMPLE 3-10 Example JavaServer Faces Page for a Task Group (Continued)

```
'#{request.contextPath}/sample/page/testPage.jsf?name=Sample%20Configuration%20Page');">
</sun:commonTask>
</sun:commonTasksGroup>
```

Note – In the actual file, there are no line breaks in the `onClick` attribute values.

This file uses the `sun:commonTasksGroup` tag to specify the task group, and two `sun:commonTask` tags to specify the tasks in the task group. The `sun:commonTasksGroup` tag has only one attribute, `title`, which specifies the name of the task group.

Adding Content to a Page

You can add content for your add-on component to an existing top-level page, such as the Configuration page or the Resources page. To add content to one of these pages, use an integration point of type `org.glassfish.admingui:configuration` or `org.glassfish.admingui:resources`.

EXAMPLE 3-11 Example Resources Page Implementation Point

In the example `console-config.xml` file, the following `integration-point` element adds new content to the top-level Resources page:

```
<integration-point
    id="sampleResourceLink"
    parentId="propSheetSection"
    type="org.glassfish.admingui:resources"
    priority="100"
    content="sampleResourceLink.jsf"
/>
```

This example specifies the following values:

- The `id` value, `sampleResourceLink`, specifies the integration point ID.
- The `parentId` value, `propSheetSection`, specifies that the content is to be a section of a property sheet on the page.
- The `type` value, `org.glassfish.admingui:resources`, specifies the integration point type as the Resources page.

To add content to the Configuration page, specify the type value as `org.glassfish.admingui:configuration`.

- The priority value, `100`, specifies the order of the content on the Resources page. The high value places it at the top of the page.
- The content value, `sampleResourceLink.jsf`, specifies the JavaServer Faces page that displays the new content on the Resources page.

Another integration-point element in the `console-config.xml` file places similar content on the Configuration page.

Creating a JavaServer Faces Page for Your Page Content

A JavaServer Faces page for page content often uses the tag `sun:property` to specify a property on a property sheet. This tag provides all the capabilities of the Project Woodstock tag `webuijsf:property`.

EXAMPLE 3-12 Example JavaServer Faces Page for a Resource Page Item

In the example, the `sampleResourceLink.jsf` file has the following content:

```
<sun:property>
  <sun:hyperlink
    tooltip="Sample Resource"
    url="/sample/page/testPage.jsf?name=Sample%20Resource%20Page" >
    <sun:image url="/resource/sample/images/sample.png" />
    <sun:staticText text="Sample Resource" />
  </sun:hyperlink>
</sun:property>

<sun:property>
  <sun:hyperlink
    tooltip="Another"
    url="/sample/page/testPage.jsf?name=Another" >
    <sun:image url="/resource/sample/images/sample.png" />
    <sun:staticText text="Another" />
  </sun:hyperlink>
</sun:property>
```

The file specifies two simple properties on the property sheet, one above the other. Each consists of a `sun:hyperlink` element (a link to a URL). Within each `sun:hyperlink` element is nested a `sun:image` element, specifying an image, and a `sun:staticText` element, specifying the text to be placed next to the image.

Each `sun:hyperLink` element uses a `toolTip` attribute and a `url` attribute. Each `url` attribute references the `testPage.jsf` file that is used elsewhere in the example, specifying different content for the `name` parameter.

You can use many other kinds of user interface elements within a `sun:property` element.

Adding a Page to the Administration Console

Your add-on component may require new configuration tasks. In addition to implementing commands that accomplish these tasks (see [Chapter 4, “Extending the `asadmin` Utility](#)”), you can provide property sheets that enable users to configure your component or to perform tasks such as creating and editing resources for it.

EXAMPLE 3-13 Example JavaServer Faces Page for a Property Sheet

Most of the user interface features used in the example reference the file `testPage.jsf` or (for tabs) the file `tabPage.jsf`. Both files are in the `src/main/resources/page/` directory. The `testPage.jsf` file looks like this:

```
<!composition template="/templates/default.layout" guiTitle="TEST Sample Page Title">
<!define name="content">
<sun:form id="propertyForm">

<sun:propertySheet id="propertySheet">
  <sun:propertySheetSection id="propertySection">
    <sun:property id="prop1" labelAlign="left" noWrap="true"
      overlapLabel="false" label="Test Page Name:" >
      <sun:staticText text="$pageSession{pageName}" >
        <!beforeCreate
          getRequestValue(key="name" value=>$page{pageName});
        />
      </sun:staticText>
    </sun:property>
  </sun:propertySheetSection>
</sun:propertySheet>
<sun:hidden id="helpKey" value="" />

</sun:form>
</define>
</composition>
```

The page uses the `composition` directive to specify that the page uses the `default.layout` template and to specify a page title. The page uses additional directives, events, and tags to specify its content.

Adding Internationalization Support

To add internationalization support for your add-on component to the Administration Console, you can place an event and handler like the following at the top of your page:

```
<!initPage
    setResourceBundle(key="yourI18NKey" bundle="bundle.package.BundleName")
/>
```

Replace the values `yourI18NKey` and `bundle.package.BundleName` with appropriate values for your component.

Changing the Theme or Brand of the Administration Console

To change the theme or brand of the Administration Console for your add-on component, use the integration point type `org.glassfish.admingui:customtheme`. This integration point affects the Cascading Style Sheet (CSS) files and images that are used in the Administration Console.

EXAMPLE 3-14 Example Custom Theme Integration Point

For example, the following integration point specifies a custom theme:

```
<integration-point
    id="myOwnBrand"
    type="org.glassfish.admingui:customtheme"
    priority="2"
    content="myOwnBrand.properties"
/>
```

The `priority` attribute works differently when you specify it in a branding integration point from the way it works in other integration points. You can place multiple branding add-on components in the `modules` directory, but only one theme can be applied to the Administration Console. The `priority` attribute determines which theme is used. Specify a value from 1 to 100; the lower the number, the higher the priority. The integration point with the highest priority will be used.

Additional integration point types also affect the theme or brand of the Administration Console:

`org.glassfish.admingui:masthead`

Specifies the name and location of the include masthead file, which can be customized with a branding image. This include file will be integrated on the masthead of the Administration Console.

`org.glassfish.admingui:loginimage`

Specifies the name and location of the include file containing the branding login image code that will be integrated with the login page of the Administration Console.

`org.glassfish.admingui:loginform`

Specifies the name and location of the include file containing the customized login form code. This code also contains the login background image used for the login page for the Administration Console.

`org.glassfish.admingui:versioninfo`

Specifies the name and location of the include file containing the branding image that will be integrated with the content of the version popup window.

EXAMPLE 3-15 Example of Branding Integration Points

For example, you might specify the following integration points. The content for each integration point is defined in an include file.

```
<integration-point
  id="myOwnBrandMast"
  type="org.glassfish.admingui:masthead"
  priority="80"
  content="branding/masthead.inc"
/>
<integration-point
  id="myOwnBrandLogImg"
  type="org.glassfish.admingui:loginimage"
  priority="80"
  content="branding/loginimage.inc"
/>
<integration-point
  id="myOwnBrandLogFm"
  type="org.glassfish.admingui:loginform"
  priority="80"
  content="branding/loginform.inc"
/>
<integration-point
  id="myOwnBrandVersInf"
  type="org.glassfish.admingui:versioninfo"
  priority="80"
  content="branding/versioninfo.inc"
/>
```

To provide your own CSS and images to modify the global look and feel of the entire application (not just the Administration Console), use the [theming feature of Project Woodstock](https://woodstock.dev.java.net/docs/specs/ThemeFS.html) (<https://woodstock.dev.java.net/docs/specs/ThemeFS.html>). Create a theme JAR file with all the CSS properties and image files that are required by your Woodstock

component. Use a script provided by the Woodstock project to clone an existing theme, then modify the files and properties as necessary. Once you have created the theme JAR file, place it in the `WEB-INF/lib` directory of the Administration Console so that the Woodstock theme component will load the theme. In addition, edit the properties file specified by your integration point (`MyOwnBrand.properties`, for example) to specify the name and version of your theme.

