WebSphere® Studio Device Developer

IBM

SMF Bundle Developer Tools User’s Guide

First Edition (November 2003)

This edition applies to Service Management Framework Bundle Developer Tools, 5.5.2. This edition applies to all subsequent releases and modifications until otherwise indicated in new releases.

Contents

Chapter 1. Getting started with Service Management Framework Bundle Developer Tools

The information in this chapter guides you through the process of creating a sample bundle with Service Management Framework (SMF) Bundle Developer. After you create a sample bundle, refer to Chapter 3, "Tasks," on page 17 for instructions on how to create your own bundle.

The sample bundle you create includes the following files:

Note: To access the custom editors that SMF Bundle Developer Tools provides for these files, double-click on the file to open the editor for that file.

- `MANIFEST.MF`

  The imported and exported packages and services, native code clauses, and other bundle details reside in the `MANIFEST.MF` file. SMF Bundle Developer Tools provide a Manifest editor with a graphical user interface that you can use to modify this file.
- `IVEATTRS.XML`

  The target configuration information for the bundle resides in the `IVEATTRS.XML` file. SMF Bundle Developer Tools provide a Bundle Attributes editor with a graphical user interface that you can use to modify this file.
- `IVERES.XML`

  The bundle resource requirements reside in the `IVERES.XML` file. SMF Bundle Developer Tools provide a Bundle Resources editor with a graphical user interface that you can use to modify this file.

Understanding the directory terminology

Throughout this document, `IVEHOME` refers to the directory within the J9 tree which contains the appropriate version of the J9 VM for your operating system and platform (for example, `C:\Program Files\IBM\DeviceDeveloper\wsdd5.0\ive-2.2\runtimes\win32\x86`).

`TECHHOME` refers to the technologies subdirectory within the WebSphere Studio Device Developer directory tree (for example, `C:\Program Files\IBM\DeviceDeveloper\wsdd5.0\technologies`).

Notes:

1. Set the `IVEHOME` environment variable so that it points to one of the following, depending on your environment (where `<wsdd install>` is the WSDD installation directory):
   - `<wsddinstall>\wsdd5.0\ive-2.2\runtimes\win32\x86`
   - `<wsdd install>\wsdd5.0\ive-2.2\runtimes\linux\x86`

   Most of the Service Management Framework scripts depend on this variable and will remind you to set it if it is not already set. `TECHHOME` is not an environment variable; however, this documentation uses it as a reference.
2. WebSphere® Studio Device Developer continues to use `wsdd5.0` for its primary directory tree even though the current release level for WebSphere Studio Device Developer is higher than 5.0.

3. This documentation uses Windows directory format for all of the directory names in this documentation.

Verifying that you have the prerequisites

Before you can run the SMF Bundle Developer Tools, verify that your development machine contains the following prerequisites:

- WebSphere Studio Device Developer version 5.5 for SMF Bundle Developer Tools 5.5.1, WebSphere Studio Device Developer version 5.6 for SMF Bundle Developer Tools 5.6, or WebSphere Studio Device Developer version 5.6 for SMF Bundle Developer Tools 5.7

  The default development environment for SMF Bundle Developer Tools is Microsoft® Windows® 2000 and WebSphere Studio Device Developer 5.7.
- At least one of the following class libraries in the `IVEHOME\lib` directory:
  - `Custom Gateway Plus Library (jclRM)`
  - `Custom Resource Managed Gateway Plus Library (jclRM)`
  - `Custom Max Library (jclMax)`
  - `J2ME Foundation Library (jclFoundation)`

This documentation assumes you are using a new workbench installation with all the default settings and preferences.

Creating a sample SMF project

Refer to the following instructions to create a sample SMF project that includes the SMF Pizza Sample Bundle:

1. Open the SMF perspective by selecting **Window**->**Open Perspective**->**Other**->**SMF** from the menu bar.

   The SMF perspective contains additional toolbar buttons and views that you can use when you create bundles.
2. Click **New** in the workbench toolbar.
3. In the left pane, select **Examples** -> **SMF**, and in the right pane, select **Pizza Bundle**. Then click **Next**.
4. Click **Finish**.

   The SMF perspective displays the Pizza Bundle in the Package Explorer view.

Starting and configuring an SMF Bundle Server

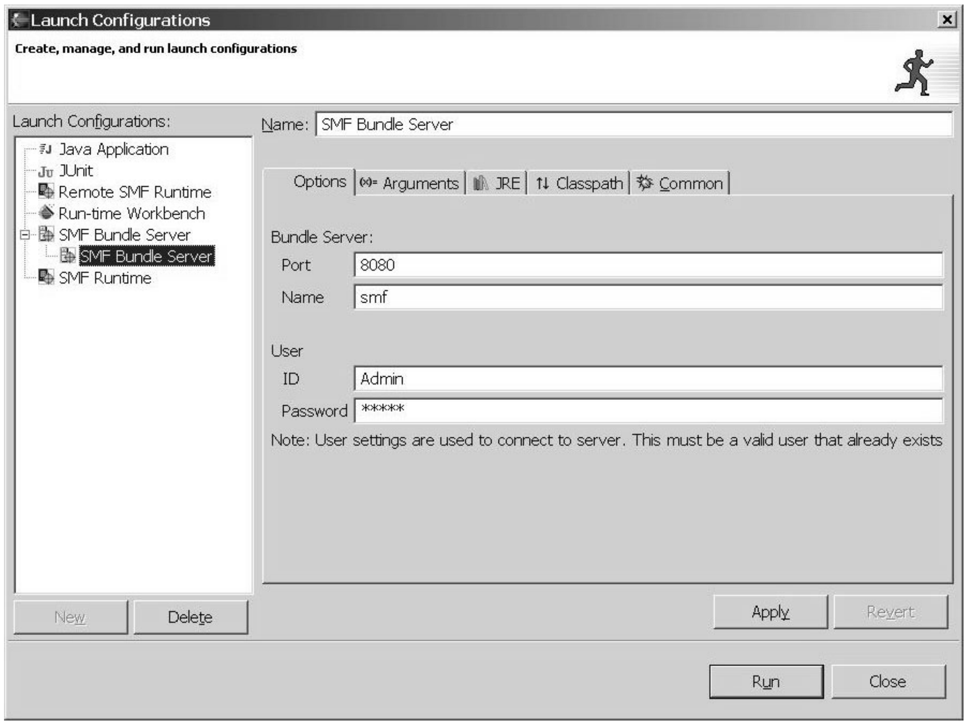
You can launch an SMF Bundle Server from the command line or from the **SMF Bundle Developer Launch Configurations** dialog. For information on launching an SMF Bundle Server from the command line, refer to the Service Management Framework Server User's Guide.

Use the following instructions to launch the SMF Bundle Server from WebSphere Device Developer:

1. On the main menu, click **Run**->**Run**.

   The Launch Configurations dialog opens, and a list of launch configurations displays on the left. Refer to the following image for a depiction of the Launch

Configurations dialog.



Name: SMF Bundle Server

Bundle Server:
Port 8080
Name smf

User
ID Admin
Password *****

Note: User settings are used to connect to server. This must be a valid user that already exists

2. From the list of launch configurations, select **SMF Bundle Server** and click **New**.

   SMF Bundle Developer Tools add a new configuration under SMF Bundle Server and displays the configuration options tabs.

   Note: In most cases, you can accept all of the default values on all of the tabs.
3. Click **Run**.

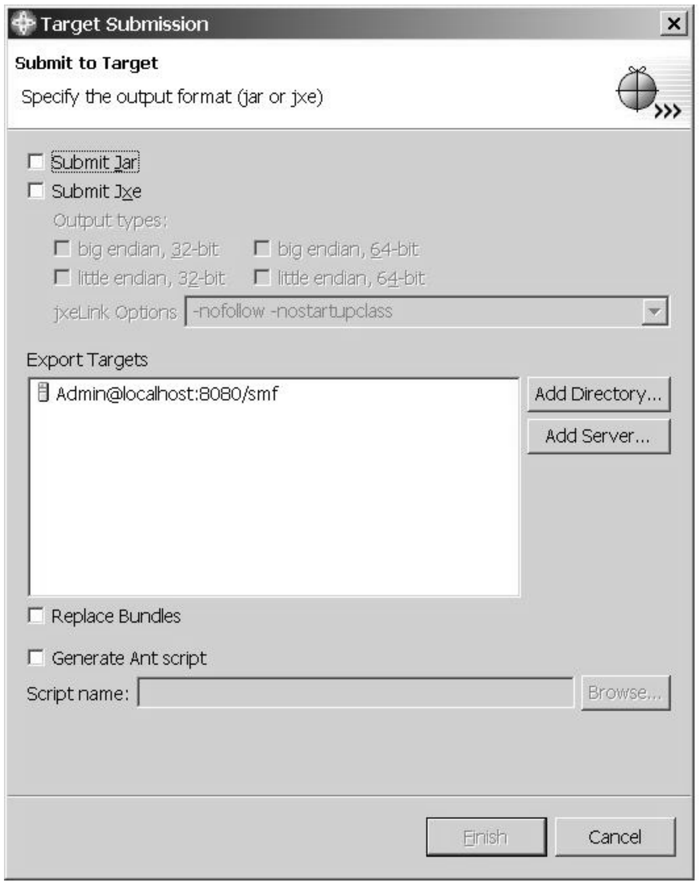
   The Console view displays startup messages for the bundle server. You can open the SMF Bundle Server view to view the server you just created.

Submitting a bundle to the bundle server

Use the following steps to submit the sample bundles you created in "Creating a sample SMF project" on page 2 to the bundle server.

1. From the Packages Explorer view, right-click **Pizza Bundle** and select **SMF -> Submit Bundle...** from the context menu.

The **Target Submission** wizard displays. Refer to the following image for a depiction of this wizard.


Target Submission

Submit to Target

Specify the output format (jar or jxe)

☐ Submit Jar

☐ Submit Jxe

Output types:

☐ big endian, 32-bit ☐ big endian, 64-bit

☐ little endian, 32-bit ☐ little endian, 64-bit

jxeLink Options: -nofollow -nostartupclass

Export Targets

Admin@localhost:8080/smf

Add Directory...

Add Server...

☐ Replace Bundles

☐ Generate Ant script

Script name:

Browse...

Finish

Cancel

2. On the Target Submission wizard, complete the following:
   a. Select the **Submit Jxe** checkbox.
   b. Select the **Submit Jar** checkbox.
   c. Select **little endian, 32-bit** as the output type.
   d. In the **Export Targets** list, select the server running on the localhost, which you created in "Starting and configuring an SMF Bundle Server" on page 2.

      Note: **Finish** does not activate until you select a server in the **Export Targets** list.
   e. Select the **Replace Bundles** box.

      Note: If the bundle already exists on the bundle server, this option notifies the bundle server to overwrite the existing bundle with the new submission.
3. Click **Finish**.
4. Click on the plus sign (+) next to the bundle server that represents your localhost server.
5. Click on the plus sign (+) next to **Bundles** to expand the list of bundles on the localhost server and verify that your bundles are in the list.

   Note: Several versions of your bundle might exist on the bundle server. To view information about each version available on the server, right-click on your sample bundle in the SMF Bundle Servers view, and select **Show Versions** from the context menu. The Bundle Details view provides information about the different versions of the bundle.

Configuring and starting the runtime and installing a bundle

Use the following instructions to start the SMF Runtime Framework.

1. In the SMF perspective, select **Run** ->**Run...** from the menu bar.

   The Launch Configurations dialog displays.
2. Select **SMF Runtime** in the **Launch Configuration** list on the left and click **New**.

   Service Management Framework Bundle Developer populates the Launch Configuration dialog with the SMF Runtime default values.
3. Click **Run**.

   SMF Bundle Developer Tools launch the runtime and loads `SMF System Bundle` in the SMF Runtime view.
4. In the SMF Bundle Servers view, right-click **PizzaServletSample** and select **Install** from the context menu.

   The bundle and its prerequisites display in the SMF Runtime view after the transfer completes successfully.
5. To access the `Pizza Servlet Sample`, which is the sample bundle you created in "Creating a sample SMF project" on page 2, open a browser window and navigate to `http://localhost/pizza`.

   Refer to "Sample applications" on page 43 for descriptions of the other sample bundles you can install.

Uninstalling a bundle and closing the runtime

After you finish testing your bundle on the runtime, use the following steps to uninstall the bundle from the runtime and close the runtime:

1. In the SMF Bundle Servers view, right-click **PizzaServletSample** and select **Uninstall** from the context menu.

   The bundle is uninstalled.

   Notes:

   a. If you prefer to uninstall the bundle and any bundles that are no longer used or needed as dependants, you can select **UninstallGC**, uninstall with garbage collect, instead of **Uninstall**. Uninstall with bundle garbage collect is only available when the DependencyAdmin bundle is installed.

   b. If the bundle you are uninstalling is part of a snapshot, the **Uninstall** menu option will be disabled. You must first delete the snapshot containing the bundle before you can uninstall it.
2. Select **Terminate Runtime** from the drop-down menu in the SMF Runtime view.

Chapter 2. Concepts

Service Management Framework (SMF) is an implementation of the OSGi Alliance Service Platform Release 3 specification. The OSGi Alliance is an industry group that defines and promotes an open standard for the networked delivery of managed services to local networks and devices. Open standards enable different manufacturers to adhere to the same set of standards, which minimizes the number of products that are incompatible. The Open Services Gateway standard complements other residential standards and is open to most other protocols and transport and device layers.

OSGi™ specification contents

The OSGi™ Service Platform Release 3 specification defines a Framework on which applications can run. Developers can write new applications and adapt existing applications to run on the Framework. The Framework enables operators to deploy multiple applications on a single Java™ Virtual Machine (JVM). Application developers partition applications into services and other resources. Services and resources are packaged into bundles, which are files that serve as the delivery unit for applications. Bundles have manifests with special headers that enable you to share classes and services at the package level.

The OSGi specification defines a set of services for application bundles to use. Of those services, Service Management Framework implements the following:

- `Configuration Admin Service` - enables an operator to set the configuration information of deployed bundles.
- `Device Access` - supports automatic detection of attached and detached hardware devices and can automatically download and start appropriate device drivers.
- `HTTP Service` - provides an embedded HTTP server that is capable of serving HTML and servlets.
- `Log Service` - provides a general purpose message logger for the OSGi environment.
- `Package Admin Service` - enables a management bundle to provide the policies for package sharing.
- `Permission Admin Service` - enables a management bundle to administer bundle's permissions and provides defaults for all bundles.
- `Preferences Service` - provides a persistent data store for bundles.
- `User Admin Service` - provides minimum authentication functionality.
- `Start Level Service` - enables a management agent to control the relative starting and stopping order of bundles.
- `URL Handlers Service Support` - enables you to dynamically register multiple URL and Content handler services, which shield bundles from Java limitations.

Service Management Framework Bundle Developer does not implement the following OSGi Release 3 specifications:

- `IO Connector Service`
- `Wire Admin Service`
- `Namespace`

- `Jini`
- `UPnP`
- `Initial Provisioning`

Service Management Framework

IBM® Service Management Framework (SMF) is a production-ready software management framework for network-delivered applications, that meets the needs of Internet-ready device manufacturers and service providers, such as cable companies and power utilities.

IBM packages the Service Management Framework (SMF) in several ways. The SMF Toolkit contains the SMF Runtime, Web Application Bundle Tool, samples and documentation. It also includes command line and Web-based interfaces. You can download the SMF Toolkit for free from http://www.ibm.com/embedded.

The SMF Bundle Development Kit is an Eclipse feature that you can install in WebSphere Studio Device Developer (WSDD). It contains the SMF Runtime, SMF Bundle Server, and SMF Bundle Developer Tools. The SMF Bundle Server and SMF Bundle Developer Tools provide a graphical user interface that you can use to develop bundles.

SMF Bundle Development Kit components

In addition to providing features of the OSGi™ Service Platform Release 3 Specification definition, Service Management Framework (SMF) Bundle Developer provides application development, deployment, and management features.

The SMF Runtime feature enables you to launch SMF from the command line and manipulate it with the SMFAdmin Web-based administration tool. The SMF Bundle Server enables you to launch the server from the command line and administer it with the Web-based administration tool. The SMF Bundle Developer Tools enable you to use the specialized editors to develop bundles.

SMF Runtime

The SMF Runtime contains the core SMF framework implementation. In addition, it contains features that enable you to manage the runtime with SMF Bundle Developer Tools and SMF Bundle Server. An SMF Runtime can request and receive updates from an SMF Bundle Server through the graphical user interface of the SMF Bundle Developer Tools, through the SMF Runtime command line interface, or through the SMF Bundle Server Web Interface.

SMF Bundle Server

The SMF Bundle Server manages the storage and deployment of bundles in a heterogeneous network of devices. Through the `Safe Bundle Install` protocol, the SMF Bundle Server delivers a bundle to a device when the runtime device requests the bundle or when a system administrator instigates a remote distribution request. ″Target″ bundle refers to the bundle that is requested by the runtime or remotely installed by an administrator.

The `Safe Bundle Install` protocol enables the runtime and server to exchange information about the installation and configuration. The server uses the installation and configuration information to assess the capabilities, capacity, and current running state of the device. As a result, the server delivers the appropriate

″target″ bundle, identifies any prerequisite bundles that the target bundle needs, and directs the prerequisite bundles to download to the device.

The server uses the properties in the following list to differentiate between multiple instances of a ″named″ bundle in the server repository. As a result, the server only delivers the target bundle that is compatible with and can run on the target device.

- `Processor Type`
- `Operating System`
- `Operating System Version`
- `Java Virtual Machine`
- ` Virtual Machine`
- `Address Length`

During bundle installation the server might conduct other types of verification, such as verifying that the proper resources are available and that the runtime authorizes the download. These verification tests ensure that the server does not deliver a bundle that leaves the runtime device in a state that cannot run. In addition, the verification tests eliminate the potentially high cost of downloading bundles that are not needed by the runtime device and the inability of the bundle to run due to conditions such as insufficient resources or access restrictions.

SMF Bundle Developer Tools

The SMF Bundle Developer Tools feature provides an integrated development environment (IDE) for developing, testing, and deploying OSGi™ bundles. In addition, the feature provides Wizards, Views, and Editors that you can use to create bundles and an SMF Perspective for observing and managing an SMF Bundle Server and SMF Runtime.

OSGi™ concepts

This section contains information on OSGi™ concepts.

Framework

The Framework is the core of the OSGi™ implementation. It manages bundle installations and updates in an OSGi environment and dependencies between bundles and services.

Bundles

A bundle is the smallest unit of management for the Framework. Bundles are Java Archive (JAR) or J9 Executable (JXE) files with a manifest that contains special headers. These headers describe the bundle to SMF and list the bundle’s dependencies, such as the packages and services required by the bundle. Bundles can register services with SMF that other bundles can use.

Like other JAR files, JAR bundle files typically contain Java class files and resources. JXE bundle files do not contain individual Java class files. Instead the class files are pre-linked into a single file named rom.classes. Storing and executing JXE files in read only memory (ROM) reduces the amount of time needed to start the application and the amount of required random access memory (RAM).

The descriptive information in the manifest file differentiates bundles from other JAR files. Non-bundle JAR files often keep very little information in the manifest file. However, a bundle's manifest file usually contains descriptive information, such as the bundle's name and version, and a list of the packages and services it requires.

Bundle life cycle

The framework manages the life cycle of bundles. As you install and run a bundle, it goes through various states. The possible states of a bundle are:

- `INSTALLED` - the bundle has been installed, but all of the bundle's dependencies have not been met. The bundle requires packages that have not been exported by any currently installed bundle.
- `RESOLVED` - the bundle is installed, and its dependencies have been met, but it is not running. If a bundle is started and all of the bundle's dependencies are met, the bundle skips this state.
- `STARTING` - a temporary state that the bundle goes through while the bundle is starting.
- `ACTIVE` - the bundle is running.
- `STOPPING` - a temporary state that the bundle goes through while the bundle is stopping.
- `UNINSTALLED` - the bundle no longer exists in the framework.

Manifest files

Every bundle must contain a manifest file. The bundle's manifest file contains data that the framework needs to correctly install and activate the bundle. Service Management Framework Bundle Developer Tools contain a manifest editor that you can use to create and edit properly formed manifest files.

Service Management Framework follows these rules for manifest files:

- Service Management Framework Bundle Developer places the OSGi headers at the beginning of the manifest file.

  Note: SMF ignores any headers that SMF does not recognize.
- Service Management Framework Bundle Developer places each manifest file in the META-INF directory inside the bundle.
- Service Management Framework Bundle Developer names each manifest file `MANIFEST.MF`.

Note: Use capital letters for the manifest directory and file name. SMF will not find the MANIFEST.MF file if the file resides in the meta-inf directory instead of the META-INF directory.

