



Introduction to the Service Management Framework™


The IBM Service Management Framework™ (SMF) is an implementation of the OSGi Service Platform specification, and provides a production-ready software management framework for network-delivered applications. IBM designed SMF to meet the needs of Internet-ready device manufacturers and service providers (for example, cable companies, telecommunication providers, Internet Service Providers, and power utilities.)

SMF 3.5 (released in April, 2003) implements OSGi Service Platform Release 3 (SPR3). It is available from <http://www.ibm.com/software/wireless/smf> as a stand-alone toolkit and as a component of the SMF Bundle Developer 5.5 plug-in for WebSphere Studio Device Developer (WSDD) 5.5. The previous release, SMF 3.1, was an implementation of OSGi SPR2, and shipped as a stand-alone toolkit as well as a plug-in to WSDD 5.0. (The WSDD plug-ins are also available through the Update Manager in the Install/Update perspective.)

OSGi SPR3 defines a number of specifications which define the core functions of the platform and an application lifecycle, and provide a services registry, package and version management, and remote management ability. These specifications are then implemented by OSGi Service Platform implementations such as SMF. The original target platforms for an OSGi implementation were service gateways, vehicle telematics and in-cabin information & entertainment systems, telephone switches, industrial controllers, home set-top boxes, and similar devices. However, current OSGi implementations (including SMF) are able to run on many different platforms.

The OSGi Alliance (<http://www.osgi.org/>) is an independent worldwide organization, founded in March, 1999, by IBM, Sun, Ericsson and Oracle. It is an open forum, which defines the OSGi Service Platform, and promotes its use. The platform itself is an open service platform for the delivery and management of multiple applications and services to all types of networked devices in home, vehicle, mobile and other environments. Specifically, the OSGi Service Platform is a Java framework for developing remotely deployed service applications. The framework is designed to provide reliability, large-scale distribution, support for a wide range of devices and collaboration between modules. The platform specifications are available at their website.

An OSGi bundle implementation provides services, and is the equivalent of an application. Services are defined by specifications (a Java interface) and implemented by one or more implementations (Java classes that implement the interface.) Because there is a strict separation between specification and implementation, a service could have multiple implementations for different environments. For example, a logging service could write to a file in one implementation, to an expanding memory buffer in another, and to a circular buffer in a third. All of these implementations would have the same specification (by implementing the same Java interface); so that a consumer of the service need not know which specific implementation was providing the service, or how the service was implemented.



The OSGi platform was designed to provide reliability, since it will typically be used in large-scale deployments. The platform is portable to attract third-party developers to create services for it. It is dynamic, allowing configurations to adapt to user and operator needs over time. It is secure, protecting service providers from each other. It is scalable, since the size of the implementations may vary over time, and since members have very different configurations for their deployment of OSGi frameworks.

The OSGi architecture is based on Java, since its portable byte code format produces deliverables that are independent of operating system or processor. Java is a relatively safe environment, since security is integral to the language, and it provides integrity by not allowing dangerous functions. Finally, it is a relatively mature platform, with an active developer community and broad industry support.

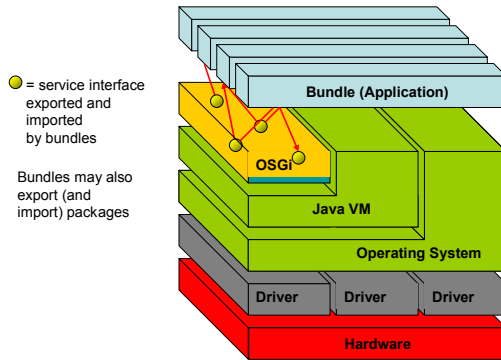
Using Java does have some considerations. Its integrity is cooperative, not like an operating system, which may have guarantees. Its security implementation is usually performance intensive, which adds performance cost. It requires a runtime (the Java Virtual Machine) which adds to the deployment size. It also lacks resource management in the core language. (However, IBM provides the jclRM class library within the WebSphere Custom Environment that does provide resource management.) Java's strengths outweigh its weaknesses in this case.