Creating an Integration Point Type

If your add-on component provides new content that you would like other people to extend, you may define your own integration point types. For example, if you add a new page that provides tabs of monitoring information, you might want to allow others to add their own tabs to complement your default tabs. This feature enables your page to behave like the existing Administration Console pages that you or others can extend.

▼ To Create an Integration Point Type

1 Decide on the name of your integration point type.

The integration point type must be a unique identifier. You might use the package name of your integration point, with a meaningful name appended to the end, as in the following example:

```
org.company.project:myMonitoringTabs
```

2 After you have an integration point ID, use handlers to insert the integration point implementation(s).

Include code like the following below the place in your JavaServer Faces page where you would like to enable others to add their integration point implementations:

```
<event>
  <!afterCreate
    getUIComponent(clientId="clientId:of:root"
                    component=>${attribute{rootComp}});
    includeIntegrations(type="org.company.project:myMonitoringTabs"
                       root="#{rootComp}");
  />
</event>
```

Change `clientId:of:root` to match the `clientId` of the outermost component in which you want others to be able to add their content (in this example, the tab set is the most likely choice). Also include your integration point ID in place of `org.company.project:myMonitoringTabs`. If you omit the `root` argument to `includeIntegrations`, all components on the entire page can be used for the `parentId` of the integration points.

- 3** To enable others to use this integration point, document it at the [GlassFish Integration Point wiki page](http://wiki.glassfish.java.net/Wiki.jsp?page=V3IntegrationPoint) (<http://wiki.glassfish.java.net/Wiki.jsp?page=V3IntegrationPoint>).

Document the integration point only if your content is publicly available.

You or others can now provide an integration point that will be integrated into this page.

See Also For more information, see the [JSFTemplating API documentation](https://jsftemplating.dev.java.net/nonav/javadoc/index.html) (<https://jsftemplating.dev.java.net/nonav/javadoc/index.html>).

Extending the `asadmin` Utility

The `asadmin` utility is a command-line tool for configuring and administering Enterprise Server. Extending the `asadmin` utility enables you to provide administrative interfaces for an add-on component that are consistent with the interfaces of other Enterprise Server components. A user can run `asadmin` commands either from a command prompt or from a script. For more information about the `asadmin` utility, see the [`asadmin\(1M\)` man page](#).

The following topics are addressed here:

- “About the Administrative Command Infrastructure of Enterprise Server” on page 47
- “Adding an `asadmin` Command” on page 48
- “Adding Parameters to an `asadmin` Command” on page 50
- “Adding Message Text Strings to an `asadmin` Command” on page 54
- “Enabling an `asadmin` Command to Run” on page 57
- “Setting the Context of an `asadmin` Command” on page 57
- “Changing the Brand in the Enterprise Server CLI” on page 57
- “Examples of Extending the `asadmin` Utility” on page 59

About the Administrative Command Infrastructure of Enterprise Server

To enable multiple containers to be independently packaged and loaded, the administrative command infrastructure of Enterprise Server provides the following features:

- **Location independence.** Administration commands can be loaded from any add-on component that is known to Enterprise Server.
- **Extensibility.** Administrative commands that are available to Enterprise Server are discovered on demand and not obtained from a preset list of commands.
- **Support for the HK2 architecture.** Commands can use injection to express their dependencies, and extraction to provide results to a user. For more information, see [Chapter 2, “Writing HK2 Components.”](#)

Adding an `asadmin` Command

An `asadmin` command identifies the operation or task that a user is to perform. Adding an `asadmin` command enables the user to perform these tasks and operations through the `asadmin` utility.

The following topics are addressed here:

- [“Representing an `asadmin` Command as a Java Class” on page 48](#)
- [“Specifying the Name of an `asadmin` Command” on page 48](#)
- [“Ensuring That an `AdminCommand` Implementation Is Stateless” on page 49](#)
- [“Example of Adding an `asadmin` Command” on page 49](#)

Representing an `asadmin` Command as a Java Class

Each `asadmin` command that you are adding must be represented as a Java class. To represent an `asadmin` command as a Java class, write a Java class that implements the `org.glassfish.api.admin.AdminCommand` interface. Write one class for each command that you are adding. Do *not* represent multiple `asadmin` commands in a single class.

Annotate the declaration of your implementations of the `AdminCommand` interface with the `org.jvnet.hk2.annotations.Service` annotation. The `@Service` annotation ensures that the following requirements for your implementations are met:

- The implementations are eligible for resource injection and resource extraction.
- The implementations are location independent, provided that the component that contains them is made known to the Enterprise Server runtime.

For information about how to make a component known to the Enterprise Server runtime, see [“Integrating an Add-On Component With Enterprise Server” on page 72](#).

Specifying the Name of an `asadmin` Command

To specify the name of the command, set the `name` element of the `@Service` annotation to the name.

Note – Command names are case-sensitive.

Commands that are supplied in Enterprise Server distributions typically create, delete, and list objects of a particular type. For consistency with the names of commands that are supplied in Enterprise Server distributions, follow these conventions when specifying the name of a command:

- For commands that create an object of a particular type, use the name `create-object`.

- For commands that delete an object of a particular type, use the name `delete-object`.
- For commands that list all objects of a particular type, use the name `list-objects`.

For example, Enterprise Server provides the following commands for creating, deleting, and listing HTTP listeners:

- `create-http-listener`
- `delete-http-listener`
- `list-http-listeners`

You must also ensure that the name of your command is unique. To obtain a complete list of the names of all asadmin commands that are installed, use the `list-commands(1)` command. For a complete list of asadmin commands that are supplied in Enterprise Server distributions, see [Sun GlassFish Enterprise Server v3 Reference Manual](#).

Ensuring That an AdminCommand Implementation Is Stateless

To enable multiple clients to run a command simultaneously, ensure that the implementation of the AdminCommand interface for the command is stateless. To ensure that the implementation of the AdminCommand interface is stateless, annotate the declaration of your implementation with the `org.jvnet.hk2.annotations.Scoped` annotation. In the `@Scoped` annotation, set the scope as follows:

- To instantiate the command for each lookup, set the scope to `PerLookup.class`.
- To instantiate the command only once for each session, set the scope to `Singleton`.

Example of Adding an asadmin Command

EXAMPLE 4-1 Adding an asadmin Command

This example shows the declaration of the class `CreateMycontainer` that represents an asadmin command that is named `create-mycontainer`. The command is instantiated for each lookup.

```
package com.example.mycontainer;

import org.glassfish.api.admin.AdminCommand;
...
import org.jvnet.hk2.annotations.Service;
...
import org.jvnet.hk2.annotations.Scoped;
import org.jvnet.hk2.component.PerLookup;
```

EXAMPLE 4-1 Adding an `asadmin` Command (Continued)

```
/**
 * Sample command
 */
@Service(name="create-mycontainer")
@Scoped(PerLookup.class)
public class CreateMycontainer implements AdminCommand {
    ...
}
```

Adding Parameters to an `asadmin` Command

The parameters of an `asadmin` command are the options and operands of the command.

- *Options* control how the `asadmin` utility performs a command.
- *Operands* are the objects on which a command acts. For example, the operand of the `start-domain(1)` command is the domain that is to be started.