Refer to the OSGi Service Platform Release 3 specification for more information about the OSGi manifest file format and syntax.

Services

Services decouple the provider from the service user. The only code a service provider and a service user share is the service definition. You can use Java interfaces to define services. Any class that implements this interface can provide the service.

Bundles that use services that are not provided by the bundle can notify the framework by including an Import-Service header in the bundle manifest. However, this is not required. When code within a bundle requests a provider of the service from the framework, the bundle imports the service at runtime.

A bundle that provides services can also include an Export-Service header in its manifest. When code within a bundle makes a provider available to the framework, the bundle exports the service at runtime.

The Import-Service and Export-Service tags inform the SMF Bundle Server of a bundle's prerequisites. Although these tags are optional, the prerequisite bundles will not install successfully if the tags are not defined.

Packages

Bundles can use code that is defined within other bundles by declaring the packages as imported packages in the manifest file. Although you can create a bundle that does not rely on any classes other than the Java base packages, most bundles import code from other bundles or the base runtime class path.

You must import any class that you use within a bundle that is not defined in the bundle or that is not a base Java class, meaning classes within packages that begin with `java.`. To import another class, include an import clause for the class's package in the bundle's manifest. You can explicitly import only whole packages; individual classes cannot be explicitly imported.

A bundle can make the classes the bundle defines available to other bundles by exporting packages. To enable other bundles to access a particular package, include an export clause for the package in the manifest of the bundle that contains the package.

Bundle activators

Bundles can include a class that implements the `org.osgi.framework.BundleActivator` interface. The Bundle-Activator header in the bundle's manifest file identifies the class to the framework. At startup time, the framework creates an instance of this class and calls its `start()` method. The activator can then publish services, start its own threads, and so on.

When the bundle shuts down, the framework calls the activator's `stop()` method. While the bundle shuts down, the activator can release resources obtained since the start method was called and revoke any services it has published.

Platform Profile

Typical Java applications build and run against a static classpath. In WebSphere Studio, you can specify this classpath through the project's Java Build Path properties. Under Service Management Framework (SMF), the base class libraries of the Java Runtime Environment (JRE) and the SMF bundles currently running in the framework are the packages available to an application's classloader. In other words, the JRE and the set of running bundles define the application's available classpath.

The Platform Profile provides a method for you to define the JRE used by the runtime environment, the build-time environment for an Extension Services project, and the set of bundles that can run on the platform.

A Platform Profile defines a set of Application Services. Application Services enable you to focus on the logical service requirements of the service instead of the requirements of the actual underlying bundles.

When you create an Extension Services bundle project, you select a Platform Profile and a set of Application Services for the project. SMF Bundle Developer Tools updates the Java Build Path for the project to reflect the Java Runtime Environment (JRE) of the platform, based on the Platform Profile you selected. Extension Services places the bundles for the Application Services you selected in the Java Build Path of the project. “Application Profile” refers to the Platform Profile and the set of Application Services you selected for the project.

The Application Tools provide Platform Profiles that represent the bundles available in the SMF runtime, for both WebSphere Everyplace Micro Environment Foundation and WebSphere Everyplace Custom Environment Max JCLs. The Application Tools also provide an Eclipse Default JRE platform profile that uses the currently defined Eclipse Default JRE. For details on the services provided by these bundles, refer to the Service Management Framework documentation.

Application Profile

Application Profile refers to the Platform Profile and the set of Application Services that you selected for the project. The Platform Profile and Application Services you select define the runtime and build time environment for the project. SMF Bundle Developer Tools can optionally update the Java Build Path for the project to reflect the Java Runtime Environment (JRE) of the platform, based on the Platform Profile you selected. SMF Bundle Developer Tools places the bundles for the Application Services you selected in the Java Build Path of the project.

Application Services

An Application Service represents a logical service, such as an XML parser or logging service, that consists of one or more bundles. Application Services enable you to focus on the logical requirements of the service instead of the requirements of the actual underlying bundles. The Applications Services and the Platform Profile you selected are referred to as the Application Profile.

SMF Bundle Developer Tools concepts

The SMF Bundle Developer Tools feature provides an integrated development environment (IDE) for developing, testing, and deploying OSGi™ bundles. The SMF Bundle Developer Tools add the SMF Perspective to the WebSphere Studio Device Developer workbench, which contains wizards, views, and editors that collectively provide an OSGi bundle developer with the tools needed to complete the following tasks:

- Identify bundle package and service imports and exports.
- Construct the OSGi manifest to include package and service imports and exports.
- Tag the bundle with device characteristics that enable the SMF Bundle Server to differentiate target devices.
- Submit bundles to the SMF Bundle Server for testing.
- Connect to any number of SMF Bundle Servers to view the contents of the server’s repository.
- Launch an SMF runtime from the Integrated Development Environment (IDE).
- Launch an SMF Bundle Server from the Integrated Development Environment (IDE).
- Connect and manage an existing SMF runtime located anywhere on the network.

Extension Services Bundle Project

An Extension Services bundle project contains a bundle and an associated Application Profile. Optionally, Extension Services bundle projects can:

- Automatically update the Java Build Path.

  SMF Bundle Developer Tools can automatically update the project's Java Build Path to reflect the project's Platform Profile and Application Services settings. Refer to "Platform Profile" on page 11 and "Application Services" on page 12 for more information.
- Provide a default bundle activator.

  SMF Bundle Developer Tools can create a default bundle activator class, `ExtensionServicesBundleActivator`. You can tailor the default bundle activator class by editing the source file for the class folder.
- Automatically update the Manifest file.

  SMF Bundle Developers Tools can automatically update the Manifest file in the Extension Services bundle project to contain appropriate OSGi metadata for the project. Extension Services manages or provides initial default values for the metadata fields by:
  - Setting `Bundle-Name` to the project name on project creation
  - Setting `Bundle-Version` to `1.0.0` on project creation
  - Setting `Bundle-Activator` to the default bundle activator if one was created
  - Updating `Import-Packages` to reflect the packages imported by the project's classes.
  - Setting `Import-Services` to include all of the services from the Application Services selected for the project. Refer to "Platform Profile" on page 11 for more information.

Bundle folders

A bundle folder is a source folder, meaning a folder on the project's classpath, that contains a `META-INF` directory. The `META-INF` directory contains a `MANIFEST.MF` file in which there is a Bundle-Name entry.

Bundle name

Within SMF Bundle Developer Tools, you can reference bundles by their location or by their Bundle-Name. The location is the fully qualified location where the bundle resides. It can be locally stored on the file system or remotely accessed using HTTP. Bundle-Name is a manifest header tag that enables you to provide a descriptive name for the bundle. For example, a locally stored bundle's Bundle-Name might be "Configuration Admin" and its location might be file:c:/bundles/cm.jar

Snapshots

Snapshots enable you to store the current state of the runtime for later use. For example, you might load all of the bundles needed on a particular target runtime and save the snapshot so that you can test different configurations. Because you used a snapshot to conduct your testing, you can return to the previous state when you finish.

Stations

Stations are Device IDs that the SMF Bundle Server uses to keep track of the devices it manages. System administrators manually assign station IDs. Station IDs can contain any unique number, such as the MAC address of a Network Interface Card.

Users

There are three types of users defined in the SMF Bundle Server: Administrator, Developer, and Runtime. When you create a user, you can grant one of the following privileges to that user:

- Administrator privileges enable the user to control all aspects of the bundle after you submit the bundle to the server. Administrators have access to all SMF functions including adding and removing servers, users, and stations.
- Developer privileges enable the user to create and submit bundles. Developers can also remove submitted bundles up until the time it has been released into production by an administrator. Developers cannot enable or disable bundles they have submitted to the server, or add or remove servers, stations, or users.
- Runtime privileges enable users to complete runtime tasks with the local SMF runtime. They have no privileges on the server.

Bundle storage implementations

You can configure an SMF Bundle Server to store its bundle repository in an SQL database (SQL-based) or system files (file-based). Both storage implementations are compatible with one another.

To determine which storage implementation is best for your environment, identify the number of users in your environment. Use the SQL configuration if a large number of developers will access the server or if the number of bundles in the repository exceeds the threshold of the file-based configuration.

Note: Currently, only Windows NT, Windows 2000, Windows XP and Linux support the SQL-based configuration.

Use the file-based configuration for individuals or small development teams because the threshold of the file-based configuration is between one hundred and one hundred and fifty bundles. Performance might lessen as the number of bundles exceeds 100-150.

Note: By default, the SMF Bundle Server uses a file-based repository and acts as a development server only. Use the SMF Bundle Server for testing purposes only.

The file-based server configuration provides the same features as the SQL configuration except the following:

- Groups: When you administer software remotely across a set of devices, the file-based server configuration does not enable you to define a collection of users, stations, or bundles.
- Permissions: The file-based server configuration does not enable you to specify which users and stations are permitted to access software bundles.

Safe Bundle Install Protocol

The `Safe Bundle Install` protocol is a protocol between the runtime and the bundle server that you can use to safely download bundles from an authenticated server. When a runtime requests a bundle from the bundle server, the runtime provides the server with the runtime configuration data and a list of the bundles that are currently installed. The bundle server uses the configuration data and the list of bundles to determine which version of the bundle to install. The bundle server compares the packages and services required by the bundle with the packages and services provided by the bundles that are already in the runtime. If the runtime does not contain the required packages and services, the bundle server searches the bundle server database to find the required packages and services and installs the packages and services before installing the bundle.

By enabling the runtime and the bundle server to communicate with each other, the Safe Bundle Install Protocol enables the bundle server to only download a bundle that is capable of executing within with the runtime configuration. As a result, the protocol minimizes the amount of traffic between the server and the runtime.

Before the bundle server downloads a bundle to the runtime, the bundle server determines whether or not the runtime contains sufficient flash storage space to support the size and duration of the download.

The Safe Bundle Install protocol is implemented as a custom URL protocol handler, which handles URLs with the `smfbd:/`*`bundle name`* format (where *`bundle name`* represents the name of the bundle).

SMF Web application development

The SMF platform enables you to access services on the Internet by providing an HTTP Web server. The SMF platform provides two different Web server models. The basic Web Server model is the HttpService, which is an implementation of the OSGi specification for Http Service. This model implements an HTTP 1.0 Web server with a Java Servlet 2.1 engine. In the Http Service model, you can register servlets and resources with the HttpService interface.

The advanced Web Server model is the Web Container, which you can use to run Web applications on the SMF platform. The SMF Web Container supports the Servlet 2.3 and Java Server Page (JSP) 1.2 specifications with minor differences. See comparison to the Servlet 2.3 and JSP 1.2 specifications in the **Reference** section of the SMF Runtime User’s Guide. You can use a Web application to register Servlets, JSPs, and resources with the Web Container. A Web application is packaged as a Web Application Archive (WAR) file. For more information about WAR files, refer to the Servlet 2.3 and JSP 1.2 specifications.

To deploy a WAR to the SMF Web Container, you must convert the WAR file into a Web Application Bundle or WAB. A WAB is an OSGi bundle that you can install, start, stop, update, or uninstall as you would any other OSGi bundle.

You can use the WAB tool, provided by SMF Toolkit, to convert WAR files into WABs. During the conversion, the WAB tool compiles the JSP files in the WAR file into Java Servlets. As a result, you do not need a JSP compiler on the target SMF platform and the size of the web container decreases.

If you only require the OSGi Http Service model and do not need the full SMF Web Container model, install the OSGi Http Service standalone implementation provided by httpservice.jar. If you need the OSGi Http Service model and the SMF Web Container model, install the optional Web Http Service implementation provided by webhttpservice.jar. The Web Http Service implementation requires that the SMF Web Container is installed and started.

Chapter 3. Tasks

This chapter describes the tasks for creating and using bundles.

Using SMF Bundle Developer Tools

This section describes how to use the SMF Bundle Developer Tools.

Using SMF Bundle Developer Tools to develop an example bundle

Use the following steps to develop an example bundle. Make sure you have the SMF toolbar activated in the SMF perspective.

Note: Refer to "Registering and unregistering a service with the OSGi Framework" on page 41 for the source code referred to in the following steps.

1. Start WebSphere Studio Device Developer (WSDD).
2. Create a new project for the example bundle.
   a. Click **File** -> **New** -> **Project**.
      The **New Project** wizard displays.
   b. Select **Extension Services** in the left pane and **Extension Services Bundle Project** in the right pane and click **Next**.
   c. Type `MyTestService` in the **Project Name** field and click **Finish**.
3. Create the interface for the test service with the following steps.
   a. Right-click on the **MyTestService** project in the Java perspective.
   b. Select **New** ->**Interface**.
   c. Type `com.ibm.osg.example.mtservice` in the **Package** field.
   d. Type `MyTestService` in the **Name** field and click **Finish**.
   e. Type the source for `com/ibm/osg/example/mtservice/MyTestService.java`.
   f. Click **Save** to compile the interface.
4. Create the implementation class for the test service with the following steps.
   a. Right-click on the **MyTestService** project in the Java perspective.
   b. Select **New** ->**Class**.
   c. Type `com.ibm.osg.example.mytestservice` in the **Package** field.
   d. Type `MyTestService` in the **Name** field and click **Finish**.
   e. Type the source for `com/ibm/osg/example/mytestservice/MyTestService.java`
   f. Click **Save** to compile the class.
5. Create the BundleActivator for the test service bundle with the following steps.
   a. Right-click on the **MyTestService** project in the Java perspective.
   b. Select **New** -> **Class**.
   c. Type `com.ibm.osg.example.mytestservice` in the **Package** field.
   d. Type `MyBundleActivator` in the **Name** field and click **Finish**.
   e. Type the source for `com/ibm/osg/example/mytestservice/MyBundleActivator.java`

   f. Click **Save** to compile the class.
6. Modify the manifest file for the MyTestService project.

   The Manifest Editor helps you prevent errors when you create a manifest file for a bundle and provides a graphical user interface (GUI) that you can use to create a manifest file.

   a. In the MyTestService project, double-click the META-INF folder and then double-click the `MANIFEST.MF` file. The Manifest Editor GUI displays.

      Note: If the manifest editor does not display, close the `MANIFEST.MF` file and right-click on the `MANIFEST.MF` file in the MyTestService project. Select **Open With**
      -> **Bundle Manifest Editor**.
   b. Type `com.ibm.osg.example.mytestservice.MyBundleActivator` in the **Bundle-Activator** field.

      Or, select the BundleActivator from the pull-down menu in the Bundle-Activator field.
   c. In the **Export Packages** area, click **Add**. Select the **com.ibm.osg.example.mtservice** package and click **OK**.
   d. Close the file, and save when prompted.
7. Create the bundle jar for MyTestService with the following steps.
   a. Right-click on the **MyTestService** project on the **Package Explorer** panel. Select **SMF** ->**Submit Bundle...**.
   b. Select **Submit Jar**. Click **Add Directory** and select the directory to output the bundle jar file. Click **OK**.
   c. Select the directory under **Export Targets** and click **Finish**.

      SMF Bundle Developer Tools place the bundle JAR file in the directory you specified.
   d. Rename the bundle JAR file `MyTestService.jar`.

      Note: You can rename the JAR file with a different name if you choose. However, this example assumes that you rename the file `MyTestService.jar`.
8. To create a new project for the get service bundle, perform the following steps:
   a. Click **File** -> **New** -> **Project**.
   b. Select **Extension Services** in the left pane and **Extension Services Bundle Project** in the right pane and click **Next**.
   c. Type `MyGetService` in the **Project Name** field and click **Next**.
   d. Select the **Eclipse Default JRE Platform Profile** and click **Next**.
   e. Select **Projects**. Select **MyTestService** and click **Finish**.
9. Create the BundleActivator for the get service bundle with the following steps.
   a. Right-click on the **MyGetService** project in the Java perspective.
   b. Select **New** -> **Class**.
   c. Type `com.ibm.osg.example.mygetservice` in the **Package** field.
   d. Type `MyBundleActivator` in the **Name** field and click **Finish**.
   e. Type the source for `com/ibm/osg/example/mygetservice/MyBundleActivator.java`.
   f. Click **Save** to compile the class.
10. Modify the manifest file for the MyGetService project with the following steps.

a. Under the MyGetService project, double-click the MANIFEST.MF file in the META-INF folder. The Manifest Editor graphical user interface (GUI) displays.

   Note: If the Manifest Editor does not display, close the `MANIFEST.MF` file and right-click the `MANIFEST.MF` file in the **MyGetService** project. Select **Open With**->**Bundle Manifest Editor**.

b. Type `com.ibm.osg.example.mygetservice.MyBundleActivator` in the **Bundle Activator** field.

c. Click **Save** to save the manifest file.

11. Create the bundle JAR for MyGetService with the following steps:

    a. Right-click on **MyGetService** in the Package Explorer. Select **SMF**-> **Submit bundle**.

    b. Select **Submit Jar**. Select a directory under **export targets** and click **Finish**.

       The bundle JAR is placed in the directory you specified.

    c. Rename the bundle JAR file `MyGetService.jar`.

       Note: You can rename the JAR file with a different name if you choose. However, this example assumes that you rename the file `MyGetService.jar`.

12. Start the SMF Bundle Server with the following steps:

    a. Select **Run**->**Run...**.

    b. Double-click **SMF Bundle Server**.

    c. Click on the **Run** button to start the SMF Bundle Server.

       Note: From the SMF Bundle Server view of the SMF perspective you can view details about the currently running SMF Bundle Server.

13. Submit the bundles to the server with the following steps:

    a. Right-click on the **MyTestService** project and select **SMF**->**Submit Bundle**.

    b. Check the **Submit jar** checkbox, select the SMF Bundle Server, and click **Finish**.

       SMF Bundle Developer Tools load the MyTestService bundle into the SMF Bundle Server.

    c. Repeat step 13a and 13b for the **MyGetService** project.

       Note: If you view the server from the SMF Bundle Servers tab of the SMF perspective, click on the plus sign (+) next to **Bundles** and verify that your bundles reside in the list.

14. Start the SMF Runtime with the following steps:

    a. Select **Run**->**Run...**.

    b. Double-click **SMF Runtime**.

    c. Click on the **Run** button to start the SMF Runtime server.

       Note: From the SMF Runtime view of the SMF perspective you can view details about the currently running SMF Runtime.

15. Install the bundles from the SMF Bundle Server to the SMF Runtime with the following steps:

    a. Open the SMF Bundle Servers view of the SMF perspective.

    b. Click on the plus sign (+) next to **Bundles** to expand the list.

    c. Right-click on the **MyTestService** bundle and select **Install Bundle**.

    d. Repeat step 15c for the **MyGetService** bundle.

You can view the output of the bundles in the Console view from the SMF perspective.

Working with Extension Services Bundle projects

This section describes how to work with Extension Services Bundle projects.

Creating an Extension Services Bundle project

Complete the following steps to create a new Extension Services Bundle project:

1. Select **File**->**New**->**Project**.

   The new project wizard is displayed.
2. Select **Extension Services** on the left panel and **Extension Services Bundle Project** on the right panel. Then click **Next**.

   The **Create an Extension Services Bundle project** panel is displayed.
3. Specify a project name in the **Project name** field.
4. Click **Next**.

   The second **Create an Extension Services Bundle project** panel is displayed.
5. Select a **Platform Profile** from the pull-down menu.

   A description of the platform profile you selected is displayed in the **Description** field.

   Note: By default, the project creates the Java Build Path from the platform profile and application services you select. This option can be disabled in the last page of this Wizard.
6. Select the boxes next to the **Application Services** you want to use.

   Note: If the platform that you selected in step 5 requires specific services, the services are automatically selected in the **Application Services** field. You can select any additional services that the application uses.
7. Click Next.

   The Java Settings page is displayed. This page allows you to configure Java Development Tools options for the new project. For more information see the "Java Development User Guide".
8. Click **Next**.

   The Extension Services Options page is displayed. The following table details the page's functions.

Table 1. Extension Services Options

Option	**Description**	**Default value**
Create Default BundleActivator	Create a class which implements org.osgi.framework.BundleActivator, and add the corresponding Bundle-Activator entry in the Bundle Manifest.	not selected
Use Platform profile to manage Java Build Path	Add the appropriate libraries to your Java Build Path for the Application Services you have selected. This setting can be changed later in the project's Extension Services property page.	Selected
Automatically manage Import-Package key in Bundle Manifest	Automatically add or remove Import-Package entries from the Bundle Manifest as you use Application Services in your bundle. This setting can be changed later in the project's Extension Services property page.	Selected

Note: In some cases, an Extension Services Bundle Project cannot automatically determine the correct Import-Package settings. For example, if your code uses the `java.lang.Class.forName(...)` api, the tool cannot correctly calculate the required Import-Package settings at compile time. In this case you should disable the **Automatically manage Import-Package** option and configure the Import-Package settings manually using the Manifest Editor.