SMF is IBM's implementation of the OSGi specification. It is a componentized implementation, optimized for embedded use, to enable viable deployment on resource constrained devices. (For example, we have run the framework and applications on Linux-based PDAs and PocketPC devices.) It is integrated with IBM WebSphere Everyplace family for deployment, and the IBM WebSphere Studio for a development environment.

SMF 3.5 runs on any environment that supports Java 2 (for example, WSDD 5.5 and above.)

SMF allows applications to share a single Java Virtual Machine, and manages the applications it controls. Since all applications run within the framework, SMF can install, start, stop, update and uninstall applications without affecting other applications executing within the framework. This makes it an excellent deployment platform in an environment where recycling the entire JVM just to reload an application should be avoided. This was one of the major design points, since the original deployment platforms for the OSGi framework (and therefore, SMF) were service gateways, vehicle telematics and in-cabin information & entertainment systems, telephone switches, industrial controllers, home set-top boxes, and similar devices which require uninterrupted processing.

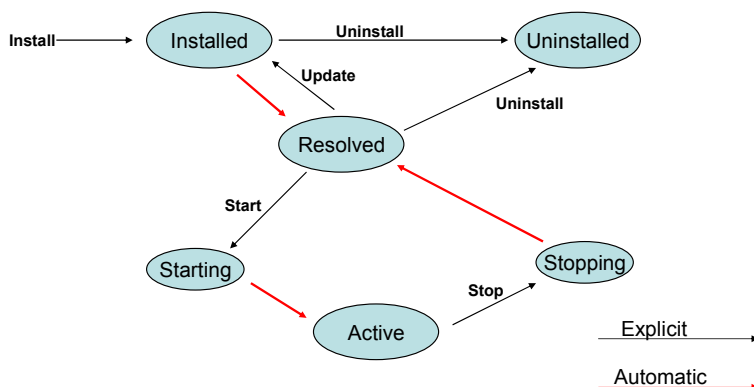
OSGi architecture



OSGi (and SMF) applications are called “bundles.” A bundle is a JAR file (or a JXE file for the WebSphere Micro Environment and WebSphere Custom Environment) containing the resources to implement zero or more services, a manifest file with bundle information (dependencies on other resources, identifiers, etc.), and an optional BundleActivator class. A bundle can also act as a library, and only export Java packages. The SMF platform can install, update, and uninstall bundles dynamically, while other bundles are active. A service is specified in a Java interface and may be implemented by multiple bundles. The Framework itself is represented as the System bundle. Code within bundles can execute searches to find services registered by other bundles.

The bundle lifecycle contains six states: installed, resolved, starting, active, stopping and uninstalled.