The following topics are addressed here:

- [“Representing a Parameter of an `asadmin` Command” on page 50](#)
- [“Identifying a Parameter of an `asadmin` Command” on page 50](#)
- [“Specifying Whether a Parameter Is an Option or an Operand” on page 51](#)
- [“Specifying the Name of an Option” on page 51](#)
- [“Specifying the Acceptable Values of a Parameter” on page 52](#)
- [“Specifying the Default Value of a Parameter” on page 52](#)
- [“Specifying Whether a Parameter Is Required or Optional” on page 53](#)
- [“Example of Adding Parameters to an `asadmin` Command” on page 53](#)

Representing a Parameter of an `asadmin` Command

Represent each parameter of a command in your implementation as a field or as the property of a JavaBeans™ specification setter method. Use the property of a setter method for the following reasons:

- To provide data encapsulation for the parameter
- To add code for validating the parameter before the property is set

Identifying a Parameter of an `asadmin` Command

Identifying a parameter of an `asadmin` command enables Enterprise Server to perform the following operations at runtime on the parameter:

- **Validation.** The Enterprise Server determines whether all required parameters are specified and returns an error if any required parameter is omitted.
- **Injection.** Before the command runs, the Enterprise Server injects each parameter into the required field or method before the command is run.
- **Usage message generation.** The Enterprise Server uses reflection to obtain the list of parameters for a command and to generate the usage message from this list.
- **Localized string display.** If the command supports internationalization and if localized strings are available, the Enterprise Server can automatically obtain the localized strings for a command and display them to the user.

To identify a parameter of a command, annotate the declaration of the item that is associated with the parameter with the `org.glassfish.api.Param` annotation. This item is either the field or setter method that is associated with the parameter.

To specify the properties of the parameter, use the elements of the `@Param` annotation as explained in the sections that follow.

Specifying Whether a Parameter Is an Option or an Operand

Whether a parameter is an option or an operand determines how a user must specify the parameter when running the command:

- If the parameter is an option, the user must specify the option with the parameter name.
- If the parameter is an operand, the user may omit the parameter name.

To specify whether a parameter is an option or an operand, set the `primary` element of the `@Param` annotation as follows:

- If the parameter is an option, set the `primary` element to `false`. This value is the default.
- If the parameter is an operand, set the `primary` element to `true`.

Specifying the Name of an Option

The name of an option is the name that a user must type on the command line to specify the option when running the command.

The name of each option that you add in your implementation of an `asadmin` command can have a long form and a short form. When running the command, the user specifies the long form and the short form as follows:

- The short form of an option name has a single dash (-) followed by a single character.
- The long form of an option name has two dashes (- -) followed by an option word.

For example, the short form and the long form of the name of the option for specifying terse output are as follows:

- Short form: `-m`
- Long form: `--monitor`

Note – Option names are case-sensitive.

Specifying the Long Form of an Option Name

To specify the long form of an option name, set the `name` element of the `@Param` annotation to a string that specifies the name. If you do not set this element, the default name depends on how you represent the option.

- If you represent the option as a field, the default name is the field name.
- If you represent the option as the property of a JavaBeans specification setter method, the default name is the property name from the setter method name. For example, if the setter method `setPassword` is associated with an option, the property name and the option name are both `password`.

Specifying the Short Form of an Option Name

To specify the short form of an option name, set the `shortName` element of the `@Param` annotation to a single character that specifies the short form of the parameter. The user can specify this character instead of the full parameter name, for example `-m` instead of `--monitor`. If you do not set this element, the option has no short form.

Specifying the Acceptable Values of a Parameter

When a user runs the command, the Enterprise Server validates option arguments and operands against the acceptable values that you specify in your implementation.

To specify the acceptable values of a parameter, set the `acceptableValues` element of the `@Param` annotation to a string that contains a comma-separated list of acceptable values. If you do not set this element, any string of characters is acceptable.

Specifying the Default Value of a Parameter

The default value of a parameter is the value that is applied if a user omits the parameter when running the command.

To specify the default value of a parameter, set the `defaultValue` element of the `@Param` annotation to a string that contains the default value. If you do not set this element, the parameter has no default value.

Specifying Whether a Parameter Is Required or Optional

Whether a parameter is required or optional determines how a command responds if a user omits the parameter when running the command:

- If the parameter is required, the command returns an error.
- If the parameter is optional, the command runs successfully.

To specify whether a parameter is optional or required, set the `optional` element of the `@Param` annotation as follows:

- If the parameter is required, set the `optional` element to `false`. This value is the default.
- If the parameter is optional, set the `optional` element to `true`.

Example of Adding Parameters to an `asadmin` Command

EXAMPLE 4-2 Adding Parameters to an `asadmin` Command

This example shows the code for adding parameters to an `asadmin` command with the properties as shown in the table.

Name	Represented As	Acceptable Values	Default Value	Optional or Required	Short Name	Option or Operand
<code>--originator</code>	A field that is named <code>originator</code>	Any character string	None defined	Required	None	Option
<code>--description</code>	A field that is named <code>mycontainerDescription</code>	Any character string	None defined	Optional	None	Option
<code>--enabled</code>	A field that is named <code>enabled</code>	<code>true</code> or <code>false</code>	<code>false</code>	Optional	None	Option
<code>--containername</code>	A field that is named <code>containername</code>	Any character string	None defined	Required	None	Operand

```
...
import org.glassfish.api.Param;
...
{
...

```

```

    @Param
    String originator;

    @Param(name="description", optional=true)
    ...
    String mycontainerDescription

    @Param (acceptableValues="true,false", defaultValue="false", optional=true)
    String enabled

    @Param(primary=true)
    String containername;

    ...
}

```

Adding Message Text Strings to an `asadmin` Command

A message text string provides useful information to the user about an `asadmin` command or a parameter.

To provide internationalization support for the text string of a command or parameter, annotate the declaration of the command or parameter with the `org.glassfish.api.I18n` annotation. The `@I18n` annotation identifies the resource from the resource bundle that is associated with your implementation.

To add message text strings to an `asadmin` command, create a plain text file that is named `LocalStrings.properties` to contain the strings. Define each string on a separate line of the file as follows:

key=string

key

A key that maps the string to a command or a parameter. The format to use for *key* depends on the target to which the key applies and whether the target is annotated with the `@I18n` annotation. See the following table.

Target	Format
Command or parameter with the <code>@I18n</code> annotation	<i>command-name.command.resource-name</i>
Command without the <code>@I18n</code> annotation	<i>command-name.command</i>

Target	Format
Parameter without the <code>@I18n</code> annotation	<code>command-name . command . param-name</code>

The replaceable parts of these formats are as follows:

- `command-name` The name of the command.
- `resource-name` The name of the resource that is specified in the `@I18n` annotation.
- `param-name` The name of the parameter.

string

A string without quotes that contains the text of the message.

Note – To display the message strings to users, you must provide code in your implementation of the `execute` method to display the text. For more information about implementing the `execute` method, see [“Enabling an `asadmin` Command to Run” on page 57](#).

EXAMPLE 4-3 Adding Message Strings to an `asadmin` Command

This example shows the code for adding message strings to the `create-mycontainer` command as follows:

- The `create-mycontainer` command is associated with the message `Creates a custom container`. No internationalization support is provided for this message.
- The `--originator` parameter is associated with the message `The originator of the container`. No internationalization support is provided for this message.
- The `--description` parameter is associated with the message that is contained in the resource `mydesc`, for which internationalization is provided. This resource contains the message text `A description of the container`.
- The `--enabled` parameter is associated with the message `Whether the container is enabled or disabled`. No internationalization support is provided for this message.
- The `--containername` parameter is associated with the message `The container name`. No internationalization support is provided for this message.

The addition of the parameters `originator`, `description`, `enabled` and `containername` to the command is shown in [Example 4-2](#).

```
package com.example.mycontainer;

import org.glassfish.api.admin.AdminCommand;
...
import org.glassfish.api.I18n;
```

EXAMPLE 4-3 Adding Message Strings to an asadmin Command *(Continued)*

```

import org.glassfish.api.Param;
import org.jvnet.hk2.annotations.Service;
...
import org.jvnet.hk2.annotations.Scoped;
import org.jvnet.hk2.component.PerLookup;

/**
 * Sample command
 */
@Service(name="create-mycontainer")
@Scoped(PerLookup.class)
public Class CreateMycontainer implements AdminCommand {

    ...

    @Param
    String originator;

    @Param(name="description", optional=true)
    @I18n("mydesc")
    String mycontainerDescription

    @Param (acceptableValues="true,false", defaultValue="false", optional=true)
    String enabled

    @Param(primary=true)
    String containername;
    ...
}

```

The following message text strings are defined in the file `LocalStrings.properties` for use by the command:

```

create-mycontainer.command=Creates a custom container
create-mycontainer.command.originator=The originator of the container
create-mycontainer.command.mydesc=A description of the container
create-mycontainer.command.enabled=Whether the container is enabled or disabled
create-mycontainer.command.containername=The container name

```

Enabling an `asadmin` Command to Run

To enable an `asadmin` command to run, implement the `execute` method in your implementation of the `AdminCommand` interface. The declaration of the `execute` method in your implementation must be as follows.

```
public void execute(AdminCommandContext context);
```

Pass each parameter of the command as a property to your implementation of the `execute` method. Set the key of the property to the parameter name and set the value of the property to the parameter's value.