9. Click **Finish**.

   SMF Bundle Developers Tools creates an Extension Services Bundle project.

Converting a Java project into an Extension Services Bundle project

Complete the following steps to convert a Java project into an Extension Services Bundle project:

1. Select **File**->**New**->**Other...**.

   The **New** project wizard is displayed.
2. Select **Extension Services** on the left panel and **Convert Java to Extension Services Project** on the right panel. Then click **Next**.

   The **Convert Java to Extension Services** panel is displayed.
3. Select the Java project that you want to convert.

   If you want to create a copy of the project before you convert it, select the **Copy before creating** box and provide a name for the new project. The new project will be converted, and the original project will remain unchanged.
4. Click **Next**.

   The second **Convert Java to Extension Services** window is displayed.
5. Select a **Platform Profile** from the pull-down menu.

   A description of the platform profile you selected is displayed in the **Description** field.

   Note: By default, the project creates the Java Build Path from the platform profile and application services you select. This option can be disabled in the last page of this Wizard.
6. Select the boxes next to the **Application Services** you want to use.

   Note: If the platform that you selected in step 5 on page 20 requires specific services, the services are automatically selected in the **Application Services** field. You can select any additional services that the application uses.
7. Click **Next**.

   The Extension Services Options page is displayed. The following table details the page's functions.

Table 2. Extension Services Options

Option	Description	Default value
Create Default BundleActivator	Create a class which implements org.osgi.framework.BundleActivator, and add the corresponding Bundle-Activator entry in the Bundle Manifest.	not selected

Table 2. Extension Services Options (continued)

Option	Description	Default value
Use Platform profile to manage Java Build Path	Add the appropriate libraries to your Java Build Path for the Application Services you have selected. This setting can be changed later in the project's Extension Services property page.	Selected
Automatically manage Import-Package key in Bundle Manifest	Automatically add or remove Import-Package entries from the Bundle Manifest as you use Application Services in your bundle. This setting can be changed later in the project's Extension Services property page.	Selected

Note: In some cases, an Extension Services Bundle Project cannot automatically determine the correct Import-Package settings. For example, if your code uses the `java.lang.Class.forName(...)` api, the tool cannot correctly calculate the required Import-Package settings at compile time. In this case you should disable the **Automatically manage Import-Package** option and configure the Import-Package settings manually using the Manifest Editor.

8. Click **Finish**.

   SMF Bundle Developers Tools converts the Java project into an Extension Services bundle project.

Setting Extension Services project properties

Extension Services Bundle projects have some project-specific properties. To modify the project properties, complete the following steps:

1. Right-click on your project and select **Properties**.
2. Select **Extension Services**.

   The Application Profile tab allows you to choose the Platform Profile that you want to use with this project. You may also select the Application Services that you want to be made available to this bundle.

   The Options tab allows you to control the behavior of the SMF Bundle Developer Tools for this project.

   - Use Platform Profile to manage Java Build Path - if this option is selected, the tools will automatically add the appropriate libraries to your Java Build Path for the Application Services you have selected.
   - Automatically manage Import-Package key in Bundle Manifest - if this option is selected, the tools will automatically add or remove Import-Package entries from the Bundle Manifest as you use Application Services in your bundle.

   Note: In some cases, an Extension Services Bundle Project cannot automatically determine the correct Import-Package settings. For example, if your code uses the `java.lang.Class.forName(...)` api, the tool cannot correctly calculate the required Import-Package settings at compile time. In this case you should disable the **Automatically manage Import-Package** option and configure the Import-Package settings manually using the Manifest Editor.
3. Make the changes you desire and click OK to commit your changes.

Restoring an Extension Services Bundle project environment

The Application Tools manage the Extension Services project environment automatically by:

- Optionally, updating the Java Build Path to reflect the project's platform profile and application services.
- Creating Extension Services metadata in the Manifest file.
- Optionally generating a bundle activator class.

If you modify any of these settings so that they no longer function properly or if you want to restore any of these settings to their original state, complete the following steps:

1. Right-click on your project in the Navigator view and select **Extension Services**->**Restore Environment**.

   The **Restore Extension Services Environment** window is displayed.
2. Refer to the following descriptions to decide which environment settings you want to restore. Then select the box next to each setting that you want to restore.

   Restore Java Build Path
   Resets the Java Build Path based on the project's platform profile and application services.

   Restore Bundle Activator
   Creates a default bundle activator for the project and overwrites any changes you made to the default bundle activator.

   Restore Bundle Manifest
   Resets the Extension Services metadata for the Manifest file.
3. Click **OK** to restore the settings you selected.

Creating bundle folders

Bundles are `JAR` or `JXE` files that you create with Service Management Framework (SMF) Bundle Developer. Bundles typically contain Java `CLASS` files and resources.

Note: The preferred and recommended way to develop bundles using SMF Bundle Developer Tools is to create an Extension Services Bundle Project.

Complete the following steps to create a folder where your bundle will reside:

1. From the main workbench window, do one of the following:
   - Click the **Create a Bundle Folder** wizard button on the workbench toolbar.
   - Or click the **Open The New Wizard** button on the toolbar and select **SMF** on the left panel and then select **Bundle Folder** on the right panel.
   - Or from the drop-down menu of **Open The New Wizard**, select **Bundle Folder**.
2. In the **New Bundle Folder** wizard, select a Java project source folder from the list of available bundle containers.

   Note: If the list does not contain any projects, create a new Java project in the workbench and then resume with the remaining steps.
3. Select the boxes as necessary to create the `IVEATTRS.XML` and `IVERES.XML` files.
4. Click **Finish**.

   After you complete the fields in the **New Bundle Folder** wizard, your bundle project folder contains a folder called `META-INF`, which contains the following:
   - `.bndldesc`
   - `IVEATTRS.XML`
   - `IVERES.XML`

- `MANIFEST.MF`

Note: If the `.bndldesc` does not display, verify that the Hide.* files filter is not selected. Click on the arrow at the top of the Package Explorer view and select **Filters**, then clear the **Hide .*** checkbox.

5. Double-click the `IVEATTRS.XML` file in the Package Explorer view and edit the bundle attribute information in the Bundle Attributes editor.

   Refer to "Editing attribute information" on page 25 for more information.
6. Double-click the `IVERES.XML` file in the Package Explorer view and edit the bundle resource information in the Bundle Resources editor.

   Refer to "Editing resource information" on page 25 for more information.
7. Double-click the `MANIFEST.MF` file in the Package Explorer view and edit the bundle manifest information in the Bundle Manifest editor.

   Refer to "Editing manifest information" on page 25 for more information.
8. Double-click the .bndldesc file in the Package Explorer view and specify the bundle's content in the Bundle Description editor.

   Refer to "Editing bundle descriptions" on page 26 for more information.

Creating precompiled bundles without source

You can submit bundles that do not contain any Java source files to the file system and to the bundle server. As such, you can submit third-party bundles as JARS or JXES.

To submit a bundle that does not contain any Java source files, complete the following steps:

1. Set up the bundle JAR file.
2. Set up the bundle that will use the bundle JAR file.

Setting up the bundle JAR file: Refer to the following steps to set up a bundle JAR file:

1. Click **File** -> **New** -> **Project**.

   The **New Project** wizard displays.
2. Select **Java** in the left pane and **Java Project** in the right pane and click **Next**.
3. Type `MyTestService` in the **Project Name** field and click **Next**.
4. Click the **Source** tab. Select **Use source folders contained in the project**.
5. Click **Create New Project Folder** and name the new folder `classfiles`.
6. Set the **Build output folder** to *`project`*`/bin`, where *`project`* represents the path of the project folder.
7. Click on the **Libraries** tab and click **Add Variable**.
8. Select **SMF_CLIENT** from the list and click **Extend...**.
9. Select **osgi.jar** and click **OK**.
10. Right-click on the `classfiles` source folder and import the bundle JAR file.

Setting up the bundle file that will use the bundle JAR file: Complete the following steps to update the Java build path to include the class files:

1. Right-click on the project and select **Properties**->**Java Build Path**.
2. On the **Projects** tab, select the project that contains the bundle JAR class files.
3. On the **Libraries** tab, select **Add class folder** and click **OK**.
4. Select the **classfile** folder in the bundle JAR project that you created in step 10 of "Setting up the bundle JAR file."

Note: If you try to edit class-file bundles, the Manifest Editor Export Package function enables you to export leaf packages only if none of the parent packages contain Java files. If you create a new bundle from a JAR file, you might have to edit the manifest file to export parent (non-leaf) packages.

Editing bundle information

You can view, edit, and update the bundle information using the SMF Bundle Developer Tools.

Editing attribute information

To update any of the attribute information for a bundle, open the Bundle Attributes Editor by doing one of the following in the Package Explorer view:

- Double-click the `IVEATTRS.XML` file.
- Right-click on the `IVEATTRS.XML` file and select **Open** from the context menu.
- Right-click on the `IVEATTRS.XML` file and select **Open With**->**Bundle Attributes Editor** from the context menu.

In the Bundle Attributes Editor, you can type information or select a value from the drop-down list.

Editing resource information

To update any of the resource information for a bundle, open the Bundle Resources Definition Editor by doing one of the following in the Packages Explorer view:

- Double-click the `IVERES.XML` file.
- Right-click on the `IVERES.XML` file and select **Open** from the context menu.
- Right-click on the `IVERES.XML` file and select **Open With**-> **Bundle Resources Definition Editor** from the context menu.

After the Bundle Resources Definition editor opens, you can update the resource information by typing a new value in the resource field. To add a new resource and its value, click the **Add** button in the **User-defined Resource Information** section. To remove a resource, select the box next to it and click the **Remove** button in the **User-defined Resource Information** section. You can remove only resources that you added to the User Defined Resource Definitions.

Editing manifest information

To update any of the resource information for a bundle, open the Bundle Manifest Editor by doing one of the following in the Packages Explorer view:

To update any of the manifest information for a bundle, open the Manifest Editor by doing one of the following in the Packages Explorer view:

- Double-click the `MANIFEST.MF` file.
- Right-click on the `MANIFEST.MF` file and select **Open** from the context menu.
- Right-click on the `MANIFEST.MF` file and select **Open With**->**Bundle Manifest Editor** from the context menu.

After the Bundle Manifest Editor opens, you can update the manifest information by typing in the value field of the key you want to update.

Adding and removing user-defined manifest items: To create a new manifest item, navigate to the **User-defined Manifest Items** section of the Manifest Editor and click **Add**.

To remove a user-defined manifest item, navigate to the **User-defined Manifest Items** section of the Manifest Editor, select the item or items you want to remove, and click **Remove**.

Adding packages and services: To add any of the import or export options to the `MANIFEST.MF` file:
1. Navigate to the import or export package or service section.
2. Click **Add** next to the package or service that you want to import or export.
3. In the dialog, select the packages or services that you want to import or export.
4. Click **OK**.

Note: In some cases, an Extension Services Bundle Project cannot automatically determine the correct Import-Package settings. For example, if your code uses the `java.lang.Class.forName(...)` api, the tool cannot correctly calculate the required Import-Package settings at compile time. In this case you should disable the **Automatically manage Import-Package** option and configure the Import-Package settings manually using the Manifest Editor.

Adding bundle native code information: You can use the **Native Code Clauses** section of the Manifest Editor to describe the files that contain the implementation of the native methods used by classes in the bundles.

Complete the following steps to add bundle native code information:
1. Navigate to the **Native Code Clauses** section of the Manifest Editor.
2. Click **Add**.
3. Complete the environment parameters information on the left. You can type comma-separated values into the **Environment** field or use the **Prompt...** button to open a dialog that enables you to select from a set of known values. On the right, use the **Edit** button to manage the list of files for that environment.
4. Click **Add** to open a dialog from which you can select files or click **Remove** to remove any selected files from the list.
5. Click **OK**.

Editing bundle descriptions

Bundle description information resides in the `.bndldesc` file in each bundle folder.

To update any of the description information for a bundle, open the editor by doing one of the following in the Package Explorer view:
- Double-click the `.bndldesc` file.
- Right-click on the `.bndldesc` file and select **Open** from the context menu.
- Right-click on the `.bndldesc` file and select **Open with**->**Bundle Description Editor** from the context menu.

Refer to “Bundle description editor” on page 57 for more information.

The Bundle Description Editor displays a list of inclusions and exclusions that represent projects, source folders, packages, folders, or individual files. Top level entries represent inclusions and sub-entries represent exclusions.
- You can add inclusions by clicking **Add Inclusion...** and selecting the a resource.
  – If the `.bndldesc` file lists a non-package folder, the bundle includes all of the non-package folder’s sub-contents.

- If `.bndldesc` file lists a package folder, the bundle includes only the package and the files directly contained in the package.
- You can choose to recursively select all of a folder's sub-folders by checking the **Recursively include subdirectories/sub-packages** box. The Bundle Description Editor lists inclusions and exclusions based on the information previously described.

- You can add exclusions by selecting an Inclusion and clicking **Add Exclusion...** and selecting the resources you want to exclude.
- To remove an included or excluded resource from the list, select the resource in the list and click **Remove**.

The following rules apply when the Bundle Description Editor decides which resources to include or exclude:

- New inclusions cannot overlap another rule including exclusions.
- To add an exclusion, you must select an inclusion. The dialog lists the contents of the inclusion from which you can select what to exclude.
- You cannot exclude the META-INF folder for a bundle.

If a bundle contains only the contents of the bundle folder, you do not need to edit the bundle description file because a bundle's content defaults to the contents of the bundle folder automatically.

Viewing problems

To view problems that exist within the bundle, open the Tasks view. From this view, you can view a specific problem in the default editor for the file type, by double-clicking on the description of a problem or by right-clicking on the problem and selecting **Go to** from the context menu.

Note: The Packages Explorer view indicates errors and warnings associated with the file.

Creating binary bundle projects

This function allows you to transform a jar file with no source code into a bundle. It is assumed that the jar file you import into the Binary Bundle Project does not contain any source code.

The Binary Bundle Project helps you insert the proper markup in the MANFEST.MF, IVEATTRS.XML, and IVERES.XML files. A Binary Bundle Project can be submitted to a Bundle Server or the file system in the same way as a normal Bundle Folder.

To create a Binary Bundle Project, perform the following steps:

1. Start the wizard by selecting **File -> New -> Other...**. Then select **SMF** in the left panel, and **Binary Bundle Project** in the right.
2. Enter the name of your project in the **Project name** field. Click **Next**.
3. In the **Jar to import** field, specify the jar that contains the bundle(s) you want to import. Keep the default check boxes checked to create IVEATTRS and IVERES. Click **Finish**. The wizard creates the binary bundle project and stores your bundles in it.
4. Right click the binary bundle project name. Select **Properties**.
5. Select **Java Build Pat**, then the **Libraries** tab. Click **Add External JARs...**. Browse the install directory path, then find and select `servlet.jar`. Click **OK**. The result is shown here. Click **OK**.

6. Refresh the view. The system will warn you if your bundle name is not unique in the workspace.

Deleting a bundle

Use the following steps to delete a bundle:

1. Remove the source folder from the project's classpath.
2. Delete the `META-INF` folder in the source folder.
3. Delete the `MANIFEST.MF` file in the `META-INF` folder.
4. Remove the `Bundle-Name` entry from the `MANIFEST.MF` file.

To delete any file or folder in the Bundle Folders or SMF Bundle Servers view, select the bundle and press the **Delete** key.

Note: When you delete a source folder you remove the bundle and all of the code and resources stored in that folder.

Managing an SMF Runtime

You can use the SMF Runtime view to manage an SMF runtime.

Starting the local Service Management Framework Runtime

Complete the following instructions to start the local Service Management Framework (SMF) Runtime:

1. In the SMF perspective, click the drop-down menu on the **Run** icon and select **Run...**. The Launch Configurations dialog displays.
2. Select **SMF Runtime**. The Launch Configuration dialog is populated with the default values for the SMF Runtime.
3. Select **New**.
4. Select **Run.**
5. Click **OK**.

Starting Service Management Framework with a SecurityManager

Complete the following steps to configure Service Management Framework (SMF) to start with a SecurityManager from the SMF Runtime Launch Configurations dialog:

1. Click the **Arguments** tab.
2. In the **VM Arguments** box, append the following information:

   Notes:

   a. The following example displays on more than one line because of the width of the page.
   b. This example uses the default TECHHOME location for WebSphere Device Developer.
   c. *`dir_name`* represents the directory where you installed WebSphere Studio Device Developer.

```
-Djava.security.manager
-Djava.security.policy=C:\dir_name\wsdd5.0\technologies\smf\client\smfide.policy
```

Debugging a local Service Management Framework Runtime

Complete the following steps to debug the local Service Management Framework (SMF) Runtime:

1. Click the drop-down arrow on the **Debug** toolbar button and select **debug**.

The debug dialog displays.

2. If necessary, set the SMF Bundle Server, User, and Station options by selecting the **SMF** tab, and then the **Bundle Server** tab.

Displaying runtime properties

To display all of the properties of a running runtime, click on the drop-down menu in the SMF Runtime view and select **Show Runtime Properties** from the context menu.

Connecting to a remote Service Management Framework Runtime

Complete the following steps to connect to a remote Service Management Framework (SMF) Runtime:

1. Click the drop-down arrow on the **Run** toolbar button and select **Run...**.
2. Select **Remote SMF Runtime** from the configurations list and click **New**.
3. Type or select the **Host** and **Port** where the agent is running.
4. Click **OK**.

Bundle garbage collection

To perform a local garbage collection and uninstall the bundles that you do not use or need as dependencies, click the drop-down arrow on the SMF Runtime view button and select **Bundle Garbage Collect**.

SMF bases the dependency criteria for garbage collection on package and service dependencies, as specified in the `smf.properties` file. Any bundles that you installed explicitly with the `file:/` protocol or as the specified bundle with the `smfbd:/` protocol are members of the root set and are not candidates for garbage collection. SMF only considers bundles that were installed as prerequisite bundles with the `smfbd:/` protocol for garbage collection.

Terminating the Service Management Framework Runtime

To terminate the Service Management Framework (SMF) Runtime, select the **Terminate Runtime** command from the drop-down menu in the SMF Runtime view.

Note: This command disconnects you from the runtime and terminates the runtime process.

Launching the framework

To launch the OSGi™ framework from the SMF Runtime view, select the **Launch Framework** command from that view's drop-down menu.

Shutting down the framework

To shutdown the OSGi™ framework from the SMF Runtime view, select the **Shutdown Framework** command from that view's drop-down menu.

Submitting bundles

To submit a bundle to the bundle server, a user must have "Developer" privileges, which are the default selections. Refer to "Changing user properties" on page 34 for more information.

1. Right-click on the bundle in the Packages Explorer view and select **Submit Bundle** from the context menu.

   Notes:

   a. You can select multiple bundle folders and projects. Select a project to selects all of the bundle folders in that project.
   b. You can also access **Submit Bundle** from the toolbar in the Packages Explorer view.

   The **Target Submission** wizard opens and displays a list of the items to export.
2. Select **Submit Jar**, or **Submit Jxe**, or both as the type of output format.

   Note: If you select **Submit Jxe**, you must also select the JXE output type to generate. You can generate **big endian** or **little endian** and **32-bit** or **64-bit** output types.

   Specify other SmartLinker options if you choose.
3. From the **Export Targets** list, select the targets where you want to submit the bundle. The targets are a combination of SMF Bundle Servers that you added in the SMF Bundle Servers view and File System targets that you added from the SMF preferences page.

   Note: Use a File System target when you want to keep a copy of the JAR or JXE file and other generated files. Use an SMF Bundle Server target when you want the bundle placed on an SMF Bundle Server.
4. Select **Replace bundles** if you want to overwrite your current bundle with an updated version.
5. If you want to create an ant script, select **Generate Ant Script** and specify the name and location of the Ant script. Ant scripts enable you to submit bundles using the same specifications in the wizard by running the script.
6. Click **Finish**.

Uninstalling a bundle

Use the following steps to uninstall a bundle:

1. Right click the bundle in the SMF Runtime view.
2. Select **Uninstall**.

Using snapshots

You can use snapshots to store the current state of the runtime for later use. You might choose to use snapshots to load all of the bundles needed on a particular target runtime and to save the snapshot so that you can test different configurations and still return to the previous state.

Note: You can only store and restore bundles that you installed with the `smfbd:/` protocol in a snapshot. You cannot save bundles that you installed with other protocols. When you restore a snapshot, all of the currently installed bundles are uninstalled and as a result, all of the bundles that were not installed with SMF Bundle Developer Tools are lost.

Storing snapshots

Complete the following steps to store a snapshot:

1. In the SMF perspective, click the drop-down menu on the **Run** icon and select **Run**.

   The Launch Configurations dialog displays.