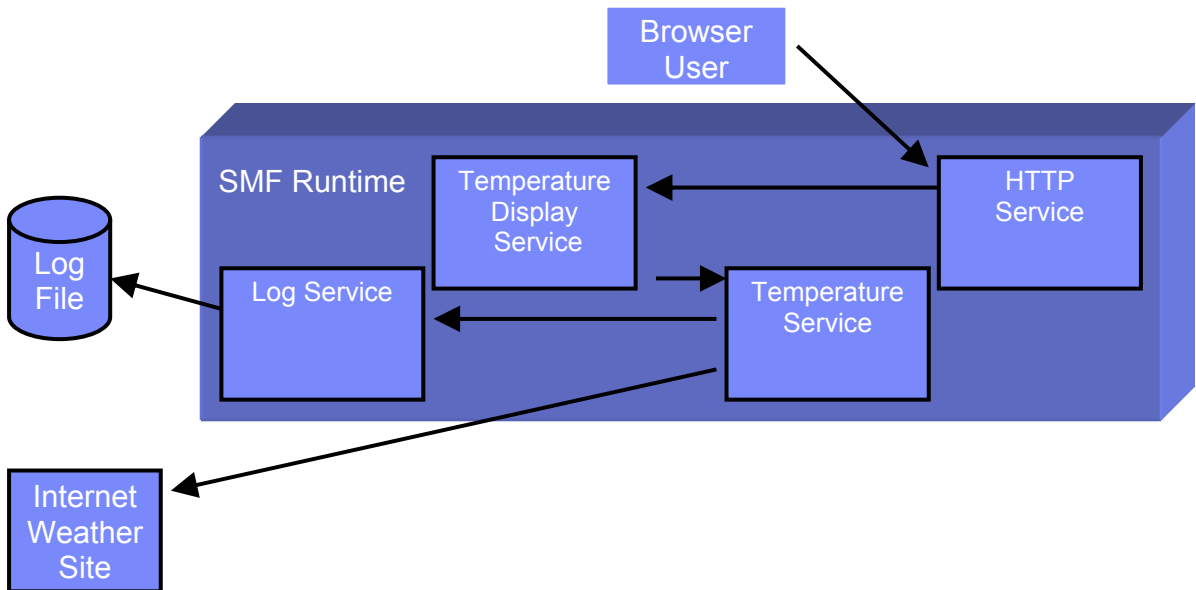
Bundle Lifecycle




When a bundle is installed into the runtime, it will enter the installed state. Once all of its prerequisites (other bundles or packages) are available, it will automatically change to the resolved state. At this point, it may be started, updated, or uninstalled dynamically and asynchronously to any other bundles within the JVM.

When a resolved bundle is started, its BundleActivator class' start() method is invoked. After it returns successfully, the bundle is active. An active bundle can be stopped at any time (and its BundleActivator's stop() method invoked), and will return to the resolved state. An update to a bundle will return it to the installed state (after the update completes), and at that point, the bundle could also automatically move into the resolved state, if all prerequisites are available. When a resolved bundle is uninstalled, it changes to the uninstalled state, and can be garbage collected at any point after that.

Each bundle contains a manifest file (MANIFEST.MF), located in the META-INF directory inside the bundle JAR file. This file contains data the framework needs to correctly install and activate it. The OSGi specification defines a set of headers that must be used to define this data.



Here is a simple example of an OSGi Framework "application", which contains four bundles running within the SMF runtime. Imagine a small device which will connect to the Internet and retrieve weather information from a website. This information will be stored on the device for later retrieval. In this implementation, we would have a number of services – a weather service that connects to the website and retrieves the information, a logging service that files the information away for later retrieval, and a display service that would provide an end user interface to the information. The display service uses the HTTP service, which is a component of the SMF runtime itself.



Within the OSGi framework, a "bundle" is the equivalent of an application. So, each of these services would be implemented as a bundle, and these bundles would be installed within the framework. A bundle can contain services and implementations of the services. Bundles register the services they have available with the Framework, so that other bundles can locate them.

A service exposes its functionality by publishing a Java interface – a listing of methods that will be available. An implementation of a service implements those methods, so that actual work can be performed. In our weather example, the weather service would not expose any methods for other bundles to use, since it only retrieves information and stores it. The logging service would expose methods to write information to the log, and to read information from it. The display service would expose a method to display information, probably by registering a servlet with the HTTP Service (specified by the OSGi Framework and included with SMF) which can be invoked from a browser. The weather and display services would both use the exposed methods of the logging service in order to store and fetch the information.

One strength of the OSGi Framework's design is that none of the service consumers need know how the service they are using is actually implemented. For example, the logging service may write to a disk file, if disk storage is available. In smaller device implementations, it may store data to Flash memory, or just keep it in RAM. However, the other services don't care, as they just call the exposed methods (discovered via the interface), and receive the proper information. In fact, there could be multiple implementations of a service on a device, and either the framework or the consumer of the service could choose which implementation to use in a given situation. The sample application also exemplifies the benefits of using the OSGi Framework for porting applications to different platforms. If this application needs to be ported to another small device which requires a different type of rendering, only the Temperature Display Service may need to be rewritten – the core application logic in the Temperature Service will not change.