Setting the Context of an `asadmin` Command

The `org.glassfish.api.admin.AdminCommandContext` class provides the following services to an `asadmin` command:

- Access to the parameters of the command
- Logging
- Reporting

To set the context of an `asadmin` command, pass an `AdminCommandContext` object to the `execute` method of your implementation.

Changing the Brand in the Enterprise Server CLI

The brand in the Enterprise Server command-line interface (CLI) consists of the product name and release information that are displayed in the following locations:

- In the string that the `version(1)` command displays
- In each entry in the `server.log` file

If you are incorporating Enterprise Server into a new product with an external vendor's own brand name, change the brand in the Enterprise Server CLI.

To change the brand in the Enterprise Server CLI, create an OSGi fragment bundle that contains a plain text file that is named `src/main/resources/BrandingVersion.properties`.

In the `BrandingVersion.properties` file, define the following keyword-value pairs:

```
product_name=product-name
abbrev_product_name=abbrev-product-name
major_version=major-version
minor_version=minor-version
```

```
build_id=build-id
version_prefix=version-prefix
version_suffix=version-suffix
```

Define each keyword-value pair on a separate line of the file. Each value is a text string without quotes.

The meaning of each keyword-value pair is as follows:

```
product_name=product-name
```

Specifies the full product name without any release information, for example, Sun GlassFish Enterprise Server.

```
abbrev_product_name=abbrev-product-name
```

Specifies an abbreviated form of the product name without any release information, for example, Sun GlassFish.

```
major_version=major-version
```

Returns the product major version, for example, 3

```
minor_version=minor-version
```

Specifies the product minor version, for example, 0.

```
build_id=build-id
```

Specifies the build version, for example, build 17.

```
version_prefix=version-prefix
```

Specifies a prefix for the product version, for example, v.

```
version_suffix=version-suffix
```

Specifies a suffix for the product version, for example, Prelude.

EXAMPLE 4-4 BrandingVersion.properties File for Changing the Brand in the Enterprise Server CLI

This example shows the content of the BrandingVersion.properties for defining the product name and release information of Sun GlassFish Enterprise Server v3.0 Prelude, build 17. The abbreviated product name is sun-glassfish.

```
product_name=Sun GlassFish Enterprise Server
abbrev_product_name=sun-glassfish
major_version=3
minor_version=0
build_id=build 17
version_prefix=v
version_suffix=Prelude
```

Examples of Extending the asadmin Utility

EXAMPLE 4-5 Example asadmin Command With Empty execute Method

This example shows a class that represents the asadmin command `create-mycontainer`.

The usage statement for this command is as follows:

```
asadmin create-mycontainer --originator any-character-string
[--description any-character-string]
[--enabled {true|false}] any-character-string
```

This command uses injection to specify that a running domain is required.

```
package com.example.mycontainer;

import org.glassfish.api.admin.AdminCommand;
import org.glassfish.api.admin.AdminCommandContext;
import org.glassfish.api.I18n;
import org.glassfish.api.Param;
import org.jvnet.hk2.annotations.Service;
import org.jvnet.hk2.annotations.Inject;
import org.jvnet.hk2.annotations.Scoped;
import org.jvnet.hk2.component.PerLookup;

/**
 * Sample command
 */
@Service(name="create-mycontainer")
@Scoped(PerLookup.class)
public class CreateMycontainer implements AdminCommand {

    @Inject
    Domain domain;

    @Param
    String originator;

    @Param(name="description", optional=true)
    @I18n("mydesc")
    String mycontainerDescription

    @Param (acceptableValues="true,false", defaultValue="false", optional=true)
    String enabled

    @Param(primary=true)
    String containername;
```

EXAMPLE 4-5 Example asadmin Command With Empty execute Method *(Continued)*

```
/**
 * Executes the command with the command parameters passed as Properties
 * where the keys are the paramter names and the values the parameter values
 * @param context information
 */
public void execute(AdminCommandContext context) {
    // domain and originator are not null
    // mycontainerDescription can be null.
}
}
```

The following message text strings are defined in the file `LocalStrings.properties` for use by the command:

```
create-mycontainer.command=Creates a custom container
create-mycontainer.command.originator=The originator of the container
create-mycontainer.command.mydesc=A description of the container
create-mycontainer.command.enabled=Whether the container is enabled or disabled
create-mycontainer.command.containername=The container name
```

EXAMPLE 4-6 Fully Functional asadmin Command

This example shows a class that represents the asadmin command `list-runtime-environment`. The command determines the operating system or runtime information for Enterprise Server

The usage statement for this command is as follows:

```
asadmin list-runtime-environment{runtime|os}
```

```
package com.example.env.cli;

import org.glassfish.api.admin.AdminCommand;
import org.glassfish.api.admin.AdminCommandContext;
import org.glassfish.api.ActionReport;
import org.glassfish.api.I18n;
import org.glassfish.api.ActionReport.ExitCode;
import org.glassfish.api.Param;
import org.jvnet.hk2.annotations.Service;
import org.jvnet.hk2.annotations.Inject;
import org.jvnet.hk2.annotations.Scoped;
import org.jvnet.hk2.component.PerLookup;

import java.lang.management.ManagementFactory;
import java.lang.management.OperatingSystemMXBean;
```

EXAMPLE 4-6 Fully Functional asadmin Command (Continued)

```

import java.lang.management.RuntimeMXBean;

/**
 * Demos asadmin CLI extension
 *
 */
@Service(name="list-runtime-environment")
@Scoped(PerLookup.class)
public class ListRuntimeEnvironmentCommand implements AdminCommand {

    // this value can be either runtime or os for our demo
    @Param(primary=true)
    String inParam;

    public void execute(AdminCommandContext context) {

        ActionReport report = context.getActionReport();
        report.setActionExitCode(ExitCode.SUCCESS);

        // If the inParam is 'os' then this command returns operating system info
        // and if the inParam is 'runtime' then it returns runtime info.
        // Both of the above are based on mxbeans.

        if ("os".equals(inParam)) {
            OperatingSystemMXBean osmb = ManagementFactory.getOperatingSystemMXBean();
            report.setMessage("Your machine operating system name = " + osmb.getName());
        } else if ("runtime".equals(inParam)) {
            RuntimeMXBean rtmb = ManagementFactory.getRuntimeMXBean();
            report.setMessage("Your JVM name = " + rtmb.getVmName());
        } else {
            report.setActionExitCode(ExitCode.FAILURE);
            report.setMessage("operand should be either 'os' or 'runtime'");
        }
    }
}

```


Adding Container Capabilities

Applications run on Enterprise Server in containers. Enterprise Server enables you to create containers that extend or replace the existing containers of Enterprise Server. Adding container capabilities enables you to run new types of applications and to deploy new archive types in Enterprise Server.

The following topics are addressed here:

- [“Creating a Container Implementation” on page 63](#)
- [“Adding an Archive Type” on page 66](#)
- [“Creating Connector Modules” on page 68](#)

Creating a Container Implementation

To implement a container that extends or replaces a service in Enterprise Server, you must create a Java programming language class that includes the following characteristics:

- It is annotated with the `org.jvnet.hk2.annotations.Service` annotation.
- It implements the `org.glassfish.api.container.Container` interface.