2. Select **SMF Runtime**->**New** and click **Run**.
3. From the drop-down menu in the SMF Runtime view, select **Store Snapshot...**.
4. In the **Store Snapshot** wizard, type a name and description for the snapshot that you would like to save.
5. Click **Finish**.

   Service Management Framework stores the state of the runtime on the server where the runtime is connected. The snapshot saves all of the bundles loaded on the runtime and the hardware properties of the runtime.
6. In the SMF Bundle Servers view, expand the Snapshots element on the localhost server and double-click the snapshot you saved. The details are displayed in the Snapshot Details view.

After you store the snapshot, the snapshot displays in the SMF Bundle Servers view.

Restoring snapshots

Complete the following steps to restore a snapshot:

1. If the Runtime is not currently running, then click the drop-down menu on the **Run** icon and select **Run** in the SMF perspective.

   The Launch Configurations dialog displays.
2. Select **SMF Runtime** -> **New** and click **Run**.
3. Verify that you are connected to the server that includes the snapshot you want to restore.
4. From the drop-down menu in the SMF Runtime view, select **Restore Snapshot**. You can access the drop down menu using the down arrow in the SMF Runtime dialog toolbar.
5. Type the name of the snapshot you want to restore, or select it from the drop-down list.

   Note: If you prefer, you can navigate through the SMF Bundle Servers view and right-click on the snapshot and select **Install** from the context menu.
6. Click **Finish**.

   The runtime retrieves the snapshot. When the snapshot loads, all of the bundles that are currently installed are uninstalled and replaced with the bundles contained in the snapshot.

   Note: A snapshot is only downloaded and restored if the hardware properties on the snapshot exactly match those on the runtime.

Viewing snapshot contents

Complete the following steps to view snapshot contents:

1. Right-click on the snapshot in the Bundle Servers view and select **Show Versions** from the context menu.

   The Snapshot Details view displays.
2. In the Snapshot Details view, right-click on a version of the snapshot and select **Show Contents** from the context menu.

Viewing snapshot version information

Complete the following steps to view snapshot version information:

1. In the SMF Bundle Servers view, expand the Snapshots item under a server.
2. Right-click on the item and select **Show Versions** from the context menu.

Details display in the bottom view, which is labeled **Snapshot Details**.

Refreshing snapshot lists

When you add or modify snapshots, the additions or modifications you made might not display immediately in the lists of the SMF Bundle Servers view. To refresh a snapshot list, click the **Refresh** toolbar button in the SMF Bundle Servers view.

Removing snapshots

Complete the following steps to remove a snapshot:

1. View the snapshot's details by right-clicking on the snapshot in the SMF Bundle Servers view.

   The Snapshot Details view displays.
2. Right-click on the snapshot in the details view and select **Remove** or **Remove All** from the context menu.

   The snapshot is removed.

Using the SMF Bundle Developer Server Tools

This section describes how to use the SMF Bundle Developer Server Tools.

Adding a connection to a bundle server

Complete the following steps to add a connection to a bundle server:

1. In the SMF Bundle Servers view, right-click the bundle server, and select **New** -> **Bundle Server** from the context menu.

   Note: You can also click the **Add connection to Bundle server** button in the workbench toolbar in the SMF perspective.
2. In the **SMF Bundle Server** wizard, type the **Host** name, **Port** number, **Username**, and **Password** for the server.

   Leave the URL prefix as the default value (`/smf`).

   Note: The host and port parameters of a new bundle server cannot be the same as the parameters of an already configured bundle server.
3. Click **Finish**.

   The new server is displayed in the SMF Bundle Servers view.

Adding a user

Complete the following steps to add a user:

1. In the SMF Bundle Servers view, right-click the bundle server and select **New** -> **User** from the context menu.

   Note: You can also click the **New user** button in the workbench toolbar.
2. In the **SMF User** wizard, type a **Username** and **Password** for the new user.
3. Type the **First Name** and **Last Name** for the new user.
4. Select one or more of the following types of privileges for the new user:**Administrator**, **Developer**, or **Runtime**.
5. In the **Servers** list box, select the server or servers that you want the new user to see.
6. Click **Finish**.

   The new user is displayed in the SMF Bundle Servers view.

Adding a station

Stations are unique IDs that the server uses for each device that connects to the server.

Complete the following steps to add a station:

1. In the SMF Bundle Servers view, right-click the bundle server and select **New** -> **Station** from the context menu.

   Note: You can also click the **New station** button in the workbench toolbar.
2. In the **SMF Station** wizard, type a station ID.

   The default station ID is `78`. You can specify any unique string of letters and numbers. You might consider using machine MAC address as a schema for station IDs.
3. In the **Servers** list box, select a server.
4. Click **Finish**.

   The new station displays in the SMF Bundle Servers view.

Starting an SMF Bundle Server

You can launch an SMF Bundle Server from the command line or from the **SMF Bundle Developer Launch Configurations** dialog. For information on launching an SMF Bundle Server from the command line, refer to the Service Management Framework Server User's Guide.

Use the following instructions to launch the SMF Bundle Server from WebSphere Device Developer:

1. On the main menu, click **Run**->**Run**.

   The Launch Configurations dialog opens, and a list of launch configurations display on the left.
2. From the list of launch configurations, select **SMF Bundle Server** and click **New**.

   SMF Bundle Developer Tools add a new configuration under SMF Bundle Server and displays the configuration options tabs.
3. By default, the configuration name is `SMF Bundle Server`. You can change this by typing a new name in the **Name** field.
4. Complete the information on the Launch Configuration dialog by referring to the following table:

Table 3. Launch Configurations tabs and descriptions

Tab	Options	Description
Options	Bundle Server Port	Specifies the port on which the bundle server listens. The default value is `8080`.
	Bundle Server Name	Specifies the Bundle Server Web application name. The default value is `smf`.
	User ID	Specifies the Bundle Server user ID to use. The default value is Admin.
	User Password	Specifies the Bundle Server user password to use. The default is `Admin`.
Arguments		Enables you to specify arguments to Java.
JRE		Enables you to specify Java Runtime Environment you want to run.

Table 3. Launch Configurations tabs and descriptions (continued)

Tab	Options	Description
Classpath (2 sub-tabs)	User classes	Specifies the seven required SMF JAR files by default.
	Bootstrap	Enables you to specify the bootstrap based on the virtual machine.
Common		Enables you to set the perspective that SMF Bundle Developer Tools launch when you start SMF Bundle Developer Tools. SMF Bundle Developer Tools launch into the SMF perspective by default.

Note: In most cases, you can accept all of the default values on all of the tabs.

5. Click **Run**.

Viewing bundle version information

Complete the following steps to view bundle version information:

1. In the SMF Bundle Servers view, expand the Bundles item under a server.
2. Right-click on the item and select **Show Versions** from the context menu.

   Details display in the bottom view, which is labeled **Bundle Details**.

Changing user properties

Complete the following steps to change user properties:

1. In the SMF Bundle Servers view, expand the Users item under a server.
2. Right-click on the item and select **Properties** from the context menu.
3. Complete any modifications to the user properties.
4. Click **Apply**.
5. Click **OK** to save the changes.

Viewing station properties

Complete the followings steps to view station properties:

1. In the SMF Bundle Servers view, expand the Stations item under a server.
2. Right-click on the item and select **Properties** from the context menu.

   The station's properties display.

Removing users or stations

To remove a user or station, right-click on the user or station in the SMF Bundle Servers view and select **Remove** from the context menu.

Removing bundles

SMF administrators can delete bundles. An administrator might choose to delete a bundle if the bundle does not adhere to a set of conventions or standards that might be necessary for a specific bundle server environment. When you delete a bundle, Service Management Framework removes the bundle completely from the bundle server repository.

Complete the following steps to remove a bundle:

1. View the bundle's details by right-clicking on the bundle in the SMF Bundle Servers view. You can remove all instances of a bundle here by selecting **Remove All**.

The Bundle Details view displays.

2. Right-click on the bundle in the details view and select **Remove** or **Remove All** from the context menu. In the Details view, you can only remove one instance of a bundle.

   The bundle is removed.

Enabling or disabling a bundle

Administrators must enable all bundles submitted by developers before the bundle is available to download.

Note: All bundles are enabled by default.

- If the **Enable** action is selected, Service Management Framework changes the status of the bundle to enabled, and the bundle is visible and downloadable.
- If the **Disable** action is selected, Service Management Framework changes the status of the bundle to disabled. The bundle is no longer visible or downloadable; however, the bundle remains in the bundle server repository.

Complete the following steps to enable or disable a bundle:

1. Right-click on the bundle and select **Show Versions** from the context menu.

   The Bundle Details view displays.
2. Right-click on the bundle in the Bundle Details view and select **Enable** or **Disable** from the context menu.

Removing a bundle server

To delete a bundle server, right-click on the bundle server in the SMF Bundle Servers view and select the **Remove** command from the context menu.

Refreshing bundles, stations, or users

When you add or modify bundles, stations, or users, the additions or modifications you made might not display immediately in the lists of the SMF Bundle Servers view. To refresh a bundle, station, or user, click the **Refresh** toolbar button in the SMF Bundle Servers view.

Viewing bundle version properties

To view the properties of a bundle version, right-click on the bundle version in the Bundle Details view and select **Open Bundle** from the context menu.

Managing bundles on the SMF Runtime

This section describes how to use the SMF Runtime view to manage bundles on a runtime.

Installing a bundle

Complete the following steps to install a bundle:

1. In the SMF Runtime view, select **Show Runtime Info** from the shortcut menu.
2. At the bottom of the SMF Runtime view, type the name of the bundle that you want installed in the text field and click **Install**.

Note: You can also install the bundle from the SMF Bundle Servers view by right-clicking on the bundle and selecting **Install** from the context menu. SMF only installs the item you selected if the runtime points to the server that contains the bundle.

Starting a bundle

To start a bundle, right-click on the bundle in the SMF Runtime view and select **Start** from the context menu.

Note: If you cannot see your bundle in the SMF Runtime view, verify that the **Show Bundles** option is selected.

Stopping a bundle

To stop a bundle, right-click on the bundle in the SMF Runtime view and select **Stop** from the context menu.

Note: If you cannot see your bundle in the SMF Runtime view, verify that the **Show Bundles** option is selected.

Updating a bundle that is running on the runtime

To update a bundle running on the runtime, right-click on the bundle in the SMF Runtime view and select **Update** from the context menu.

Note: If you cannot see your bundle in the SMF Runtime view, verify that the **Show Bundles** option is selected.

Uninstalling a bundle

To uninstall a bundle, right-click on the bundle in the SMF Runtime view and select **Uninstall** from the context menu.

Note: If you cannot see your bundle in the SMF Runtime view, verify that the **Show Bundles** option is selected. You cannot uninstall the system bundle.

Uninstalling a bundle with garbage collection

The bundle garbage collection deletes only the bundles affected by the uninstalled bundle and uninstalls the bundles that are not used or needed as dependents. The bundle garbage collection bases the dependency criteria on package and service dependencies, as specified in `smf.properties` file. Bundles that you installed explicitly with the `file:/` protocol or as the specified bundle with the `smfbd:/` protocol) are members of the root set and are not candidates for garbage collection. The garbage collection only considers bundles that were installed as prerequisite bundles with the `smfbd:/` protocol.

To uninstall a bundle and perform a local bundle garbage collection on its dependent bundles, right-click on the bundle in the SMF Runtime view and select **Uninstall GC** from the context menu. You must have the dependency admin bundle installed.

Note: If you cannot see your bundle in the SMF Runtime view, verify that the **Show Bundles** option is selected.

Viewing bundle properties

To view bundle properties, double-click on the bundle in the SMF Runtime view or right-click on the bundle in the SMF Runtime view and select **Open Bundle** from the context menu.

Note: The **Open Bundle** context menu option opens the bundle editor in read-only mode.

Viewing service properties

To view service properties, double-click on the service in the SMF Runtime view or right-click on the bundle in the SMF Runtime view and select **Properties** from the context menu. You can only see properties for one service at a time.

Chapter 4. Reference

This chapter provides reference materials you will need for working with the SMF Bundle Developer Tools.

Working with OSGi bundles

OSGi™ bundles consist of a JAR file that contains Java classes, resources, and a manifest file. Bundles can register services for other bundles to use, use services registered by other bundles, export Java packages for other bundles to use, and import Java packages from other bundles.

OSGi core bundles

The runtime framework includes bundles that provide basic services that other bundles can use. Some of the bundles are implementations of services defined by the OSGi™ specifications. When you launch the SMF Runtime, the runtime installs a subset of the core bundles. You can install all of the core bundles to the SMF Runtime from the SMF Bundle Server.

The `OSGi Service Platform Release 3 Service Interfaces` bundle provides the interface specifications of the OSGi standard services, such as the `osgi-services.jar` or `osgi-services.jxe` file. This SMF release includes bundles that provide implementations of the OSGi standard services. Refer to the User's Guide for Service Management Framework Runtime for more information about the core bundles.

Creating OSGi bundles

This section describes how to create an OSGi bundle. For more detailed information about writing bundles, refer to the OSGi Service Platform Release 3 specification.

Note: The SMF Sample Bundles project includes a sample bundle, called the Pizza servlet bundle, in the `TECHHOME\smf\client\samples\source` directory. The Pizza servlet bundle is a simple bundle that registers a servlet with the HTTP Service.

Refer to the following sections for information about writing bundles with SMF.

Conventions for creating bundles

When you create SMF bundles, use the following conventions:

- Clean up objects and threads properly during your stop method. SMF does not terminate lingering threads.
- Do not permit the start and stop methods for your bundle to use a lot of time. Because services start and stop one at a time, each service must wait to start or stop until the previous service finishes starting or stopping.
- Do not assume that a service is always present. A service is only present when the bundle that registered the service is available.

Creating manifest files

Each bundle must contain a manifest file. The bundle's manifest file contains data that the framework needs to correctly install and activate the bundle. When you specify data in a manifest file, you must use the headers that were defined by the

OSGi™ specification. You can use user-defined headers; however, the framework ignores any headers that it does not understand. In addition, you must place the OSGi headers at the beginning of the manifest file. Refer to the OSGi Service Platform Release 3 specification for more information about the OSGi Manifest file format and syntax.

When you use the Service Management Framework Bundle Developer Tools to create the manifest file for your bundle, the Bundle Developers Tools name your manifest file `Manifest.mf` and place it in the `META-INF` directory inside your bundle.

Use the following list of headers in your Manifest file.

- Import-Package

  Use this header to specify the names of any package that you want your bundle to import from other bundles. If you do not specify the packages your bundle needs in this header, you will get a `noClassDefFound` exception when the bundle loads.

  Note: You must also specify the package you want to import in the Export-Package header of the bundle that provides the package.
- Export-Package

  Use this header to specify the name of any package that you want your bundle to export to other bundles. If you do not specify the packages needed by other bundles in this header, the dependent bundles will not resolve.
- Bundle-Activator

  Use this header to specify the fully-qualified name of the BundleActivator class.

  A bundle designates a special class to act as a Bundle Activator. The Framework must instantiate this class and invoke the start and stop methods to start or stop the bundle as needed. The bundle's implementation of the BundleActivator Interface enables the bundle to initialize a task, such as registering services, when the bundle starts and to perform clean-up operations when the bundle stops.

  Note: This is not a mandatory header. If the purpose of your bundle is only to export packages for use by other bundles, then you do not need to specify a BundleActivator.

  The **New Bundle Folder** wizard automatically creates an OSGi Manifest file for you. Service Management Framework Bundle Developer provides a Bundle Manifest Editor that you can use to edit the contents of the Manifest file.

Refer to the OSGi Service Platform Release 3 specification, which Service Management Framework Bundle Developer provides, for descriptions of other bundle headers, such as the following, which provide bundle description information:

- Bundle-Name
- Bundle-Description
- Bundle-Copyright
- Bundle-Vendor
- Bundle-Version
- Bundle-DocUrl
- Bundle-ContactAddress

Understanding services

In the OSGi™ environment, bundles are built around a set of cooperating services that are available from a shared service registry. The service interface defines the OSGi service, which is implemented as a service object.

Registering and unregistering a service with the OSGi Framework

The framework passes a BundleContext object to your bundle when it invokes your BundleActivator's start method. Your bundle can use the BundleContext object to interact with the framework by calling the methods of the BundleContext object. One method that your bundle can call is `registerService`, which uses a service object and an interface name to register a service with the framework's service registry.

Our service registers with the following interface. In an ideal environment, one package contains all of the classes and interfaces that you want to export to the OSGi framework. In this example, `com.ibm.osg.example.mtservice` package is exported. In the following example, a service called `com.ibm.osg.example.mtservice.MyTestService` registers with the framework. This service prints a debugging message to the standard output.

```
/******* START OF com/ibm/osg/example/mtservice/MyTestService.java *******/
package com.ibm.osg.example.mtservice;
public interface MyTestService {
     // One method is provided by the service.  This method will simply print
     // the message to standard out.
     public void printMessage(String message);
}
/******* END OF com/ibm/osg/example/mtservice/MyTestService.java *******/
```

The following class provides the implementation for our service. The class is not a part of the `com.ibm.osg.example.mtservice` exported package. Do not export packages that contain classes that provide implementation to services.

```
/****** START OF com/ibm/osg/example/mytestservice/MyTestService.java ******/
package com.ibm.osg.example.mytestservice;
public class MyTestService implements com.ibm.osg.example.mtservice.MyTestService{
     public void printMessage(String message){
          System.out.println("MyTestService - " + message);
     }
}
/****** END OF com/ibm/osg/example/mytestservice/MyTestService.java ******/
```

The following BundleActivator class registers the `com.ibm.osg.example.mtservice.MyTestService` service with the framework.

```
/****** START OF com/ibm/osg/example/mytestservice/MyBundleActivator.java ******/
package com.ibm.osg.example.mytestservice;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;

public class MyBundleActivator implements BundleActivator {

    ServiceRegistration registration;

/*Create a new instance of the TestService and then use the BundleContext object to register it.*\
/*Store the registration object to use to unregister the service when the bundle is
stopped by the framework.*\
 public void start(BundleContext context) {
   MyTestService testservice = new MyTestService();
   if( registration == null ){
               registration =
    context.registerService("com.ibm.osg.example.mtservice.MyTestService",testservice,null);
           }
 }
     public void stop(BundleContext context) {
  if ( registration != null ){
              registration.unregister();
          }
          registration=null;
     }
}
/****** END OF com/ibm/osg/example/mytestservice/MyBundleActivator.java ******/
```

Getting and un-getting services from the OSGi Framework

Bundles register and unregister services. Bundles that depend on services must account for the possibility that the requested service might not be available. The service can register or unregister with the framework at any time. You can use a ServiceTracker to enable your bundles to query or listen for service registrations and to react accordingly.

```
/****** START OF com/ibm/osg/example/mygetservice/MyBundleActivator.java ******/
package com.ibm.osg.example.mygetservice;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.util.tracker.ServiceTracker;
import com.ibm.osg.example.mtservice.MyTestService;

public class MyBundleActivator implements BundleActivator, Runnable{
 private boolean done=false;
 private ServiceTracker testServiceTracker;

     // Bundle Activator Start Method
     public void start(BundleContext context){
/* Here we initialize and open our ServiceTracker. It will track any service registering under
the "com.ibm.osg.example.mtservice.MyTestService" interface. *\
  testServiceTracker = new ServiceTracker(context,"com.ibm.osg.example.mtservice.MyTestService",null);
          testServiceTracker.open();
// Here we start a thread that will continue to use our service until
// the bundle is stopped.
  Thread t=new Thread(this);
          t.setName("mygetservice thread");
          t.start();

     }

    /*Bundle Activator Stop Method -- here we stop the thread and close the
ServiceTracker*\

  public void stop(BundleContext context){
           done=true;
           testServiceTracker.close();
      }
    //Here is a method that uses the service we are tracking.  First we get
    //the service from the tracker, then we call its printMessage method.
      public void useService(String message){
           MyTestService testService = (MyTestService)testServiceTracker.getService();
           if( testService != null ){
                // If the service is available then use it.
                testService.printMessage(message);
   }
           else{
                // If the service is not available then perform an acceptable action.
                // Here we just print the message to standard out and indicate the service
                // was not available.
                System.out.println("No MyTestService available - " + message);
            }
        }

    // Simply continues to use the test service every second until the done flag is set.
  public void run(){
            int i = 0;
   done = false;
   while (!done){
    useService("message from test " + i++);
                try{
                     Thread.sleep(1000);
                }
                 catch( InterruptedException ie ){
                }
            }
        }
}
/****** END OF com/ibm/osg/example/mygetservice/MyBundleActivator.java ******/
```

For an example that uses ServiceTrackers and getting services, refer to the Pizza servlet sample source code (`PizzaBundle.java`). The Pizza servlet sample resides in `TECHHOME\smf\client\samples\source\src-pizza.zip` or in the Sample Bundles project.

Sample applications

Sample bundles reside in the `TECHHOME\smf\client\samples` directory. You can use the sample code to develop your own bundles. For information on installing the sample bundles, refer to “Creating a sample SMF project” on page 2.

Note: In the samples below, *nn* represents the port number that you used to launch Service Management Framework.

Pizza servlet sample

The Pizza servlet uses the Http Service to display the results of input from radio buttons and the output of a calculation viewed through a browser. This sample also demonstrates the use of the ServiceTracker and ServiceTrackerCustomizer. You can access this sample at `http://localhost:`*`nn`*`/pizza` .

Handle security sample

This sample demonstrates how to use the User Admin Service and the Service Tracker.

The handleSecurity sample provides an example of a default handleSecurity plug-in. The handleSecurity method implements basic authentication and uses the UserAdmin Service to validate user names and passwords. After you install this plug-in, the handleSecurity method of the plug-in authenticates any resource or servlet that is registered with the default Http Context.

This sample provides an implementation of the handleSecurity method for the Default Http Context in the IBM Http Service. Both smfadmin and fileadmin use the Default Http Context. By default this sample is installed and active. This sample authenticates access to smfadmin and fileadmin using the User Admin Service to authenticate the user name and password. If this sample is not active, then it cannot authenticate access to smfadmin and fileadmin.

Simple permission policy sample

This sample uses the Permission Admin service to configure the permission of a bundle. This sample reads the META-INF/permission.info resource in each installed bundle and calls Permission Admin to set those permissions for the bundle.

Simple sample

The simple sample demonstrates how a BundleActivator sends a log message to the standard OSGi LogService when the service is started and stopped. If the LogService is not available, then the service writes a message to `System.out`.

Driver locator sample

A DriverLocator returns a list of driver IDs based on the properties of the device and the input stream for the driver ID. The following device.properties configuration file sample maps a property name to a value, driver ID, and location in an XML input file. You can map multiple Driver IDs to a single property name-value pair.

```
<?xml version="1.0" encoding="UTF-8"?>
 <driverlocator>
   <devicepropertypair property="FooAppliances" value="Oven">
         <driveridlocation driverID="343Oven" driverLocation="http://www.foo.com/drivers/343Oven" />
      <driveridlocation driverID="293Oven" driverLocation="http://www.foo.com/drivers/293Oven" />
   </devicepropertypair>
   <devicepropertypair property="FooAppliances" value="Toaster">
       <driveridlocation driverID="400Toaster" driverLocation="http://www.foo.com/drivers/400Toaster" />
   </devicepropertypair>
 </driverlocator>
```

This file resides in the SimpleDriverLocator bundle storage location.

Web Application Bundle samples

PizzaJSP sample

PizzaJSP is a sample of a Web Application project that uses Java Server Pages and Servlets. This project requires WebSphere Studio Application Developer or WebSphere Studio Site Developer tools to compile and produce a WAR file. You can use the WAR file as input into the WAB tool to produce a WAB. You can access this sample at `http://localhost:`*`nn`*`/PizzaJSP`.

For more information about WABs and the WAB tool, see "Web application development" on page 74.

Web Log sample

Web Log is a sample of a Web Application project that accesses the OSGi environment. This project requires WebSphere Studio Application Developer or WebSphere Studio Site Developer tools to compile and produce a WAR file. You can use the WAR file as input to the WAB tool to produce a WAB.

You can access this sample at `http://localhost:`*`nn`*`weblog`. For more information about WABs and the WAB tool see "Web application development" on page 74.

Note: If you install these projects in WebSphere Studio Device Developer, some errors about missing natures might be displayed. This occurs because the projects require WebSphere Studio Application Developer or WebSphere Studio Site Developer tools.

SMF Bundle Server

After you read "Bundle resolution" and "Bundle identification and version matching" to learn about the SMF bundle server, read "Using SMF Bundle Developer Tools" on page 17 for steps to complete tasks with the bundle server.

Bundle resolution

The first time an OSGi™ bundle registers with a bundle server, the bundle server examines the bundle and extracts and stores meta information. The bundle server uses the meta information to determine whether the bundle differs from other bundles stored on the bundle server. If the bundle server cannot differentiate the bundle from other bundles, the server rejects the bundle because it is not unique.

Devices determine which bundle to download based on the differences between the bundles. In Service Management Framework, bundles are uncompressed JAR files that contain a set of files and file formats. The bundle server is not sensitive to any of these files and file formats. However, Service Management Framework requires bundles to contain entries that enable Service Management Framework to provide a rich set of bundle selection and device matching capabilities.

Refer to "Bundle identification and version matching" for more information about how SMF uses these entities to select bundles and match bundles to devices.

Bundle identification and version matching

When an SMF Runtime issues an InstallSet request to the SMF Bundle Server, the request includes information that the server uses to install the appropriate bundles and versions. This information includes:

- The name of the bundle that the runtime wants to download
- The target attributes configured in `IVEATTRS.XML`.
- The set of bundles that are currently installed in the runtime.
- The current set of resources available on the target, such as Flash or the number of sockets, listed in the `IVERES.XML` file.

When the bundle server receives an InstallSet request, the bundle server determines whether the requested bundle exists in the bundle server repository. If the bundle does not exist in the repository, the bundle server returns an exception status to the runtime. If the bundle exists in the repository, the bundle server determines whether the bundle requested by the target is already installed in the runtime. If the target has already installed the bundle, the bundle server determines whether there is a later version of the bundle available on the bundle server. If the runtime already has the latest version of the requested bundle installed, the bundle server terminates the request.

The bundle server determines which bundle to download by using a matching algorithm. The bundle server applies the algorithm against all potential bundle candidates that match the name requested by the target. Then, the algorithm compares the target attributes in the `IVEATTRS.XML` file with the attributes in the installation request. If the bundle attributes and the bundle name in the repository match the bundle attributes and bundle name requested by the runtime, the bundle server downloads the bundle from the repository to the runtime.

After the bundle server then verifies which bundle to download, the bundle server verifies whether or not the target has appropriate access rights to download the bundle. If the target does not have appropriate access rights, the bundle server returns an exception to the runtime with a status code that indicates that the runtime has insufficient access rights to download the requested bundle.

When a runtime requests a bundle and two or more bundles with the same name satisfy the runtime configuration, the bundle server uses the following criteria to determine which bundle to install:

- The bundle server always chooses the latest version of a bundle over an earlier version assuming that the attributes of the latest version of the bundle match the attributes required by the runtime.
- If versions of the latest version of the bundle match the attributes of an earlier version of the bundle, the bundle server uses a system to determine which attributes are more valuable. In the value system, a bundle that explicitly states an attribute required by the target configuration is given a higher rating than a bundle for which that same attribute is unspecified.
- If the bundle developer provides an attribute, the value of the attribute must match that value of the attribute in the runtime configuration. If the bundle developer does not provide an attribute, the bundle remains as a potential download candidate. However, the bundle server supersedes the download candidate with any other candidate bundle that explicitly specifies a matching value for the attribute. If the matching algorithm is applied in the order of the attributes, then the bundle server chooses the bundle that explicitly specifies an attribute value that matches the runtime's attribute value and the bundle whose attribute is highest on the attribute list. Refer to Table 17 for a list of the bundle attribute options contained in the `IVEATTRS.XML` file.

SMF Bundle Developer Tools

This section provides reference information for using the SMF Bundle Developer Tools.

Default views in the SMF perspective

The Service Management Framework workbench includes the following default views within the SMF perspective:

Preferences

You can access the following preferences on the main SMF preferences page by selecting **Window** -> **Preferences** and clicking on **SMF** in the left panel.

Table 4. SMF Preferences

Option	Description
External Targets	Represents a file system directory to which you can submit and export bundles.
Add	Opens a file dialog where you can select a directory for the external targets.
Remove	Removes the selected target or targets from the list.
Action on double click in the SMF Bundle Servers view	Enables you to specify which action occurs when you double-click on a bundle: show the bundle contents or install the bundle on the runtime.

Editor preferences

You can access the following preferences on the SMF Editor preferences page by selecting **Window** -> **Preferences** and navigating to the **SMF** -> **Editor** page in the left panel.

The Editor Preferences dialog contains the following pages:

IveAttrs: The fields on this page provide the initial values for entries in the Bundle Attributes editor.

IveRes: The fields on this page provide options that enable you to specify which sections in the Bundle Resources editor are expanded when you open the editor.

Manifest: The fields on this page provide options that enable you to specify which sections in the Bundle Resources editor are expanded when you open the editor. The fields on this page provide the initial values for entries in the Bundle Manifest editor.

Runtime preferences

You can access the following preferences on the SMF Editor preferences page by selecting **Window** -> **Preferences** and navigating to the **SMF** -> **Runtime** page.

Note: The runtime preferences you change affect only new SMF Runtime launches. Any SMF Runtime launches that you created before you changed the preferences continue to use the original preferences.

Table 5. Runtime preferences

Option	**Description**	**Default**
Hardware Type	Specify the target hardware type.	pc
Hardware Model	Specify the target hardware model.	clone
Hardware Version	Specify the version of the target hardware.	unknown
Processor	Specify the target processor type. Refer to the OSD specification for valid values.	x86
Endian	Specify the output type. Valid values are `le` for little endian or `be` for big endian.	le
Address Length	Specify the length of the address. Valid values are 32 or 64.	32
OS	Specify the target operating system. Refer to the OSD specification for valid values.	
OS Version	Specify the version of the target operating system. Refer to the OSD specification for valid values.	
Language	Specify the ISO 639 language code of the target. Refer to the ISO 639 specification for valid values.	en [English]
Country	Specify the A2 ISO 3166 country code of the target. Refer to the A2 ISO 3166 specification for valid values.	blank

Bundle Management:

To access the SMF Bundle Management preferences page, select **Window** -> **Preferences** and navigate to the **SMF** -> **Client** -> **Bundle Management** page.

Submission preferences

You can access the SMF Submission preferences page by selecting **Window** -> **Preferences** and navigating to the **SMF** -> **Submission** page.

In the Export File Names section of the Submission Preferences editor, you can specify the format of the .jar and .jxe filenames. In the Build Information section, you can set options for setting build information in the manifest or JAR files. In the JXE VM Options section, you can set options to suppress warning users when the VM field in the IVEATTRS.XML file had to be explicitly set. In the Files to Exclude section, you can list file names that you want to exclude from being copied into the bundle when you submit the bundle. For example, you might choose to prevent the .classpath file from being added when you include a project in a bundle.

Views

SMF uses a variety of views and editors.

Package Explorer view

The Package Explorer view enables you to modify the source code of your bundles. Refer to the following table for a list of the Package Explorer view commands, locations, and descriptions.

Table 6. Package Explorer view commands

Command	**Location**	**Description**
Create Bundle Folder	button on the main toolbar	This command enables you to create a new bundle.
Submit Bundle	context menu	This command enables you to submit bundle to a target, such as a server or file system directory.
Hide non-bundle projects	local toolbar	This filter enables you to view only those projects that contain bundles.
Re-Cache	context menu	This option recaches the model for the projects you selected.
Re-Cache All	context menu	This option recaches the model for the entire workspace.
Validate	context menu	This option validates the selected bundle files.
Validate All	context menu	This option validates all bundles in the workspace.

SMF Runtime view

Use the view filter buttons in this view to display information about bundles, services, or packages that are currently installed or running in a runtime.

Table 7. SMF Runtime view filters

View Filter Button	Description
Show Bundles	This filter button refreshes the view to show the current state of all bundles. **Note:** The number next to the bundle represents the bundle ID.
Show Services	This filter button refreshes the view to show all registered services. **Note:** The number next to the service represents the service ID.
Show Packages	This filter button refreshes the view to show all available packages.

Refer to the following table for a list of views available from the drop-down menu.

Table 8. SMF Runtime view commands available from the drop-down menu

Command	Description
Show Full Path	Toggles the display of the full path of the services.
Show Runtime Info	Toggles the display of the runtime information, such as the location of the runtime. This option also toggles the display of the text field and Install button necessary for installing bundles directly from the file system.
Show Runtime Properties	Opens the Runtime Properties editor, where you can view the IVE properties, system properties, and SMF properties for the runtime.
Disconnect	Disconnects the IDE from the SMF runtime. This option does not terminate the runtime or affect the OSGi framework.
Terminate Runtime	Closes the runtime and terminates the runtime's Java Virtual Machine.
Bundle Garbage Collect	Perform a global bundle garbage collection and uninstalls the bundles that are not used or needed as dependents. The option determines which bundles are dependents based on package and service dependencies. Bundles that were installed explicitly are members of the root set and are not candidates for garbage collection. The garbage collection only removes bundles that were installed as prerequisite bundles. **Note:** This command is only available if the DependencyAdmin bundle is installed.
Shutdown Framework	Shuts down the OSGi framework.
Launch Framework	Launches the OSGi™ framework.
Store Snapshot	Saves a snapshot of the runtime.
Restore Snapshot	Restores the runtime to the previous state.

Refer to the following table for a list of views available from the shortcut menu.

Table 9. SMF Runtime view commands available from the shortcut menu

Command	**Description**
Open Bundle	Opens a read-only, multi-page Properties editor that includes the Manifest, Resource, and Attributes editors.
Properties	Displays the properties of the selected item, such as service information.
Show Exception	Displays the exception that issued during the last bundle state change. This action is only enabled for bundles that have a red error mark.
Update	Updates the selected bundle with the latest version available from the server.
Stop	Stops the selected bundle.
Start	Starts the selected bundle.
Uninstall GC	Uninstalls the specified bundle and performs a local bundle garbage collect on the dependent bundles. This option collects only those bundles affected by the uninstalled bundle and uninstalls only the bundles that are not used or needed as dependents. The dependency criteria is based on package and service dependencies. Bundles that were installed explicitly from the file system are members of the root set and are not candidates for garbage collection. This option only considers bundles that were installed as prerequisite bundles by the bundle server for garbage collection. **Note:** This command is only available if the DependencyAdmin bundle is installed.
Uninstall	Uninstalls the selected bundle.

Runtime log view

The Runtime Log view displays the contents of the log from the SMF Runtime. The Runtime Log view uses the OSGi™ Log Reader on the runtime to access the log and log events on the SMF Runtime. You must install the LogService bundle for this view to show log entries.

The following table describes the commands that are available with the Runtime Log view:

Table 10. SMF Runtime Log view commands

Command	**Location**	**Description**
Clear Log	toolbar button	Clears the contents of the log view. This command does not affect the log on the runtime.
Refresh Log	toolbar button	Refreshes the log with the current contents of the runtime log.

SMF Bundle Servers view

This view shows the available SMF Bundle Servers and the bundles, users, stations, and snapshots within them.

Table 11. SMF Bundle Servers view commands

Command	**Location**	**Description**
New	context menu	Adds a new SMF bundle server, station, or user.
Refresh	toolbar button	Refreshes the SMF bundle servers view.
Remove	context menu	Removes all selected servers, stations, users, and snapshots.
Remove all versions	context menu	Removes all versions of the selected bundle from the server.
Show versions	context menu	Displays all versions of the selected bundle in the Bundle Details view.
Import Bundles	context menu	Imports bundles into the currently selected server from a different server or a file system target.
Export	context menu	Exports the selected bundle to a location selected in the resulting dialog.
Properties	context menu	Enables the user to view the properties of a user or a station.
Export All Bundles	context menu	Exports all bundles to the location you select in the dialog.
Stop	context menu	Terminates the bundle server you selected.
Install	context menu	Installs the selected bundle to the SMF runtime

Bundle Details view

The Bundle Details view enables you to view the following list of bundle details:

- name
- version
- whether or not the bundle is enabled
- processor type
- address length
- whether big or little endian
- operating system
- operating system version
- virtual machine type
- implementation type
- language
- country

The following commands are available from the context menu in this view.

Table 12. Bundles Details view commands

Command	**Description**
Remove	Removes the selected bundle.
Export	Exports the selected bundle to the location you select in the resulting dialog.

Table 12. Bundles Details view commands (continued)

Command	Description
Disable	Disables the selected bundle if the bundle is enabled.
Enable	Enables the selected bundle if the bundle is disabled.
Open Bundle	Opens a read-only multi-page editor that includes the Manifest, Resource, and Attributes editors, and an editor that shows information about the bundle on the server.

Snapshot Details view

The following commands are available from the shortcut menu in this view.

Table 13. Snapshot Details view commands

Command	Description
Remove	This command removes the selected snapshot from the server.
Show Contents	This command displays the details of the selected snapshot in the Bundle Details view.

Editors

Service Management Framework provides the following editors:

Bundle Manifest editor

The Bundle Manifest editor enables you to define information in the `MANIFEST.MF` file. This file describes the packages and services imported and exported by a bundle. The set of import and export specifications determine the prerequisites for the bundle and identify which bundles satisfy the prerequisites.

The Bundle Manifest editor contains a list of predefined keys.

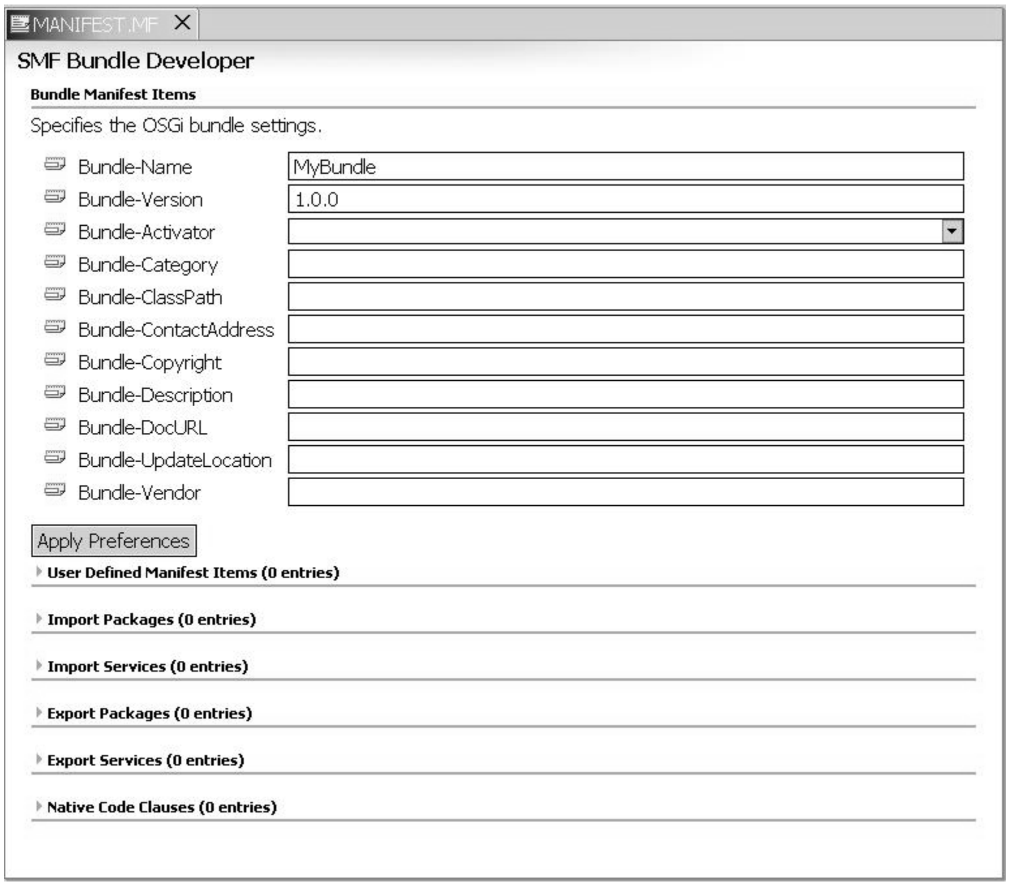
You can set a default value for several of the manifest entries by setting the preferences for the SMF Manifest editor. These preferences display in the Bundle Manifest editor when you create a new manifest file. See "Editor preferences" on page 47 for more information about setting the preferences for the editor.

To set the Bundle Manifest preferences, select **Window** -> **Preferences** from the menu bar, then navigate to the **SMF** -> **Editor** -> **Manifest page**.

Apply Preferences:

The **Apply Preferences** button in the middle of the Bundle Manifest editor sets the values in this editor based on the settings on the Bundle Manifest section of the Editor Preferences page.

Bundle manifest items: The editor uses the manifest information you specify to create the `MANIFEST.MF` file for the bundle. You can access the Bundle manifest editor by double-clicking on the MANIFEST.MF file. The following image represents the Bundle manifest editor.


MANIFEST.MF
SMF Bundle Developer

Bundle Manifest Items

Specifies the OSGi bundle settings.