Marking the Class with the `@Service` Annotation

Add a `com.jvnet.hk2.annotations.Service` annotation at the class definition level to identify your class as a service implementation.

```
@Service
public class MyContainer implements Container {
    ...
}
```

To avoid potential name collisions with other containers, use the fully qualified class name of your container class in the `@Service` annotation's name element:

```
package com.example.containers;
...

@Service(name="com.example.containers.MyContainer")
public class MyContainer implements Container {
...
}
```

Implementing the Container Interface

The `org.glassfish.api.container.Container` interface is the contract that defines a container implementation. Classes that implement `Container` can extend or replace the functionality in Enterprise Server by allowing applications to be deployed and run within the Enterprise Server runtime.

The `Container` interface consists of two methods, `getDeployer` and `getName`. The `getDeployer` method returns an implementation class of the `org.glassfish.api.deployment.Deployer` interface capable of managing applications that run within this container. The `getName` method returns a human-readable name for the container, and is typically used to display messages belonging to the container.

The `Deployer` interface defines the contract for managing a particular application that runs in the container. It consists of the following methods:

`getMetaData`

Retrieves the metadata used by the `Deployer` instance, and returns an `org.glassfish.api.deployment.MetaData` object.

`loadMetaData`

Loads the metadata associated with an application.

`prepare`

Prepares the application to run in Enterprise Server.

`load`

Loads a previously prepared application to the container.

`unload`

Unloads or stops a previously loaded application.

`clean`

Removes any artifacts generated by an application during the `prepare` phase.

The `DeploymentContext` is the usual context object passed around `deployer` instances during deployment.

EXAMPLE 5-1 Example Implementation of Container

This example shows a Java programming language class that implements the Container interface and is capable of extending the functionality of Enterprise Server.

```
package com.example.containers;
contains
@Service(name="com.example.containers.MyContainer")
public class MyContainer implements Container {
    public String getName() {
        return "MyContainer";
    }

    public Class<? extends org.glassfish.api.deployment.Deployer> getDeployer() {
        return MyDeployer.class;
    }
}
```

EXAMPLE 5-2 Example Implementation of Deployer

```
package com.example.containers;

@Service
public class MyDeployer {

    public Metadata getMetadata() {
        return new Metadata(...);
    }

    public <V> v loadMetadata(Class<V> type, DeploymentContext dc) {
        ...
    }

    public boolean prepare(DeploymentContext dc) {
        // performs any actions needed to allow the application to run,
        // such as generating artifacts
        ...
    }

    public MyApplication load(MyContainer container, DeploymentContext dc) {
        // creates a new instance of an application
        MyApplication myApp = new MyApplication (...);
        ...
        // returns the application instance
        return myApp;
    }

    public void unload(MyApplication myApp, DeploymentContext dc) {
```

EXAMPLE 5-2 Example Implementation of `Deployer` (Continued)

```
        // stops and removes the application
        ...
    }

    public void clean (DeploymentContext dc) {
        // cleans up any artifacts generated during prepare()
        ...
    }
}
```

Adding an Archive Type

An archive type is an abstraction of the archive file format. An archive type can be implemented as a plain JAR file, as a directory layout, or a custom type. By default, Enterprise Server recognizes JAR based and directory based archive types. A new container might require a new archive type.

There are two sub-interfaces of the `org.glassfish.api.deployment.archive.Archive` interface, `org.glassfish.api.deployment.archive.ReadableArchive` and `org.glassfish.api.deployment.archive.WritableArchive`. Typically developers of new archive types will provide separate implementations of `ReadableArchive` and `WritableArchive`, or a single implementation that implements both `ReadableArchive` and `WritableArchive`.

Implementations of the `ReadableArchive` interface provide read access to an archive type. `ReadableArchive` defines the following methods:

`getEntry(String name)`

Returns a `java.io.InputStream` for the specified entry name, or null if the entry doesn't exist.

`exists(String name)`

Returns a boolean value indicating whether the specified entry name exists.

`getEntrySize(String name)`

Returns the size of the specified entry as a long value.

`open(URI uri)`

Returns an archive for the given `java.net.URI`.

`getSubArchive(String name)`

Returns an instance of `ReadableArchive` for the specified sub-archive contained within the parent archive, or null if no such archive exists.

`exists()`

Returns a boolean value indicating whether this archive exists.

`delete()`

Deletes the archive, and returns a `boolean` value indicating whether the archive has been successfully deleted.

`renameTo(String name)`

Renames the archive to the specified name, and returns a `boolean` value indicating whether the archive has been successfully renamed.

Implementations of the `WritableArchive` interface provide write access to the archive type. `WritableArchive` defines the following methods:

`create(URI uri)`

Creates a new archive with the given path, specified as a `java.net.URI`.

`closeEntry(WritableArchive subArchive)`

Closes the specified sub-archive contained within the parent archive.

`closeEntry()`

Closes the current entry.

`createSubArchive(String name)`

Creates a new sub-archive in the parent archive with the specified name, and returns it as a `WritableArchive` instance.

`putNextEntry(String name)`

Creates a new entry in the archive with the specified name, and returns it as a `java.io.OutputStream`.

Implementing the ArchiveHandler Interface

An archive handler is responsible for handling the particular layout of an archive. Java EE defines a set of archives (WAR, JAR, and RAR, for example), and each of these archives has an `ArchiveHandler` instance associated with the archive type.

Each layout should have one handler associated with it. There is no extension point support at this level; the archive handler's responsibility is to give access to the classes and resources packaged in the archive, and it should not contain any container-specific code. The `java.lang.ClassLoader` returned by the handler is used by all the containers in which the application will be deployed.

`ArchiveHandler` defines the following methods:

`getArchiveType()`

Returns the name of the archive type as a `String`. Typically, this is the archive extension, such as `jar` or `war`.

`getDefaultApplicationName(ReadableArchive archive)`

Returns the default name of the specified archive as a `String`. Typically this default name is the name part of the URI location of the archive.

`handles(ReadableArchive archive)`

Returns a `boolean` value indicating whether this implementation of `ArchiveHandler` can work with the specified archive.

`getClassLoader(DeploymentContext dc)`

Returns a `java.lang.ClassLoader` capable of loading all classes from the archive passed in by the `DeploymentContext` instance. Typically the `ClassLoader` will load classes in the scratch directory area, returned by `DeploymentContext.getScratchDir()`, as stubs and other artifacts are generated in the scratch directory.

`expand(ReadableArchive source, WritableArchive target)`

Prepares the `ReadableArchive` source archive for loading into the container in a format the container accepts. Such preparation could be to expand a compressed archive, or possibly nothing at all if the source archive format is already in a state that the container can handle. This method returns the archive as an instance of `WritableArchive`.

Creating Connector Modules

Connector modules are small add-on modules that consist of application “sniffers” that associate application types with containers that can run the application type. Enterprise Server connector modules are separate from the associated add-on module that delivers the container implementation to allow Enterprise Server to dynamically install and configure containers on demand.

When a deployment request is received by the Enterprise Server runtime:

1. The current `Sniffer` implementations are used to determine the application type.
2. Once an application type is found, the runtime looks for a running container associated with that application type. If no running container is found, the runtime attempts to install and configure the container associated with the application type as defined by the `Sniffer` implementation.
3. The `Deployer` interface is used to prepare and load the implementation.

Associating File Types with Containers Using the Sniffer Interface

Containers do not necessarily need to be installed on the local machine for Enterprise Server to recognize the container's application type. Enterprise Server uses a “sniffer” concept to study the artifacts in a deployment request and to choose the associated container that handles the application type that the user is trying to deploy. To create this association, create a Java programming language class that implements the `org.glassfish.api.container.Sniffer` interface. This implementation can be as simple as looking for a specific file in the application's archive (such as the presence of `WEB-INF/web.xml`), or as complicated as running an annotation

scanner to determine an XML-less archive (such as enterprise bean annotations in a JAR file). A `Sniffer` implementation must be as small as possible and must not load any of the container's runtime classes.