Bundle-Name	MyBundle
Bundle-Version	1.0.0
Bundle-Activator	
Bundle-Category	
Bundle-ClassPath	
Bundle-ContactAddress	
Bundle-Copyright	
Bundle-Description	
Bundle-DocURL	
Bundle-UpdateLocation	
Bundle-Vendor	

Apply Preferences
User Defined Manifest Items (0 entries)
Import Packages (0 entries)
Import Services (0 entries)
Export Packages (0 entries)
Export Services (0 entries)
Native Code Clauses (0 entries)

The following table lists the options within the Bundle Manifest editor.

Table 14. Bundle manifest items

Option	Description
Bundle-Name	Represents the bundle name that identifies the software component represented by the bundle.
Bundle-Version	The SMF bundle management extensions use this value to identify the most recent version of a bundle. Specify this value in the following format: `major`, `minor`, `micro`. (Where `major` represents the major version number, `minor` represents the minor version number, and `micro` represents the micro version number.)
Bundle-Activator	Specifies the name of the class within the bundle that implements the BundleActivator interface.
Bundle-Category	Enables bundles to be listed in different categories.
Bundle-ClassPath	Describes the classpath within the bundle.
Bundle-ContactAddress	Represents the e-mail address of a person that a user can contact regarding the bundle.
Bundle-Copyright	Represents the copyright year for the bundle.
Bundle-Description	Specifies the purpose or function of the bundle.
Bundle-DocURL	Specifies a URL that contains additional information about the bundle.
Bundle-UpdateLocation	Specifies the location where an updated version of the bundle resides.

Table 14. Bundle manifest items (continued)

Option	Description
Bundle-Vendor	Identifies the vendor of the bundle.

User-defined Manifest items:

The following tags are reserved and cannot be added to the list of manifest keys:
- Import-Package
- Export-Package
- Import-Service
- Export-Service
- Specification-Version
- Bundle-NativeCode

Refer to the OSGi specification provided in the `TECHHOME\smf\client\docs\OSGi` directory, or visit http://www.osgi.org for more information about these manifest keys.

Import/Export Manifest packages and services: You can specify packages and services to import and export with this editor. Refer to "Packages" on page 11 for more information about packages. Refer to "Services" on page 10 for more information about services.

Table 15. Import/Export manifest items

Manifest Editor Section	Description
Export Packages	Describes packages from the bundle that other bundles can use.
Export Services	Describes services that other bundles can use.
Import Packages	Describes the packages that the bundle requires. Another bundle must export these packages.
Import Services	Describes the services the bundle can use.

- The **Add** button opens a dialog that lists the available choices for each section.
- The **Remove** button removes any items that are selected in the box next to them.
- The **Compute** button, which is available only in the Import Packages section, removes all currently-imported packages and replaces the current list of imports with a new list of required imports based on class references within the bundle's source code.

Native code clauses: The **Native Code Clauses** section describes the files that contain the implementation of the native methods used by classes in the bundles.

You can create a clause for native paths by clicking the **Add** button, which creates a blank entry in the editor that comprises several fields, such as native path parameters.

Each field has a drop-down list of known or commonly used values that you can choose from, and you can add your own custom values.

Note: Some custom values might cause Service Management Framework Bundle Developer to display a warning.

The **Add** button enables you to select files contained within the scope of the bundle defined in the `.bndldesc` file. **Remove** enables you to remove the selected files from the table.

Selecting the box next to a native code clause identifies which clause to delete when you select the **Remove** button.

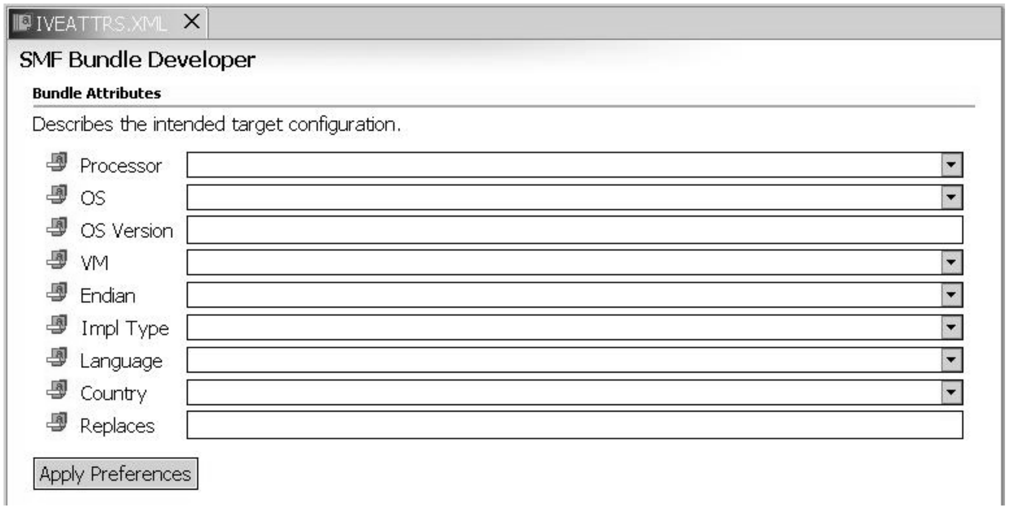
Table 16. Native Code Parameters

Parameter	Description
processor	Identifies the architecture of the target. Refer to the OSD specification for a list of valid values. You can specify more than one value by separating values with commas.
osname	Identifies the operating system of the target. Refer to the OSD specification for valid values. You can specify more than one value by separating values with commas.
osversion	Identifies the version of the operating system of the target.
language	Specifies the ISO 639 language code of the target. Refer to the ISO 639 specification for valid values. You can specify more than one value by separating values with commas.
Edit button	The **Edit** button opens a selection dialog that enables you to manage the list of files for the selected environment.

Bundle Attributes editor

The Bundle Attributes editor enables you to define the bundle's attribute information, contained in the `IVEATTRS.XML` file. This file contains a set of attributes, formatted in XML, that describes the intended target of the bundle. More specifically, these attributes enable you to describe the properties of a target device, and as a result, you can limit the types of target devices that can download your bundle. If you specify the required attributes, the bundle server ensures that only target devices that match those attributes will receive your bundle in response to an installation request.

Refer to the following image for a depiction of the Bundle Attributes editor. Then, refer to the following table for a list of the bundle attributes that you can configure with this editor. The editor uses the attribute information you enter to create the `IVEATTRS.XML` file, which the bundle server uses to select the best bundle for a particular platform.


IVEATTRS.XML

SMF Bundle Developer

Bundle Attributes

Describes the intended target configuration.

Processor	
OS	
OS Version	
VM	
Endian	
Impl Type	
Language	
Country	
Replaces	

Apply Preferences

Table 17. Bundle attribute information options

Option	Description
Processor	Identifies the architecture of the target. Refer to the OSD specification for a list of valid values.
OS	Identifies the operating system of the target. Refer to the OSD specification for a list of valid values.
OS Version	Identifies the version of the target operating system.
VM	Identifies the Java Virtual Machine that is running on the target.
Endian	Specifies whether the processor of the target must run in little-endian or big-endian mode if the bundle contains natives for only one type. Valid values are `be` for big endian or `le` for little endian.
ImplType	Identifies the base Java class library subset on the runtime. Valid values are `JCL_Foundataion_1_3`, `JCL_GatewayPlus_1_3`, and `JCL_Max_1_3`.
Language	Specifies the target's ISO 639 language code. Refer to the ISO 639 specification for valid values.
Country	Specifies the target's A2 ISO 3166 country code. Refer to the A2 ISO 3166 specification for valid values.
Replaces	This is a reserved value.

Click **Apply Preferences** in the middle of the Bundle Attributes editor to set the values in this editor based on the settings on the Bundle Attributes section of the Editors preferences page. To access the Editors preferences page, select **Window** -> **Preferences**, from the menu bar and navigate to the **SMF** -> **Editor** -> **IveAttrs page**.

Bundle Resources editor

The Bundle Resources editor enables you to define the bundle's resource information, which resides in the `IVERES.XML` file. The `IVERES.XML` file is an optional entity for an OSGi bundle. This file contains a set of resource specifications that determine whether sufficient resources exist on a target device at the time of a download. The bundle server compares the resources that the bundle requires with the resources available on the target device. For example, the bundle server would not download a bundle that needs 300K bytes of RAM to a device that has only 200K bytes of RAM available.

You can specify which actions SMF takes when a resource is unavailable. Because RAM and space requirements vary among operating systems, you can specify the actions SMF takes based on a specific operating system.

Refer to the following table for a list of the predefined resources that you can specify with the Bundle Resources editor.

Table 18. Predefined resource definitions

Option	Description
New Space Size	The number of bytes to be allocated in each new space region of the bundle's memory space.
Old Space Size	The number of bytes to be allocated for tenured objects in the bundle's memory space.
Files	The total number of files a bundle can create in its data directory.
Quota	The number of bytes of storage within the local filesystem required by the bundle.
Sockets	The maximum number of socket connections the bundle may have open at any one time.
Threads	The number of threads the bundle may create at any one time.

The following tags are reserved, and therefore cannot be added to the list of resources:
- BundleStorage
- Quota
- TotalRAM

Click **Add** to create a new user-defined key and assign a value to the key. You can expand and collapse the user defined resource items by the user. The Preferences editor enables you to specify whether or not the list is collapsed when you open the editor. You can check the box next to any user-defined key to identify the key as one to be removed when you click **Remove**.

Bundle description editor

This file describes the content of a bundle similar to the way the `.jardesc` file describes the content of a Java archive (JAR) file. However, you must create the `.bndldesc` file before you create the bundle. By contrast, the `.jardesc` file is an output result of creating a JAR file.

The `.bndldesc` file affects the contents of the JAR/JXE file when you submit the bundle and the `.bndldesc` file is actively involved in the validation and problem determination for bundles.

Complete the following steps to view the `.bndldesc` file:
1. In the Package Explorer view, click the menu drop-down arrow and select **Filters**.
2. On the **Java Elements Filter** page, deselect the **Hide *.files** filter.
3. Double-click the `.bndldesc` file.

By default, the `.bndldesc` file identifies the source folder that is the bundle folder. Because the default file contents are sufficient in most cases, you do not need to modify the `.bndldesc` file. However, in cases where you might share or factor code, you might want to include resources in the bundle even when those resources are not directly in the bundle's source folder. For example, in order for the bundle to export a package, the bundle must contain that package. Therefore, the exported package must reside in the bundle's own source folder or be accessible through an entry in the bundle's `.bndldesc` file.

Refer to the following example for the format of the `.bndldesc` file:

```
<!ELEMENT bndldesc (include)>
<!ELEMENT include (exclude)*>
<!ATTLIST include

path CDATA #REQUIRED
>

path CDATA #REQUIRED
>
<!ATTLIST exclude

path CDATA #REQUIRED
>
```

path is a slash-delimited (/) path to the resource.

Runtime Properties editor

You can use the Runtime Properties editor to edit the current values of all of the Java system properties, including the properties loaded from the `smf.properties` file. You can also use the Runtime Properties editor to modify the properties that the runtime uses when the runtime launches.

Refer to the reference documentation for the J9 virtual machine for more information about the system properties you can modify. Refer to the SMF Runtime documentation for more information about the SMF specific keys.

Wizards

The SMF workbench component provides a variety of wizards for working with bundles.

New Extension Services Bundle Project Wizard

The New Extension Services Bundle Project Wizard is the recommended way to create a new bundle in SMF. Refer to the following tables for a list of New Extension Services Bundle Project Wizard options and their default values, if any.

Table 19. Extension Services Bundle Project options

Option	**Description**	**Default value**
Project name	Enter a name for your new Extension Services Bundle Project	No default value
Project Contents	You may de-select ″Use default″ and click Browse to select a filesystem location for your new Extension Services Bundle Project	The ″Use default″ option creates the project in your current workspace

This is followed by the Platform Profile page:

Table 20. Platform profile options

Option	Description	Default value
Platform Profile	Select from the list the Platform Profile this Extension Services Bundle Project will target. You can change your selection later in the Extension Services property page.	No default value
Application Services	Check the Application Services that your Extension Services Bundle Project will require. You can change your selection later in the Extension Services property page. Grey entries are required by the Platform Profile and cannot be un-checked.	The ″Core OSGi Interfaces″ Application Service is required by all Platform Profiles.

This is followed by the Java Settings page. This page allows you to configure Java Development Tools options for the new project. For more information see the *Java Development User Guide*.

This is followed by the Extension Services Options page:

Table 21. Extension Services Options

Option	Description	Default value
Create Default BundleActivator	Create a class which implements org.osgi.framework.BundleActivator, and add the corresponding Bundle-Activator entry in the Bundle Manifest.	not selected
Use Platform profile to manage Java Build Path	Add the appropriate libraries to your Java Build Path for the Application Services you have selected. This setting can be changed later in the project's Extension Services property page.	Selected
Automatically manage Import-Package key in Bundle Manifest	Automatically add or remove Import-Package entries from the Bundle Manifest as you use Application Services in your bundle. This setting can be changed later in the project's Extension Services property page.	Selected

Convert Java to Extension Services Bundle Project Wizard

The Convert Java to Extension Services Bundle Project Wizard allows you to convert an existing Java Project to an Extension Services Bundle Project. Refer to the following tables for a list of Convert Java to Extension Services Bundle Project Wizard options and their default values, if any.

Table 22. Convert Java to Extension Services Project page

Option	Description	Default value
Java Project selection box	Select a Java Project from the list to convert to an Extension Services Bundle Project.	If a Java Project was selected when you launched this wizard, it will already be selected on this page.

Table 22. Convert Java to Extension Services Project page (continued)

Option	Description	Default value
Copy before creating	Check this option to create a copy of the existing Java Project before converting it to an Extension Services Bundle Project. The original project will be unchanged.	Not checked - converts the existing project
New name	This option only appears if you select the ″Copy before creating″ option. Enter a name for the new project.	No default

This is followed by the Platform Profile page:

Table 23. Platform Profile page

Option	Description	Default value
Platform Profile	Select from the list the Platform Profile this Extension Services Bundle Project will target. You can change your selection later in the Extension Services property page.	No default value
Application Services	Check the Application Services that your Extension Services Bundle Project will require. You can change your selection later in the Extension Services property page. Grey entries are required by the Platform Profile and cannot be un-checked.	The ″Core OSGi Interfaces″ Application Service is required by all Platform Profiles.

This is followed by the Extension Services Options page:

Table 24. Extension Services Options page

Option	Description	Default value
Create Default BundleActivator	Create a class which implements org.osgi.framework.BundleActivator, and add the corresponding Bundle-Activator entry in the Bundle Manifest.	Not selected
Use Platform profile to manage Java Build Path	Add the appropriate libraries to your Java Build Path for the Application Services you have selected. This setting can be changed later in the project's Extension Services property page.	Selected
Automatically manage Import-Package key in Bundle Manifest	Automatically add or remove Import-Package entries from the Bundle Manifest as you use Application Services in your bundle. This setting can be changed later in the project's Extension Services property page.	Selected

New Bundle Folder wizard

The New Bundle Folder wizard is a deprecated way to create a new bundle in SMF that is retained for backwards compatibility. It is recommended to use the New Extension Services Bundle Project Wizard instead. Refer to the following table for a list of the New Bundle Folder wizard options and their default values, if any.

Table 25. New bundle wizard options

Option	Description	Default value
Available Bundle Folder containers	Select a source folder or project that does not contain a META-INF folder.	The default selection is based on the selection in the Package Explorer view.
IVEATTRS.XML	Select the checkbox to create the bundle attributes file.	selected
IVERES.XML	Select the checkbox to create the bundle resources file.	selected

Import Bundles wizard

This wizard enables you to import bundles into a Bundle Server from one or more sources. You can expand the hierarchy and select the bundles you want to import. You can access this wizard from the context menu of a selected server in the SMF Bundle Servers view.

Select **Replace Bundles** to overwrite any existing bundles with the bundles being imported of the same name.

Click **Add Directory** to add additional directories to the list of valid import targets.

Click **Add Server** to launch the New SMF Bundle Server wizard where you can define a new server.

Export Bundles wizard

The Export Bundles wizard enables you to export bundles from a Bundle Server to another Bundle Server or to the file system.

To access the Export Bundles wizard, select the bundles you want to export in the SMF Bundle Servers view, then select the Export Bundles wizard from the context menu.

Select **Replace Bundles** to overwrite any existing bundles with the bundles being exported of the same name.

Click **Add Directory** to add additional directories to the list of valid export targets.

Click **Add Server** to launch the New Server wizard where you can define a new server.

Submit Bundle wizard

This wizard enables you to submit a bundle to the Bundle Server or file system. This wizard is available from the context menu of a bundle folder.

Refer to the following table for a list of the Submit Bundle wizard options and their descriptions.

Table 26. Submit Bundle options

Option	Description
Submit JAR	Submit the bundle application in JAR format. You can select this option alone or in addition to the **Submit JXE** option if you want to submit the bundle in more than one format.
Submit JXE	Submit the bundle application in JXE format. For more information about the JXE format, refer to the J9 and SmartLinker documentation.
Output types	Select the types of output that you want to include in the JXE file for this bundle application. To submit more than one kind of JXE file, select as many JXE file output types as you want.
jxeLink Options	Indicate the SmartLinker options that the wizard uses when the JXE file is created. **Note:** `jxelink` is the command-line executable for the SmartLinker tool. For more information about the jxelink tool, refer to the J9 and SmartLinker documentation.
Export Targets	This list includes all the locations to which the selected application can be submitted. Click **Add Directory** to add additional directories to the list of valid export targets. Click **Add Server** to launch the New Server wizard where you can define a new server. **Note:** You must select **Add Directory** or **Add Server**.
Replace Bundles	Select this option to override any existing bundles with bundles being submitted that have the same name.
Generate ANT Script	Select this option and specify the name and location to generate an Ant script that you can re-run any time to submit the bundles using the same specifications in the wizard.

New SMF Bundle Server wizard

This wizard helps you create a new SMF bundle server connection. Refer to the following table for a list of the SMF Bundle Server wizard options, descriptions, and default values, if any.

Table 27. New bundle server options

Option	Description	Default
Host	Specify the host name for the server.	
Port	Specify the port number for the server.	8080
URL Prefix	Defines the location on the Web server where the SMF bundle server resides. **Note:** In most cases, you do not need to modify the default value.	/smf
User	Specify a username for the server.	
Password	Specify a password for the server.	

New Station wizard

This wizard enables you to create a new SMF station. When an SMF Bundle Server communicates with multiple runtimes, the Bundle Server distinguishes one runtime from another with the Station ID of the runtime. Refer to the Bundle Server documentation for more information about the Station ID.

Refer to the following table for a list of the New Station wizard options and their descriptions.

Table 28. New Station Options

Option	Description
Station ID	Specify the new SMF station identification number.
Bundle Servers	Select a server from the list.

New SMF User wizard

This wizard enables you to create a new SMF user. Refer to the following table for a list of the New SMF User wizard options, descriptions, and default values, if any.

Table 29. New User options

Option	Description	Default
User ID	Specify a username for the new user.	
Password	Specify a password for the new user.	
First Name	Specify the first name of the new user.	
Last Name	Specify the last name of the new user.	
Administrator	Assign the new user Administrator privileges.	Off
Developer	Assign the new user Developer privileges.	Off
Runtime	Assign the new user Runtime privileges.	On
Bundle Servers	Select a server from the list.	

Restore Snapshot wizard

This wizard enables you to load a snapshot onto the runtime. The wizard stops and uninstalls all bundles from the runtime before installing the new bundles. The load snapshot feature enables you to revert to a previous configuration of bundles without having to remember the exact bundles you installed. You can browse for a list of possible snapshots and their contents from the SMF Bundle Servers view.

Refer to the following table for a list of the Load Snapshot wizard options and their descriptions.

Table 30. Restore Snapshot options

Option	Description
Snapshot Name	Specify the name of the snapshot to install from the current SMF bundle server.

Store Snapshot wizard

This wizard enables you to save a snapshot of the runtime. Storing a snapshot enables you to save your current configuration of bundles for later use. The runtime bundles are not affected by this action, and the server will store the list of currently installed bundles along with the description of the snapshot you supply.

Refer to the following table for a list of the Store Snapshot wizard options and their descriptions.

Table 31. Store Snapshot options

Option	Description
Snapshot Name	Specify the name you want to use to store the current configuration. The Load Snapshot wizard uses this name when you restore the snapshot.
Snapshot Description	Specify a short description that describes the purpose of this configuration.

Dialogs

Service Management Framework includes the following dialogs:

SMF Runtime Launch Configuration dialog

This dialog enables you to configure the runtime launch options. This dialog contains six tabs; however, only one is specific to SMF. For information about the other tabs in this dialog, refer to the WebSphere Studio Device Developer Java support documentation.

The SMF tab contains three sub-tabs:
- Bundle Server
- Platform
- Misc

Bundle Server tab: This tab enables you to configure which SMF bundle server you want the runtime to communicate with. This is the server from which bundles are installed and snapshots are saved to and loaded from.

Refer to the following table for a list of the Bundle Server tab options, descriptions, and their default values.

Table 32. SMF Runtime Launch Configuration dialog: Bundle Server tab options

Option	Description	Default
Hostname	Specifies the host on which the SMF bundle server is running.	localhost
Port	Specifies the port on which the server is listening.	8080
Webapp	Specifies the URL prefix for this server.	/smf
User ID	Specifies the user ID to use when opening a session to the server.	guest
User Password	Specifies the password for the server.	password
Station ID	Specifies the station ID that this runtime uses.	78

Platform tab: The platform tab enables you to define how the bundles are stored on the runtime. Refer to the following list of the settings you can specify for bundle storage:
- Use file-based bundle storage

  Use this option if you do need to configure flash management and bundle size. Refer to the following table for a list of the file-based bundle storage options and their descriptions.

Table 33. SMF Runtime Launch Configuration dialog: file-based bundle storage options

Option	Description
Location	Specifies the directory that contains the bundle files. Leave this value blank to use the current directory.
Reset File before launched	Clears any previous bundles from the file system before the runtime is started.

- Use other platform

  This is an advanced option that enables you to launch the runtime with your own custom platform object. You must add the platform implementation to the runtime classpath through the classpath tab before you can launch successfully. Refer to the following table for a list of the other platform options and their descriptions.

Table 34. SMF Runtime Launch Configuration dialog: other platform option

Option	Description
Platform classname	Specifies the name of the class to use that provides bundle storage support
Platform options	This section enables you to pass options into the custom platform object.

- Misc tab

  This tab is a miscellaneous collection of options that you can configure. Refer to the following table for a list of the miscellaneous options, descriptions, and their default values, if any.

Table 35. SMF Runtime Launch Configuration dialog: Misc tab

Option	Description	Default
Flash bundles data directory	Specifies the directory in which the FlashPlatform persistently stores the configuration data and permission information.	bundlesData
Default HTTP port	Defines the HTTP ports on which the http service listens.	
Enabled resource management	Enables resource management on the runtime.	

Remote SMF Runtime Launch Configuration dialog

This dialog enables you to configure a connection to a runtime running on a different machine, such as an embedded development board. Refer to the following table for a list of the Remote SMF Runtime Launch Configuration options, descriptions, and their default values.

Table 36. Remote SMF Runtime Launch Configuration dialog

Option	Description	Default
Host Name	Specifies the host on which the runtime is running.	localhost
Runtime Port #	Specifies the port on which the runtime is listening for the ide.	1457

SMF Bundle Server Launch dialog

This dialog enables you to configure options for an SMF Bundle Server when it is launched. In most cases, you do not need to modify the default values.

The Options tab enables you to configure where to create your server and the connection to create in the Bundle Server view. Refer to the following table for a list of the SMF Bundle Server Launch Configuration dialog Options tab options, descriptions, and default values, if any.

Table 37. SMF Bundle Server Launch Configuration dialog: Options tab

Option	Description	Default
Port	Specifies the port on which the bundle server listens.	8080
Name	Specifies the webapp name of the server.	SMF
ID	Specifies the user ID that connects to the server.	Admin
Password	Specifies the user password that connects to the server.	

The remaining tabs are standard Java launching screens. Refer to the following table for a list of the tabs and the options set by default on each of the tabs:

Table 38. SMF Bundle Server Launch Configuration dialog: Other options

Tab Name	Options set by default
Arguments	Server home variable and default working directory
JRE	Uses default
Classpath	User classes including seven required by SMF JAR files and the bootstrap image. The Virtual Machine determines the default bootstrap image value.
Common	Sets the SMF to be the default running perspective.

SMF Search dialog

This dialog enables you to search bundles based on their imports and exports.

Select a bundle from the list of bundle folders. After you select a bundle, the left panel displays a list of packages and services that are exported by the bundle you selected. By selecting packages or services, you can search for bundles that import the selected packages and services.

The right panel displays the bundles and services that are imported by the bundle you selected. By selecting items, you can search for bundles that export the selected packages and services.

Service Properties dialog

This dialog displays the service properties, such as the **objectClass**, **serviceid**, and **vendor**. You can access this dialog by double-clicking on a registered service in the SMF Runtime view.

Station Properties dialog

This dialog displays the station properties, such as the **Net Address**, **Service Name**, and **Service Port**. This dialog also displays session information, such as **Session User ID**, **Date Created**, and **Date Last Accessed**. You can access this dialog by double-clicking on a station in the SMF Bundle Servers view.

User Properties dialog

This dialog displays the properties, such as **User ID**, **First name**, **Last name**, and **Password**, for the user you selected. You can access this dialog by double-clicking on a user in the SMF Bundle Servers view. For more information about these values, refer to the SMF Bundle Server documentation.

SMF Bundle Developer Tools Ant Support

The Ant build tool is an Apache open source project written in Java. SMF Bundle Developer Tools provide Ant tasks and types that you can use to automate many tasks. You can use the Ant tasks to complete the following tasks:

- Submit a bundle to a bundle server from a bundle folder. You can use smfbd.submitJarBundle or smfbd.submitJxeBundle.
- Export bundles from a bundle server to a directory with smfbd.copyBundles.
- Import bundles to a bundle server from a directory or from another bundle server with smfbd.copyBundles.
- Enable, disable, or remove bundles from a bundle server with smfbd.bundleActions.
- Compute the Import-Package bundle manifest headers of bundle folders in the workspace with smfbd.computeImports.

Ant types

Refer to the following sections for descriptions of ant types that are specific to SMF Bundle Developer Tools.

bundleinfo

Description:

An Ant Type that identifies the bundle information that the Bundle Server uses to search for a set of bundles on the server. Refer to the following table for a list of the attributes, descriptions, and their default values, if any.

Table 39. bundleinfo attributes and descriptions

Attribute	Description	Default value
name	Represents the Bundle-Name. Use "*" to indicate any Bundle-Name.	"*"
type	Represents the type of Bundle. Valid values are “jar”, “jxe” or “*”. Use “*” to indicate any type.	"*"
endian	Represents the endian value. Valid values are “le”, or “be”. Use “*” to indicate any endian	"*"
version	Represents the Bundle-Version. Use “*” to indicate any Bundle-Version.	"*"

Examples:

The following example defines a set of all bundles:

```
<bundleinfo/>
```

The following example defines a set of all “jxe” bundles:

```
 <bundleinfo type="jxe"/>
```

The following example define a set of all “jxe” bundles for a little endian platform:

```
<bundleinfo type="jxe" endian="le" />
```

The following example defines a set that only includes the bundles with the name MyBundle that are the version 1.0.0:

```
<bundleinfo name="MyBundle" version="1.0.0"/>
```

bundleserver

Description:

An Ant Type that identifies a bundle server. Refer to the following table for a list of the bundleserver attributes, descriptions and whether or not the attribute is required.

Table 40. bundleserver attributes and descriptions

Attribute	Description	Required
host	Represents the host name of the bundle server.	Yes
port	Represents the port number that the bundle server runs on. In most cases, this value should be set to “8080”.	Yes
user	Represents a valid administrative user on the bundle server	Yes
password	Represents the user password on the bundle server.	Yes
webapp	Represents the name of the Web application of the bundle server. In most cases, this value should be set to “/smf”.	Yes

Examples:

The following example defines the localhost bundle server using the settings to connect to the default bundle server:

```
<bundleserver host="localhost" user="Admin"
      password="Admin" port="8080" webapp="/smf"/>
```

bundleserverlist

Description:

An Ant Type that identifies a bundle server and a list of bundleinfo elements. You can use this information to copy bundles from one server to another or to perform bundle actions on a list of bundles on a server.

Parameters specified as nested elements:

bundleserver

The bundleserver to which the list of bundleinfo elements belong. Only one nested bundleserver is permitted. You cannot associate more than one bundleserver with a bundleserverlist.

bundleinfo

A bundleinfo that is added to the list of bundleinfo elements for the bundleserverlist. You can have one or more bundleinfo elements in a bundleserverlist.

Examples:

The following example defines a set of all jxe bundles on the localhost bundleserver:

```
<bundleserverlist>
    <bundleserver host="localhost" user="Admin"
      password="Admin" port="8080" webapp="/smf"/>
    <bundleinfo type="jxe"/>
  </bundleserverlist>
```

filedir

Description:

An Ant Type that identifies a file system directory. Refer to the following table for a list of the filedir attributes, descriptions, and whether or not the attribute is required.

Table 41. filedir attributes and descriptions

Attribute	Description	Required
path	Represents the path of the file system directory	Yes

Examples:

The following example defines a filedir for the path c:\temp\mybundles:

```
<filedir path="c:/temp/mybundles"/>
```

Ant tasks

This section describes the Ant tasks provided with SMF Bundle Developer.

smfbd.bundleActions

Description: An Ant Task that performs an action on a set of bundles on a bundle server. A bundleserverlist defines the set of bundles. Refer to the following table for a list of the smfbd.bundleActions attributes, descriptions, and whether or not the attribute is required.

Table 42. smfbd.bundleActions attributes and descriptions

Attribute	Description	Required
type	Represents the type of action to perform on the set of bundles. Valid values are “remove”, “enable”, or “disable”	Yes

Parameters specified as nested elements:

bundleserverlist

Represents the set of bundles from a bundle server on which to perform the action.

Examples:

The following example removes all bundles on the localhost bundleserver:

```
<smfbd.bundleActions type="remove">

<bundleserverlist>
      <bundleserver host="localhost" user="Admin"
        password="Admin" port="8080" webapp="/smf"/>
      <bundleinfo/>
    </bundleserverlist>
  </smfbd.bundleActions>
```

The following example disables all jxe bundles on the localhost bundleserver:

```
  <smfbd.bundleActions type="disable">
    <bundleserverlist>
      <bundleserver host="localhost" user="Admin"
        password="Admin" port="8080" webapp="/smf"/>
      <bundleinfo type="jxe"/>
    </bundleserverlist>
  </smfbd.bundleActions>
```

The following example enables all jxe bundles for a little endian platform on the localhost bundleserver:

```
  <smfbd.bundleActions type="enable">
    <bundleserverlist>
      <bundleserver host="localhost" user="Admin"
        password="Admin" port="8080" webapp="/smf"/>
      <bundleinfo type="jxe" endian="le"/>
    </bundleserverlist>
  </smfbd.bundleActions>
```

smfbd.computeImports

Description:

An Ant Task that computes the Bundle Manifest header Import-Package for a set of Manifest files. Refer to the following table for a list of the smfbd.computeImports attributes, descriptions, and whether or not the attribute is required.

Table 43. smfbd.computeImports attributes and descriptions

Attribute	Description	Required
manifest	Represents the path relative to the workspace root of a single bundle manifest file.	This attribute is not required if the manifest and the project are not specified and the Import-Package header is computed for all open projects in the workspace. Do not specify the manifest attribute with the project attribute.
project	Represents the name of a project to compute all Manifest Import-Package headers on.	This value is not required if the manifest and the project are not specified and the Import-Package header is computed for all open projects in the workspace. Do not specify the project attribute with the manifest attribute.

Examples:

The following example computes the Import-Package header for all bundle manifest files that exist in open projects for a workspace:

```
<smfbd.computeImports/>
```

The following example computes the Import-Package header for all bundle manifest files in a single project named MyBundleProject:

```
<smfbd.computeImports project="MyBundleProject"/>
```

The following example computes the Import-Package header for a single manifest file:

```
<smfbd.computeImports
    manifest="/MyBundleProject/bundle1/META-INF/MANIFEST.MF"/>
```

smfbd.copyBundles

Description:

An Ant Task that copies a set of bundles to a bundleserver or filedir target. You can define the set of bundles with a fileset or bundleserverlist. Refer to the following table for a list of smfbd.copyBundles attributes, descriptions, and whether or not the attribute is required.

Table 44. smfbd.copyBundles attributes and descriptions

Attribute	Description	Required
replace	Indicates whether to replace the bundle if it already exists. Valid values are ″true″ and ″false″.	This value is not required and the default value is “false”.

Parameters specified as nested elements:

bundleserver

Represents the name of the bundleserver target that you want to copy bundles to. You can include multiple bundleserver target elements as nested elements of <smfbd.copyBundles>.

filedir

Represents the name of the file system target that you want to copy bundles to. You can include multiple filedir target elements as nested elements of <smfbd.copyBundles>.

fileset

Represents the set of files to copy to all the indicated targets. This is the standard ant <fileset> type. You can include multiple filesets as nested elements of <smfbd.copyBundles>.

bundleserverlist

Represents the set of bundles from a bundle server to copy to all the indicated targets.

Examples:

The following example copies all jxe bundles on the localhost bundleserver to the `c:/temp/mybundles` directory:

```
 <smfbd.copyBundles replace="true">
    <filedir path="c:/temp/mybundles"/>
    <bundleserverlist>
      <bundleserver host="localhost" user="Admin"
        password="Admin" port="8080" webapp="/smf"/>
      <bundleinfo type="jxe"/>
    </bundleserverlist>
  </smfbd.copyBundles>
```

The following example copies all bundles on the localhost bundleserver to a remote bundleserver at IP 192.168.0.101:

```
<smfbd.copyBundles replace="true">
    <bundleserver host="192.168.0.101" user="Admin"
      password="Admin" port="8080" webapp="/smf"/>
    <bundleserverlist>
      <bundleserver host="localhost" user="Admin"
        password="Admin" port="8080" webapp="/smf"/>
      <bundleinfo/>
    </bundleserverlist>
  </smfbd.copyBundles>
```

The following example copies all bundles from a directory to the localhost bundleserver:

```
  <smfbd.copyBundles replace="true">
    <bundleserver host="localhost" user="Admin"
      password="Admin" port="8080" webapp="/smf"/>
    <fileset dir=build/mybundles includes="**/*"/>
  </smfbd.copyBundles>
```

smfbd.submitJarBundle

Description: An Ant Task that submits a bundle from a bundle folder as a JAR to a bundleserver or filedir target. Refer to the following table for a list of the smfbd.submitJarBundle attributes, descriptions, and whether or not the attribute is required.

Table 45. smfbd.submitJarBundle attributes and descriptions

Attribute	Description	Required
bundlepath	Represents the path to the bundle folder in the workspace that contains the bundle to submit. The path must begin with a "/".	Yes
bundlefilename	Represents the name of the output file for the bundle. If the value does not have a .jar extension then that task adds the .jar extension.	This attribute is not required. The default value is the format specified in the SMF Submission preferences.
replace	Indicates whether to replace the bundle if it already exists. Valid values are ″true″ and ″false″.	This attribute is not required. The default value is "false".

Parameters specified as nested elements:

bundleserver

Represents the name of the bundleserver target to submit the bundle to. You can include multiple bundleserver target elements as nested elements of `<smfbd.submitJarBundle>`.

bundlefilename

Represents the name of the output file for the bundle. If the value does not have a .jar extension then that task adds the .jar extension. This attribute is not required. The default value is the format specified in the SMF Submission preferences.

filedir

Represents the file system target to submit the bundle to. You can include multiple filedir target elements as nested elements of `<smfbd.submitJarBundle >`.

Examples: The following example submits the pizza sample bundle to the bundle server running on localhost:

```
<smfbd.submitJarBundle bundlepath="/SMF Sample Bundles/pizza" replace="true">
  <bundleserver host="localhost" user="Admin"
    password="Admin" port="8080" webapp="/smf"/>
</smfbd.submitJarBundle>
```

The following example submits the pizza sample bundle to the `c:/temp/mybundles` directory with the file format specified in the SMF Submission preferences:

```
<smfbd.submitJarBundle bundlepath="/SMF Sample Bundles/pizza" replace="true">
  <filedir path="c:/temp/mybundles" bundlefilename="PizzaBundle"/>
</smfbd.submitJarBundle>
```

The following example submits the pizza sample bundle to the `c:/temp/mybundles` directory and specifies that the submitted file name should be MyPizzaSample.jar:

```
<smfbd.submitJarBundle bundlepath="/SMF Sample Bundles/pizza"
    bundlefilename="MyPizzaSample.jar"
    replace="true">
   <filedir path="c:/temp/mybundles"/>
 </smfbd.submitJxeBundle>
```

smfbd.submitJxeBundle

An Ant Task that submits a bundle from a bundle folder as a jxe to a bundleserver or filedir target. Refer to the following table for a list of the smfbd.submitJxeBundle attributes, descriptions, and whether or not the attribute is required.

Table 46. smfbd.submitJxeBundle attributes and descriptions

Attribute	Description	Required
bundlepath	Represents the path to the bundle folder in the workspace that contains the bundle to submit. The path must begin with a "/".	Yes
replace	Indicates whether to replace the bundle if it already exists. Valid values are ″true″ and ″false″.	This value is not required. The default value is "false".
addresslength	Indicates the address length. Valid values are "32" and "64".	This value is not required. The default value is "32".
endian	Indicates the endian value. Valid values are "le" or "be".	This value is not required. The default value is "le".

Table 46. smfbd.submitJxeBundle attributes and descriptions (continued)

Attribute	Description	Required
jxeoptions	Indicates the jxe options.	This value is not required. The default value is "-nofollow -nostartupclass"

Parameters specified as nested elements:

bundleserver

Represents the bundleserver target to submit the bundle to. You can included multiple bundleserver target elements as nested elements of `<smfbd.submitJxeBundle>`.

bundlefilename

Represents the name of the output file for the bundle. If the value does not have a .jxe extension then the ant task adds the .jxe extension. This value is not required. The default value is the format specified in the SMF Submission preferences.

filedir

Represents the file system target to submit the bundle to. You can include multiple filedir target elements as nested elements of `<smfbd.submitJxerBundle >`.

Examples: The following example submits the pizza sample bundle as a jxe with address length of 32 using little endian to the bundle server running on localhost:

```
<smfbd.submitJxeBundle bundlepath="/SMF Sample Bundles/pizza" replace="true">
   <bundleserver host="localhost" user="Admin"
     password="Admin" port="8080" webapp="/smf"/>
 </smfbd.submitJxeBundle>
```

The following example submits the pizza sample bundle as a jxe with address length of 32 using big endian to the `c:/temp/mybundles` directory with the file format specified in the SMF Submission preferences:

```
<smfbd.submitJxeBundle bundlepath="/SMF Sample Bundles/pizza"
  endian="be" addresslength="64" replace="true">
  <filedir path="c:/temp/mybundles"/>
</smfbd.submitJxeBundle>
```

The following example submits the pizza sample bundle as a jxe to the `c:/temp/mybundles` directory and specifies that the submitted file name should be MyPizzaSample.jxe:

```
<smfbd.submitJxeBundle bundlepath="/SMF Sample Bundles/pizza"
    bundlefilename="MyPizzaSample.jxe"
    replace="true">
   <filedir path="c:/temp/mybundles"/>
 </smfbd.submitJxeBundle>
```

Web application development

To develop a Web application, complete the following tasks:

1. Develop a Web application according to the Servlet 2.3 and JSP 1.2 specifications. Refer to "Developing a Web application" for more information.
2. Add OSGi-specific logic.

   Refer to "Accessing the OSGi environment from a Web application" for instructions to complete this optional step.
3. Translate the Web application's war file into a Web Application Bundle (WAB) file using the WAB tool.

   Refer to the Service Management Framework Runtime User's Guide for instructions to use the command line WAB tool. Refer to the Application Tools for Extension Services Developer's Guide for instructions to use the graphical user interface (GUI) WAB tool.
4. Deploy the WAB file as if it was a bundle.

   Refer to "Deploying the WAB file" on page 77 for more information.

Developing a Web application

The SMF Web container supports the Servlet 2.3 and JSP 1.2 specifications with minor restrictions. As such, you can use standard Web application development tools and methodologies. However, you must ensure that you develop against a Java Runtime Environment and library set that matches the runtime environment of the targeted SMF. For example, develop a Web application targeted for an SMF runtime that uses J2ME Foundation against the J2ME Foundation libraries.

Adding OSGi-specific logic

This task is required only if the Web application's logic needs to access the SMF framework. For example, add OSGi-specific logic if the Web application needs to access a service through the framework's registry.

Accessing the OSGi environment from a Web application

Refer to the following sections for information to access the OSGi environment from a Web application:

- "Custom bundle activator"
- "Accessing the bundle context" on page 76

Custom bundle activator: The WAB tool generates a default bundle activator if the war file does not contain one. However, you can provide a custom bundle activator if your Web application needs to interact with the framework in an application-specific way, or if you need to customize the default behavior of the Web application service.