A simple version of a `Sniffer` implementation uses the `handles` method to check the existence of a file in the archive that denotes the application type (as `WEB-INF/web.xml` denotes a web application). Once a `Sniffer` implementation has detected that it can handle the deployment request artifact, Enterprise Server calls the `setUp` method. The `setUp` method is responsible for setting up the container, which can involve one or more of the following actions:

- Downloading the container's runtime (the first time that a container is used)
- Installing the container's runtime (the first time that a container is used)
- Setting up one or more repositories to access the runtime's classes (these are implementations of the `HK2.com.sun.enterprise.module.Repository` interface, such as the `com.sun.enterprise.module.impl.DirectoryBasedRepository` class)

The `setUp` method returns an array of the `com.sun.enterprise.module.Module` objects required by the container.

The `Sniffer` interface defines the following methods:

`handles(ReadableArchive source, ClassLoader loader)`

Returns a boolean value indicating whether this `Sniffer` implementation can handle the specified archive.

`getURLPatterns()`

Returns a `String` array containing all URL patterns to apply against the request URL. If a pattern matches, the service method of the associated container is invoked.

`getAnnotationTypes()`

Returns a list of annotation types recognized by this `Sniffer` implementation. If an application archive contains one of the returned annotation types, the deployment process invokes the container's deployers as if the `handles` method had returned true.

`getModuleType()`

Returns the module type associated with this `Sniffer` implementation as a `String`.

`setup(String containerHome, Logger logger)`

Sets up the container libraries so that any dependent bundles from the connector JAR file will be made available to the HK2 runtime. The `setup` method returns an array of `com.sun.enterprise.module.Module` classes, which are definitions of container implementations. Enterprise Server can then load these modules so that it can create an instance of the container's `Deployer` or `Container` implementations when it needs to. The module is locked as long as at least one module is loaded in the associated container.

`teardown()`

Removes a container and all associated modules in the HK2 modules subsystem.

`getContainerNames()`

Returns a `String` array containing the `Container` implementations that this `Sniffer` implementation enables.

`isVisible()`

Returns a `boolean` value indicating whether this `Sniffer` implementation should be visible to end-users.

`getDeploymentConfigurations(final ReadableArchive source)`

Returns a `Map<String, String>` of deployment configuration names to configurations from this `Sniffer` implementation for the specified application (the archive source). The names are created by Enterprise Server; the configurations are the names of the files that contain configuration information (for example, `WEB-INF/web.xml` and possibly `WEB-INF/sun-web.xml` for a web application). If the `getDeploymentConfigurations` method encounters errors while searching or reading the specified archive source, it throws a `java.io.IOException`.

Making Sniffer Implementations Available to the Enterprise Server

Package `Sniffer` implementation code into modules and install the modules in the *as-install/modules* directory. Enterprise Server will automatically discover these modules. If an administrator installs connector modules that contain `Sniffer` implementations while Enterprise Server is running, Enterprise Server will pick them up at the next deployment request.

Packaging, Integrating, and Delivering an Add-On Component

Packaging an add-on component enables the component to interact with the Enterprise Server kernel in the same way as other components. Integrating a component with Enterprise Server enables Enterprise Server to discover the component at runtime. If an add-on component is an extension or update to existing installations of Enterprise Server, deliver the component through Update Tool.

The following topics are addressed here:

- [“Packaging an Add-On Component” on page 71](#)
- [“Integrating an Add-On Component With Enterprise Server” on page 72](#)
- [“Delivering an Add-On Component Through Update Tool” on page 72](#)

Packaging an Add-On Component

To enable an add-on component to plug in to the Enterprise Server kernel in the same way as other components, package the component as an OSGi bundle.

A bundle is the unit of deployment in the OSGi module management subsystem. To package a component as an OSGi bundle, package the component's constituent files in a Java archive (JAR) file with appropriate manifest entries. The manifest entries provide information about the component that is required to enable the component to be plugged into the Enterprise Server kernel, such as:

- Name
- Version
- Dependencies
- Capabilities

Integrating an Add-On Component With Enterprise Server

Integrating an add-on component with Enterprise Server enables Enterprise Server to discover the component at runtime. To integrate an add-on component with Enterprise Server, ensure that the JAR file that contains the component is copied to or installed in the *as-install/modules/* directory.

Delivering an Add-On Component Through Update Tool

If an add-on component is an extension or update to existing installations of Enterprise Server, deliver the component through Update Tool. To deliver an add-on component through Update Tool, create an Image Packaging System (IPS) package to contain the component and add the package to a suitable IPS package repository.

For information about how to create IPS packages, see the [IPS best practices document](http://wikis.sun.com/display/IpsBestPractices/) (<http://wikis.sun.com/display/IpsBestPractices/>).

Integration Point Reference

This appendix provides reference information about integration points, which are described in Chapter 3, “Extending the Administration Console.” For complete information about integration points, see <http://wiki.glassfish.java.net/Wiki.jsp?page=V3IntegrationPoint>.

Define an integration point for each user interface feature in the `console-config.xml` file for your add-on component.

The following topics are addressed here:

- “Integration Point Attributes” on page 73
- “`org.glassfish.admingui:navNode` Integration Point” on page 74
- “`org.glassfish.admingui:rightPanel` Integration Point” on page 75
- “`org.glassfish.admingui:rightPanelTitle` Integration Point” on page 76
- “`org.glassfish.admingui:serverInstTab` Integration Point” on page 76
- “`org.glassfish.admingui:commonTask` Integration Point” on page 77
- “`org.glassfish.admingui:configuration` Integration Point” on page 77
- “`org.glassfish.admingui:resources` Integration Point” on page 78
- “`org.glassfish.admingui:customtheme` Integration Point” on page 78
- “`org.glassfish.admingui:masthead` Integration Point” on page 79
- “`org.glassfish.admingui:loginimage` Integration Point” on page 79
- “`org.glassfish.admingui:loginform` Integration Point” on page 80
- “`org.glassfish.admingui:versioninfo` Integration Point” on page 80

Integration Point Attributes

For each integration-point element, specify the following attributes. Each attribute takes a string value.

id

An identifier for the integration point. The remaining sections of this appendix do not provide details about specifying this attribute.

parentId

The ID of the integration point's parent.

type

The type of the integration point.

priority

A numeric value that specifies the relative ordering of integration points with the same parentId. A lower number specifies a higher priority (for example, 100 represents a higher priority than 400). You may need to experiment in order to place the integration point where you want it. This attribute is optional.

content

A relative path to the JavaServer Faces page that contains the content to be integrated. Typically, the file contains a JavaServer Faces code fragment that is incorporated into a page. The code fragment often specifies a link to another JavaServer Faces page that appears when a user clicks the link.

org.glassfish.admingui:navNode **Integration Point**

Use an org.glassfish.admingui:navNode integration point to insert a node in the Administration Console navigation tree. Specify the attributes and their content as follows.

type

org.glassfish.admingui:navNode, the left-hand navigation tree

parentId

The id value of the navNode that is the parent for this node. The parentId can be any of the following:

tree

The root node of the entire navigation tree. Use this value to place your node at the top level of the tree. You can then use the id of this node to create additional nodes beneath it.

registration

The Registration node

applicationServer

The Enterprise Server node

applications

The Applications node

resources

The Resources node

configuration

The Configuration node

webContainer

The Web Container node under the Configuration node

httpService

The HTTP Service node under the Configuration node

Note – The `webContainer` and `httpService` nodes are available only if you installed the web container module for the Administration Console (the `console-web-gui.jar` OSGi bundle).

priority

A numeric value that specifies the relative ordering of the node on the tree, whether at the top level or under another node.

content

A relative path to the JavaServer Faces page that contains the content to be integrated, or a URL to an external resource that returns the appropriate data structure for inclusion.

For an example, see [Example 3–2](#).

org.glassfish.admingui:rightPanel **Integration Point**

Use an `org.glassfish.admingui:rightPanel` integration point to specify content for the right frame of the Administration Console. Specify the attributes and their content as follows.

type

`org.glassfish.admingui:rightPanel`

parentId

None.

priority

A numeric value that specifies the relative ordering. If multiple plug-ins specify content for the right frame, the one with greater priority will take precedence.

content

A path relative to the root of the plug-in JAR file to a file containing the content for the right panel. Alternatively, it may contain a full URL which will deliver the content for the right panel.

org.glassfish.admingui:rightPanelTitle **Integration Point**

Use an `org.glassfish.admingui:rightPanel` integration point to specify the title for the right frame of the Administration Console. Specify the attributes and their content as follows.

`type`

`org.glassfish.admingui:rightPanelTitle`

`parentId`

None.

`priority`

A numeric value that specifies the relative ordering. If multiple plug-ins specify content for the right frame, the one with greater priority will take precedence.

`content`

Specifies the title to display at the top of the right panel.

org.glassfish.admingui:serverInstTab **Integration Point**

Use an `org.glassfish.admingui:serverInstTab` integration point to place an additional tab on the Enterprise Server page of the Administration Console. Specify the attributes and their content as follows.

`type`

`org.glassfish.admingui:serverInstTab`

`parentId`

The `id` value of the tab set that is the parent for this tab. For a top-level tab on this page, this value is `serverInstTabs`, the tab set that contains the general information property pages for Enterprise Server.

For a sub-tab, the value is the `id` value for the parent tab.

`priority`

A numeric value that specifies the relative ordering of the tab on the page, whether at the top level or under another tab.

`content`

A relative path to the JavaServer Faces page that contains the content to be integrated.

When you use this integration point, your JavaServer Faces page must call the `setSessionAttribute` handler for the command event to set the session variable of the `serverInstTabs` tab set to the `id` value of your tab. For example, the file may have the following content:

```
<sun:tab id="sampleTab" immediate="true" text="Sample First Tab">
  <!command
    setSessionAttribute(key="serverInstTabs" value="sampleTab");
    gf.redirect(page="#{request.contextPath}/page/tabPage.jsf?name=Sample%20First%20Tab");
  />
</sun:tab>
```

The id of the sun:tab custom tag must be the same as the value argument of the setSessionAttribute handler.

For examples, see [Example 3-4](#) and [Example 3-5](#).

org.glassfish.admingui:commonTask Integration Point

Use an org.glassfish.admingui:commonTask integration point to place a new task or task group on the Common Tasks page of the Administration Console. Specify the attributes and their content as follows.

type

org.glassfish.admingui:commonTask

parentId

If you are adding a task group, the id value of the Common Tasks page, which is commonTasksSection.

If you are adding a single task, the id value of the task group that is the parent for this tab, such as deployment (for the Deployment group).

priority

A numeric value that specifies the relative ordering of the tab on the page, whether at the top level or under another tab.

content

A relative path to the JavaServer Faces page that contains the content to be integrated.

For examples, see [Example 3-7](#) and [Example 3-9](#).

org.glassfish.admingui:configuration Integration Point

Use an org.glassfish.admingui:configuration integration point to add a component to the Configuration page of the Administration Console. Typically, you add a link to the property sheet section of this page. Specify the attributes and their content as follows.

type

org.glassfish.admingui:configuration

parentId

The id value of the property sheet for the Configuration page. This value is `propSheetSection`, the section that contains the property definitions for the Configuration page.

priority

A numeric value that specifies the relative ordering of the item on the Configuration page.

content

A relative path to the JavaServer Faces page that contains the content to be integrated.

org.glassfish.admingui:resources **Integration Point**

Use an `org.glassfish.admingui:resources` integration point to add a component to the Resources page of the Administration Console. Typically, you add a link to the property sheet section of this page. Specify the attributes and their content as follows.

type

`org.glassfish.admingui:resources`

parentId

The id value of the property sheet for the Resources page. This value is `propSheetSection`, the section that contains the property definitions for the Resources page.

priority

A numeric value that specifies the relative ordering of the item on the Resources page.

content

A relative path to the JavaServer Faces page that contains the content to be integrated.

For an example, see [Example 3-11](#).

org.glassfish.admingui:customtheme **Integration Point**

Use an `org.glassfish.admingui:customtheme` integration point to add your own branding to the Administration Console. Specify the attributes and their content as follows. Do not specify a `parentId` attribute for this integration point.

type

`org.glassfish.admingui:customtheme`

priority

A numeric value that specifies the relative ordering of the item in comparison to other themes. This value must be between 1 and 100. The theme with the smallest number is used first.

content

The name of the properties file that contains the key/value pairs that will be used to access your theme JAR file. You must specify the following keys:

`com.sun.webui.theme.DEFAULT_THEME`

Specifies the theme name for the theme that this application may depend on.

`com.sun.webui.theme.DEFAULT_THEME_VERSION`

Specifies the theme version this application may depend on.

For example, the properties file for the default Administration Console brand contains the following:

```
com.sun.webui.theme.DEFAULT_THEME=suntheme
com.sun.webui.theme.DEFAULT_THEME_VERSION=4.3
```

For an example, see [Example 3-14](#).

org.glassfish.admingui:masthead Integration Point

Use an `org.glassfish.admingui:masthead` integration point to specify the name and location of the include masthead file, which can be customized with a branding image. This include file will be integrated on the masthead of the Administration Console. Specify the attributes and their content as follows. Do not specify a `parentId` attribute for this integration point.

type

`org.glassfish.admingui:masthead`

priority

A numeric value that specifies the relative ordering of the item in comparison to other items of this type. This value must be between 1 and 100. The theme with the smallest number is used first.

content

A file that contains the content, typically a file that is included in a JavaServer Faces page.

For an example, see [Example 3-15](#).

org.glassfish.admingui:loginimage Integration Point

Use an `org.glassfish.admingui:loginimage` integration point to specify the name and location of the include file containing the branding login image code that will be integrated with the login page of the Administration Console. Specify the attributes and their content as follows. Do not specify a `parentId` attribute for this integration point.

type

`org.glassfish.admingui:loginimage`

parentId

None; a login image does not have a parent ID.

priority

A numeric value that specifies the relative ordering of the item in comparison to other items of this type. This value must be between 1 and 100. The theme with the smallest number is used first.

content

A file that contains the content, typically a file that is included in a JavaServer Faces page.

For an example, see [Example 3–15](#).

org.glassfish.admingui:loginform **Integration Point**

Use an `org.glassfish.admingui:loginform` integration point to specify the name and location of the include file containing the customized login form code. This code also contains the login background image used for the login page for the Administration Console. Specify the attributes and their content as follows. Do not specify a `parentId` attribute for this integration point.

type

`org.glassfish.admingui:loginform`

priority

A numeric value that specifies the relative ordering of the item in comparison to other items of this type. This value must be between 1 and 100. The theme with the smallest number is used first.

content

A file that contains the content, typically a file that is included in a JavaServer Faces page.

For an example, see [Example 3–15](#).

org.glassfish.admingui:versioninfo **Integration Point**

Use an `org.glassfish.admingui:versioninfo` integration point to specify the name and location of the include file containing the branding image that will be integrated with the content of the version popup window. Specify the attributes and their content as follows. Do not specify a `parentId` attribute for this integration point.

type

`org.glassfish.admingui:versioninfo`

priority

A numeric value that specifies the relative ordering of the item in comparison to other items of this type. This value must be between 1 and 100. The theme with the smallest number is used first.

content

A file that contains the content, typically a file that is included in a JavaServer Faces page.

For an example, see [Example 3-15](#).