Add the custom bundle activator class under the WEB-INF/classes tree. The `META-INF/MANIFEST.MF` manifest file must exist and must have the custom bundle activator class listed as the value of the Bundle-Activator property.

The custom bundle activator class can extend the `com.ibm.osg.webapp.AbstractWebApplicationActivator` abstract class. This abstract class performs registers the web application. Refer to the following steps to understand how the class registers a Web application with the Web container.

Note: The AbstractWebApplicationActivator handles all of these steps except the first step.

1. Instantiate a WebApplicationService object.

2. Register the WebApplicationService with the Framework service registry under the `WebApplicationService.WEBAPP_SERVICE` service interface.
3. Associate the properties defined in WEB-INF/wab.properties with the WebApplicationService when it registers. You must specify the following list of properties:

   Note: The WAB tool verifies that these properties are present.

   - WebApplicationService.WEBAPP_CONTEXT

     A string value that represents the context root for the Web application.
   - WebApplicationService.WEBAPP_VERSION

     A string value that represents the version of the OSGi web container specification that this Web application is compliant with. This value conforms to the Java 2 Package Versioning Specification. The current specification value is ″1.0.0″.

To satisfy classloader requirements, the class extending `AbstractWebApplicationActivator` must override the `getWebApplicationService()` method to instantiate and return the `WebApplicationService` object. (Step 1 completes this process.) Refer to the following example to understand how the `WebApplicationService` can be instantiated by extending the `AbstractWebApplicationService` class:

```
WebApplicationService myService =
       new AbstractWebApplicationService(context,null) {};
```

The classloader will not behave properly if the WebApplicationService is not defined in the bundle activator or some other class that is local to the Web application.

Refer to the following bundle activator example that was created by extending the AbstractWebApplicationActivator:

```
import java.io.IOException;
import org.osgi.framework.BundleContext;
import com.ibm.osg.webapp.*;
/**
* Minimum bundle activator for web applications.
*/
public class DefaultWebApplicationActivator
       extends AbstractWebApplicationActivator {

       private WebApplicationService webAppService;

       public WebApplicationService getWebApplicationService() {
              if (webAppService == null) {
                     webAppService =
                            new AbstractWebApplicationService(context, null) {};
              }
              return webAppService;
       }

}
```

Accessing the bundle context: A bundle accesses the OSGi framework’s services through the bundle’s BundleContext. To enable a Web application to access the Web Application Bundle’s BundleContext develop a custom BundleActivator for the Web Application Bundle. The custom BundleActivator stores the BundleContext in a location that is accessible by the Web application’s servlet code. Refer to the following example of a BundleActivator that stores the BundleContext in a static field:

```
import org.osgi.framework.BundleContext;
import com.ibm.osg.webapp.AbstractWebApplicationActivator;
import com.ibm.osg.webapp.AbstractWebApplicationService;
import com.ibm.osg.webapp.WebApplicationService;
public class CustomWebAppBundleActivator extends
       AbstractWebApplicationActivator {

private WebApplicationService webAppService;
public WebApplicationService getWebApplicationService() {
              if (webAppService == null) {
                  webAppService =
                        new AbstractWebApplicationService(context, null) {};
              }
              return webAppService;
       }

       public void start(BundleContext context) throws Exception {
              WebAppBundleContext.context = context;
              super.start(context);
       }

}

public class WebAppBundleContext {
       public static BundleContext context;
}
```

Deploying the WAB file

Because the WAB file is an OSGi bundle, you can submit it to an SMF Bundle Server and deploy to the SMF runtime in the same manner as other bundles.

Appendix A. Service Management Framework Architecture

SMF bundle requirements

To fully manage bundles, Service Management Framework bundle servers require bundles to provide the following information:

- The bundle must specify the specification version for every package that the bundle exports.
- The bundle must include an `IVEATTRS.XML` file.

The bundle can also include an `IVERES.XML` file.

Specification version

Bundles must express a specification version for all of the packages that the bundle exports. An algorithm uses the specification version to find the set of prerequisite bundles for the bundle.

Without the specification-version information, the bundle server cannot verify whether the package exported by one bundle is adequate to support another. In addition, the bundle server is also unable to verify whether or not bundles that reside on the Runtime server need to be updated.

IVEATTRS

The bundle server uses the IVEATTRS.XML file to select the best bundle for a particular platform. For example, the bundle server uses the information in the IVEATTRS.XML file to determine that the bundle server should not provide a JXE bundle for a little-endian architecture to a runtime server that runs on a big-endian processor.

The IVEATTRS.XML file resides in the META-INF directory. The file name must be in all upper-case letters. Refer to the following example for the format of the IVEATTRS.XML file.

```
<?xml version="1.0" standalone="yes"?>
<IVEAttrs>
   <Processor/>
   <Endian>le</Endian>
   <AddressLength>32</AddressLength>
   <OS/>
   <OSVersion/>
   <VM>J9v14</VM>
   <ImplType>JCL_Gateway_1_3</ImplType>
   <Language>en</Language>
   <Country>us</Country>
   <Replaces/>
</IVEAttrs>
```

```
DTD for IVEATTRS.XML Files
<!-- Name: -//com-ibm-ive-eccomm//DTD IVEATTRS//EN -->
<!-- Version: 1.0 4/12/2000 -->
<!ELEMENT IVEAttrs (Processor,OS,OSVersion,VM,ImplType,
    Language,Country,Replaces,RAMRequirements)>
<!-- Processor identifies the target's architecture.
Where possible draw from OSD specification-->
<!ELEMENT Processor (#PCDATA)>
<!-- OS identifies the target's operating system.
Where possible draw from OSD specification-->
<!ELEMENT OS (#PCDATA)>
<!-- OSVersion identifies the version of the target's operating system.-->
<!ELEMENT Endian (#PCDATA)>
<!-- Endian identifies whether the client is operating as a big
or little endian processor-->
<!ELEMENT AddressLength (#PCDATA)>
<!-- AddressLength specifies the length of the target
processor architecture's address-->
<!ELEMENT OSVersion (#PCDATA)>
<!-- VM identifies the JVM running on the target-->
<!-- ImplType is a code which identifies the base Java class
    libraries available on the client-->
<!ELEMENT ImplType (#PCDATA)>
<!-- Language specifies the target's ISO 639 language code-->
<!ELEMENT Language (#PCDATA)>
<!-- Specifies the target's A2 ISO 3166 country code-->
<!ELEMENT Country (#PCDATA)>
<!-- Interpretation of the Replaces tag is reserved-->
<!ELEMENT Replaces (#PCDATA)>
```

Implementation type

The ImplType tag identifies the base class library support required by the bundle. Because the base configuration can vary depending on the type of target, there is no predefined list of acceptable values for this attribute. If you define a new custom base configuration for a new platform, you must assign a new implementation type code to the new custom base configuration.

Compatible implementation types

Because some base configurations are compatible with others, the bundle server must keep a list of which ImplTypes are compatible with others. For example, the bundle server might use the list to determine whether a bundle targeted at `jclGwp` is appropriate for a runtime that runs `jclMax`. However, if the bundle server locates an exact match for ImplType, the bundle server chooses the exact match over a different, but compatible ImplType.

Because the bundle server is aware of compatible implementation types you can make bundles available on multiple target types without creating multiple bundles whose only difference is their ImplType.

IVERES

You can use the IVERES.XML file to record the impact you expect the bundle to have on the bundle's host resources. The bundle server uses the information in this file to ensure that runtimes have sufficient resources to run the bundles provided to them. The runtime framework can use the IVERES.XML file to enforce the resource constraints.

The IVERES.XML file resides in the META-INF directory. The filename must be all uppercase.

You can modify the format of the IVERES.XML to fit specify and manage resource requirements for resources that have not been identified. Refer to the following example for the format of a typical IVERES.XML file:

```
<?xml version="1.0" standalone="yes"?>
<IVERes>
   <Resource>
      <Name>RAM</Name>
         <Requirement>19507</Requirement>
   </Resource>
   <Resource>
      <Name>Threads</Name>
         <Requirement>6</Requirement>
   </Resource>
</IVERes>
```

```
DTD for IVERES.XML Files
The DTD for the IVERES.XML File is:
<!-- Name: -//com-ibm-ive-eccomm//DTD IVERES//EN -->
<!-- Version: 1.0 4/5/2000 -->
<!ELEMENT IVERes (Resource*)>
<!ELEMENT Resource (Name,(Requirement|Available))>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Requirement (#PCDATA)>
<!ELEMENT Available (#PCDATA)>
```

Refer to "Reserved resource names" for a list of the reserved resource names and their descriptions.

Reserved resource names

Refer to the following table for a list of the reserved resource names and their descriptions:

Table 47. Reserved resource names

Resource name	Description
NewSpaceSize	The number of bytes to be allocated in each new space region of the bundle's memory space

Table 47. Reserved resource names (continued)

Resource name	Description
OldSpaceSize	The number of bytes to be allocated for tenured objects in the bundle's memory space
Files	The total number of files a bundle can create in its data directory.
Quota	The number of bytes of storage within the local filesystem required by the bundle
Sockets	The maximum number of socket connections the bundle may have open at any one time
Threads	The number of threads the bundle can create at any one time

Appendix B. Platform Profile and Extension Points

Creating a Platform Profile

An XML-based platform profile descriptor file describes the Platform Profile. You can define your own Platform Profile by creating a platform profile descriptor file and adding it to the tools through the platform profile extension point. Refer to the `com.ibm.smf.tools.profiles` plug-in for platform profile examples. A schema for platform profiles is also available in the `com.ibm.smf.tools.project` plug-in under `schema/PSPPlatformProfile.xsd`.

The following information assumes that you have knowledge of the OSGi Alliance concepts, such as bundles and services. Refer to the Service Management Framework User’s Guide for more information about these concepts.

You can externalize the descriptive text in the profile for internationalization. The Platform Profile uses the same mechanism for externalizing strings as WebSphere Studio uses for plugin.xml. The tools consider words within text strings that begin with the percent character (%) to be keywords for referencing corresponding text from a properties file. The default properties file basename is the same as the Platform Profile basename. You can explicitly specify the name of the properties file through the `PlatformProfile` elements’ `PropertyFile` attribute. Properties files for specific languages follow the Java resource bundle naming conventions.

Refer to the “Platform Profile Document Type Definition” for the platform profile format. Refer to “Platform Profile elements and attributes” on page 84 for descriptions of the elements and attributes.

Platform Profile Document Type Definition

The following Document Type Definition (DTD) defines the platform profile format.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT PlatformProfile
(JCL,ApplicationService*,RequiredApplicationService*,RequiredRuntimeService*)>
<!ATTLIST PlatformProfile
  ID            CDATA #REQUIRED
  Name          CDATA #REQUIRED
  Provider      CDATA #IMPLIED
  Description   CDATA #IMPLIED
  PropertyFile  CDATA #IMPLIED
>

<!ELEMENT JCL (#PCDATA)>
<!ATTLIST JCL
  Name          CDATA #REQUIRED
  Type          (ECLIPSE_JRE | J9_WCE | J9_WME) #REQUIRED
>

<!ELEMENT ApplicationService (Bundle*,ImportService*, DynamicImport-Package*,
      ClasspathLibrary*, BootLibrary*, NativeLibrary*)>
<!ATTLIST ApplicationService
  Name          CDATA #REQUIRED
  ID            CDATA #IMPLIED
  Description   CDATA #IMPLIED
  Version       CDATA #IMPLIED
>
```

```
<!ELEMENT RequiredApplicationService EMPTY>
<!ATTLIST RequiredApplicationService
  ID            CDATA #REQUIRED
>

<!ELEMENT RequiredRuntimeService EMPTY>
<!ATTLIST RequiredRuntimeService
  ID            CDATA #REQUIRED
>

<!ELEMENT Bundle (#PCDATA)>
<!ATTLIST Bundle
  ReferenceType (PLUGIN_RELATIVE | CLASSPATH_VARIABLE | ABSOLUTE) #REQUIRED
>

<!ELEMENT ImportService (#PCDATA)>

<!ELEMENT DynamicImport-Package (#PCDATA)>

<!ELEMENT ClasspathLibrary (#PCDATA)>

<!ELEMENT BootLibrary (#PCDATA)>
<!ATTLIST BootLibrary
  Position (PREPEND | APPEND) #IMPLIED
>

<!ELEMENT NativeLibrary (#PCDATA)>
<!ATTLIST NativeLibrary
  OS ( Windows 2000 | Windows2000 | Win2000 | Windows XP | WindowsXP
 | WinXP | Linux ) #REQUIRED
  Processor ( x86 ) #REQUIRED
>
```

Platform Profile elements and attributes

In the following descriptions, *development time* refers to the development environment of an Extension Services project and *runtime* refers to the Extension Services server runtime environment.

PlatformProfile element

The `PlatformProfile` element contains the definition of a Platform Profile. Refer to the following table for the elements in the PlatformProfile element and their descriptions.

Table 48. Elements in the PlatformProfile element

Element	Description
JCL	Defines the Java Class Library that the platform uses.
ApplicationService	Defines each Application Service provided by the platform.
RequiredApplicationService	Defines the Application Services that are required at development time and runtime.
RequiredRuntimeService	Defines the Application Services that are required at runtime, but are not required at development time.

Refer to the following table for the attributes in the PlatformProfile element and their descriptions.

Table 49. Attributes in the PlatformProfile element

Attributes	Description
ID	Specifies a unique identification, in the format of a Uniform Resource Locator (URL), for the platform profile.
Name	Specifies a descriptive name for the Platform Profile.
Provider	Specifies the provider name.
Description	Provides a text description of the Platform Profile.
PropertyFile	Specifies a file name of the properties file to be used for resource strings. This name should not include the `.properties` extension. If this attribute is not included, the default value is the base file name of the platform profile.

JCL element

The `JCL` element defines the Java Class Library (JCL). The Application Tools will set this JCL in the classpath of an Extension Services project that uses this platform. Extension Services also uses the JCL as an Extension Services server that is configured to use this platform. The Application Tools identify which JCL to use based on the value of this element and the value of the Type attribute. Refer to the following table for the attributes in the JCL element and their descriptions.

Table 50. Attributes in the element

Attributes	Description
Name	Specifies a descriptive name for the Java Class Library.
Type	Specifies the type of the Java Class Library. Application Tools uses this attribute and the value of the JCL element to determine which Java Class Library to use. Refer to the following list for valid values: • `ECLIPSE_JRE` indicates that the server uses the default Java Runtime Environment (JRE) that was selected in the WebSphere Studio Java Properties. If you specify this value, the value of the JCL element is ignored. • `J9_WCE` indicates that the server uses the IBM J9 Java Virtual Machine (JVM) and the Java Class Library indicated by the value of the JCL element. The value of the JCL element must be one of the IBM s • Custom Environment Java Class Libraries. • J9_WME indicates that the server uses the IBM J9 Java Virtual Machine (JVM) and the Java Class Library indicated by the value of the JCL element. The value of the JCL element must be one of the WebSphere Everyplace Micro Environment Java Class Libraries.

ApplicationService element

The `ApplicationService` element defines an Application Service provided by the platform. `Bundle` elements define the set of bundles that are associated with the Application Service. `ImportService` elements define the service import dependencies for users of this Application Service. `DynamicImport-Package` elements define the dynamic import package requirements for users of the Application Service. `ClasspathLibrary` elements define classpath libraries used by the Application Services. `BootLibrary` elements define boot classpath libraries used by the Application Service. `NativeLibrary` elements define the native libraries used by the Application Service.

Refer to the following table for the attributes in the ApplicationService element and their descriptions.

Table 51. Attributes in the ApplicationService element

Attributes	**Description**
Name	Specifies a descriptive name for the Application Service.
ID	Specifies a unique identification used to refer to this element.
Version	Specifies the version of the Application Service. The version format is `major.minor.service`, where `major`, `minor`, and `service` are integers that represent the version of the Application Service.
Description	Provides a text description of the Application Service.

Bundle element

The `Bundle` element value references an the OSGi Alliance bundle. The `ReferenceType` attribute of the Bundle element defines the form of the OSGi bundle reference. Refer to the following table for a list of the `ReferenceType` attribute values and their descriptions.

Table 52. ReferenceType attribute form values and descriptions

Value	**Description**
PLUGIN_RELATIVE	Specifies a resource reference that is relative to a plug-in. The first part of the reference path must be the plug-in ID. The remainder of the path represents the location of the resource within the plug-in. For example, a reference to `myLib.jar` under the `lib` directory, within the `com.mycompany.myplugin` plug-in is `com.mycompany.myplugin/lib/myLib.jar`. **Note:** This value is not supported for the `ReferenceType` attribute for `Bundle` elements. It is supported for the `BootLibrary` and `NativeLibrary` elements.
CLASSPATH_VARIABLE	Specifies a resource reference that is relative to the value of a classpath variable that is defined in the WebSphere Studio environment. For example, if you defined the classpath variable `MY_BUNDLES` as `C:/SMF/bundles`, then the reference `MY_BUNDLES/myBundle.jar` refers to the file `C:/SMF/bundles/myBundle.jar`.
ABSOLUTE	Specifies an absolute file reference to the resource. This is an operating system specific reference, such as `C:/SMF/bundles/myBundle.jar`.

ImportService element

The `ImportService` element specifies a service that a project using the associated Application Service must import. The value of the ImportService element is the service interface class. When a project selects the Application Service associated with this element, the Application Tools add this service to the project Manifest file under the Import-Service header.

DynamicImport-Package element
The DynamicImport-Package element specifies a dynamic package specification that a project using the Application Service associated with this element must import. When a project selects the Application Service associated with this element, the Application Tools add this dynamic package to the project Manifest file under the DynamicImport-Package header.

ClasspathLibrary element
The ClasspathLibrary element references a library jar file that is needed on the development time and runtime classpath by the Application Service. The ReferenceType attribute of the ClasspathLibrary defines the form of the library reference. Refer to Table 52 on page 86 for a list of the ReferenceType attribute values and their descriptions.

BootLibrary element
The BootLibrary element references a library jar file that is needed on the runtime boot classpath by the Application Service. The ReferenceType attribute of the BootLibrary defines the form of the library reference. Refer to Table 52 on page 86 for a list of the ReferenceType attribute values and their descriptions. The Position attribute of the BootLibrary defines where the library appears in the boot classpath. Refer to the following table for a list of the Position attribute values and their descriptions.

Table 53. Position attribute values and descriptions

Value	Description
PREPEND	Specifies that the boot library should be prepended to the boot classpath.
APPEND	Specifies that the boot library should be appended to the boot classpath. This is the default value if the Position attribute is not specified.

NativeLibrary element
The NativeLibrary element references a file or directory that contains native libraries used by the Application Service. The ReferenceType attribute of the NativeLibrary defines the form of the library reference. Refer to Table 52 on page 86 for a list of the ReferenceType attribute values and their descriptions. The OS attribute of the NativeLibrary defines what operating system this location is appropriate for. The Processor attribute of the NativeLibrary defines what processor type this location is appropriate for.

RequiredApplicationService element
The RequiredApplicationService element specifies an Application Service that must be present at development time by a project that uses this platform and that must be present at runtime on an Extension Services server that runs this platform. The Application Tools automatically include this Application Service in the set of services that a project uses when the project selects the platform profile. The Application Tools automatically install and start this service on local Extension Services servers that are configured to use this platform.

You can reference the Application Service with the `ID` attribute. The value you specify for the `ID` attribute must match the `ApplicationService` element's `ID` attribute in the platform profile.

RequiredRuntimeService element

The `RequiredRuntimeService` element specifies an Application Service that must be present at runtime on an Extension Services server that runs this platform. The Application Tools automatically install and start this service on local Extension Services servers that are configured to use this platform.

You can reference the Application Service with the `ID` attribute. The value you specify for the `ID` attribute must match the `ApplicationService` element's `ID` attribute in the platform profile.

Extension points

Application Tools for Extension Services includes the "com.ibm.smf.tools.project.platformprofilerepository" extension point.

com.ibm.smf.tools.project.platformprofilerepository

Platform Profile

Identifier:

com.ibm.smf.tools.project.platformprofilerespository

Description:

This extension point allows a plug-in to contribute platform profiles.

Configuration markup:

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

The value of RepositoryLocation is a path to a directory, which is relative to the directory where the plug-in resides, that contains the Platform Profile descriptor files.

Examples:

The following example contributes Platform Profiles that reside in the `resources/profiles` directory for the plug-in.

```
<extension point="com.ibm.smf.tools.project.platformprofilerepository">
  <RepositoryLocation>
     resources/profiles
  </RepositoryLocation>
</extension>
```

Appendix C. Notices

This information was developed for products and services offered in the U.S.A. IBM might not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM might have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

Trademarks