Index

A

- abbrev_product_name keyword, 58
- acceptableValues element
 - @Param annotation, 52
- add-on components
 - delivering, 72
 - integrating, 72
 - overview, 15-16
 - packaging, 71
 - specifying ID values, 30-31
- AdminCommand interface, 48
 - execute method, 57
- AdminCommandContext class, 57
- Administration Console
 - adding content to pages, 40-42
 - adding functionality to, 31-42
 - adding internationalization support, 43
 - adding nodes to navigation tree, 32-34
 - adding pages to, 42
 - adding tabs and tab sets to pages, 34-37
 - adding task groups to Common Tasks page, 38-40
 - adding tasks to Common Tasks page, 37-38
 - architecture, 28-29
 - changing theme or brand of, 43-45
 - extending, 27-46
- annotations
 - @Param, 51
 - @I18n, 54
 - @Scoped, 49
 - @Service, 48
- Apache Felix OSGi framework, 16
- Apache Maven, *See* Maven

- Archive interface, 66-68
- archive types, *See* containers: archive types
- ArchiveHandler interface, 67-68
- asadmin command
 - branding, 57-58
 - context, 57
- asadmin commands
 - adding, 48
 - default parameter values, 52
 - error messages, 54-56
 - internationalization, 54
 - naming, 48
 - operands, 50-54
 - options, 50-54
 - parameters, 50-54
 - running, 57
 - strings, 54-56
 - text, 54-56
 - validation of parameters, 52

B

- brand of Administration Console, changing, 43-45
- branding, asadmin command, 57-58
- BrandingVersion.properties file, 57-58
- build_id keyword, 58

C

- class loaders, 67-68

classes

- AdminCommandContext, 57
- stateless, 49
- CLI (command-line interface), branding, 57-58
- command-line interface (CLI), branding, 57-58
- ComponentManager class, 21
- components, instantiating, 21
- connector modules, *See* containers: connector modules
- Console Add-On Component Service, 28-29
- console-config.xml file, 30, 73-81
 - console-config element, 30-31
 - integration-point element, 30-31
- console providers, 28-29
 - implementing, 28-29
- ConsoleProvider interface, 28-29
- Container interface, 63-66
- containers
 - archive types, 66-68, 68-70, 70
 - connector modules, 68-70
 - developing, 63-70
 - examples, 65
 - implementing, 64-66
 - loading, 68-70, 70
 - naming, 63-64
- content attribute, integration-point element, 73-74
- context, asadmin command, 57
- @Contract annotation, 20, 25-26
- conventions, asadmin command names, 48

D

- default values, asadmin command parameters, 52
- delivering, add-on components, 72
- Deployer interface, 64-66, 68-70
- deployment, examples, 65-66

E

- error messages, asadmin commands, 54-56
- examples
 - containers, 65
 - deployers, 65-66
- execute method, AdminCommand interface, 57

- @Extract annotation, 23
- extraction, 23

F

- Felix OSGi framework, 16
- fields, representation of command parameters as, 50
- files, BrandingVersion.properties, 57-58

G

- getConfiguration method, 28-29

H

- Habitat, class, 23
- HK2, scopes, 20-21
- HK2 (Hundred-Kilobyte Kernel)
 - architecture, 19
 - overview, 16
 - services, 20
- Hundred-Kilobyte Kernel (HK2)
 - architecture, 19, 22-24
 - extraction, 23
 - injection, 22-23
 - instantiating, 21
 - instantiation, 24
 - inversion of control, 22-24
 - lifecycle, 21-22
 - overview, 16
 - runtime, 20-22
 - services, 20

I

- @I18n annotation, 54
- id attribute, integration-point element, 73-74
- Image Packaging System (IPS), 72
- @Inject annotation, 22-23, 24
- injection, 22-23
- instantiation, 24

integrating, add-on components, 72
 integration-point element, 30-31
 attributes, 31-42, 73-74
 integration points, 28-29, 30
 attributes, 73-74
 creating types, 45-46
 org.glassfish.admingui:commonTask, 37-38,
 38-40, 77
 org.glassfish.admingui:configuration, 40-42,
 77-78
 org.glassfish.admingui:customtheme, 43-45,
 78-79
 org.glassfish.admingui:loginform, 80
 org.glassfish.admingui:loginimage, 79-80
 org.glassfish.admingui:masthead, 79
 org.glassfish.admingui:navNode, 74-75
 org.glassfish.admingui:resources, 40-42, 78
 org.glassfish.admingui:rightPanel, 75
 org.glassfish.admingui:rightPanelTitle, 76
 org.glassfish.admingui:serverInstTab, 34-37,
 76-77
 org.glassfish.admingui:treeNode, 32-34
 org.glassfish.admingui:versioninfo, 80-81
 reference, 73-81
 interfaces
 AdminCommand, 48, 57
 internationalization
 asadmin commands, 54
 providing for add-on components, 43
 IPS (Image Packaging System), 72

J

JSFTemplating project, templates, 29
 JSFTemplating tags
 sun:commonTask, 38
 sun:commonTasksGroup, 39-40
 sun:property, 41-42
 sun:tab, 36-37
 sun:treeNode, 33-34

L

lifecycle interfaces, 21-22
 long form, option names, 52

M

major_version keyword, 58
 Maven, 25-26
 methods, AdminCommand, 57
 minor_version keyword, 58
 modular architecture, Enterprise Server, 15-16

N

name element
 @Param annotation, 52
 @Service annotation, 48
 names, asadmin commands, 48
 navigation nodes, adding to Administration
 Console, 32-34

O

operands, asadmin commands, 50-54
 optional element, @Param annotation, 53
 options
 asadmin commands, 50-54
 long names, 52
 short names, 52
 org.glassfish.admingui:commonTask integration
 point type, 37-38, 38-40, 77
 org.glassfish.admingui:configuration integration
 point type, 40-42, 77-78
 org.glassfish.admingui:customtheme integration
 point type, 43-45, 78-79
 org.glassfish.admingui:loginform integration
 point type, 80
 org.glassfish.admingui:loginimage integration
 point type, 79-80
 org.glassfish.admingui:masthead integration point
 type, 79

org.glassfish.admingui:navNode integration point type, 74-75
org.glassfish.admingui:resources integration point type, 40-42, 78
org.glassfish.admingui:rightPanel integration point type, 75
org.glassfish.admingui:rightPanelTitle integration point type, 76
org.glassfish.admingui:serverInstTab integration point type, 34-37, 76-77
org.glassfish.admingui:treeNode integration point type, 32-34
org.glassfish.admingui:versioninfo integration point type, 80-81
OSGi Alliance, 16
overview
 add-on components, 15-16
 extensibility, 15-16

P

packaging, add-on components, 71
pages, adding to Administration Console, 42
pages of Administration Console, adding content to, 40-42
@Param annotation, 51
parameters
 asadmin commands, 50-54
 default values, 52
 validation of, 52
parentId attribute, integration-point element, 73-74
plug-ins, *See* add-on components
PostConstruct interface, 21-22
PreDestroy interface, 21-22
primary element, @Param annotation, 51
priority attribute, integration-point element, 73-74
product name, defining, 57-58
product_name keyword, 58
properties, representation of command parameters as, 50

R

ReadableArchive interface, 66-68
release information, defining, 57-58
running, asadmin commands, 57

S

@Scoped annotation, 20-21
@Scoped annotation, 49
@Service annotation, 20, 25-26, 63-66, 65
@Service annotation, 48
setter methods, command parameters and, 50
short form, option names, 52
shortName element, @Param annotation, 52
singletons, 20-21
Sniffer interface, 68-70, 70
sniffers, 68-70, 70
stateless classes, 49
strings, asadmin commands, 54-56
sun:commonTask tag, 38
sun:commonTasksGroup tag, 39-40
sun:property tag, 41-42
sun:tab tag, 36-37
sun:treeNode tag, 33-34

T

tabs and tab sets, adding to Administration Console, 34-37
task groups, adding to Administration Console, 38-40
tasks, adding to Administration Console, 37-38
Templating for JavaServer Faces Technology, *See* JSFTemplating project
text, asadmin commands, 54-56
theme of Administration Console, changing, 43-45
type attribute, integration-point element, 73-74

U

Update Tool, 72

V

validation, `asadmin` command parameters, 52
version_prefix keyword, 58
version_suffix keyword, 58

W

WritableArchive interface, 66-68