For developing bundles, IBM provides the SMF Bundle Developer, a WebSphere Studio Workbench feature that runs within WSDD. WebSphere Studio Workbench is the IBM-supported version of the Eclipse platform (see <http://www.eclipse.org> for information and downloads), an open-source integrated development environment. The core Eclipse platform is extended by using plug-ins, code which is specifically written to 'plug into' the base platform. The Bundle Developer is such a plug-in. The Workbench is an IBM offering (not a product) and is the core technology behind the rest of the WebSphere Studio family of products.

The SMF Bundle Developer consists of the SMF Runtime, Bundle Server, and Bundle Developer graphical user interface for creating and debugging bundles. Its basic features will run on standard JREs and can be used in a WebSphere Studio Application Developer or vanilla Eclipse environment. However, some features are dependent on the WebSphere Micro Environment (or WebSphere Custom Environment), and therefore WSDD, such as



launching a flash store-based runtime, and resource management. WSDD is the only environment that IBM supports for running the SMF Bundle Developer, however WSDD will also act as a plug-in for WebSphere Studio Application Developer (WSAD) and WebSphere Studio Site Developer, so it will run under those environments, as well.

As a WSDD plug-in, the SMF Bundle Developer requires WSDD 5.0 (or above) and either the Custom Gateway Plus, Custom Resource Managed Gateway Plus, or Custom Max library, if using the WebSphere Custom Environment, or the Foundation configuration, if using the WebSphere Micro Environment.

The Bundle Developer plug-in provides an SMF Perspective within WSDD, which consists of a graphical interface to identify bundle packages, imports, and exports. It has an editor for OSGi Framework manifest files, and allows developers to tag a bundle with device characteristics to differentiate target devices. It also allows developers to connect to an SMF Bundle Server, view the server's contents, and submit bundles to the server. After bundles are on the server, it provides an interface to launch an SMF runtime, connect to local or remote runtimes and install bundles into the runtime for testing.

The SMF Runtime that ships with WSDD integrates with the SMF Bundle Developer integrated development environment, and provides pluggable platform implementations, a file-based one, and a Flash-based one (based on a Flash memory simulator.)

SMF has a command line interface from which bundles can be managed, and it also contains a bundle called SMFAdmin which uses a servlet to system administrators to perform many of the framework's administrative tasks such as installing new bundles from an HTTP server through a browser interface.

The SMF Bundle Server maintains a bundle catalog, and can be shared by multiple developers. It interacts with a management agent for the SMF runtime, and can manage an unlaunched SMF platform. It provides bundle "snapshots" (to store the runtime's state for recovery or reset), and dependency checking for loading bundles.

Snapshots are a way to store the current state of the runtime for later use. A typical use for snapshots is for developers to load all of the bundles needed on a particular target runtime and then to save the snapshot so that they can test different configurations and still be able to return to the previous state.

The SMF Bundle Server is included as both a stand-alone server running under the WebSphere Micro Environment and as a web application that can be installed under Apache's Tomcat servlet engine. It is designed for use during development and testing of your bundles, rather than for production usage.

You can configure the Bundle Server to store its repository either in the file system (the default) or in a SQL database.



The file-based implementation does not support groups (sets of users, stations or bundles) and permissions. It is generally intended for an individual or small development team environment. The bundle threshold is somewhat subjective, but it is expected that exceeding somewhere between 100 and 150 bundles in the repository will likely degrade performance enough to warrant moving to an SQL-based configuration.

The SQL-based configuration can be used by individuals or small teams but scales better in larger development teams than the file-based configuration. Choose an SQL configuration when the server is to be used by more than a few developers, or when the number of bundles in the repository grows to a level that the performance of a file-based configuration is no longer adequate.

The Bundle Developer uses the Safe Bundle Install Protocol to install bundles into the runtime. This protocol has the runtime provide the SMF bundle server with its configuration data and a list of currently-installed bundles. The bundle server then determines the correct version of a bundle, resolves the bundle before it is downloaded by determining whether all the required packages and services are available in the runtime, and provides a list of prerequisite bundles needed by the runtime.

The Safe Bundle Install Protocol provides two important functions: it minimizes the amount of traffic between the server and runtime, and it ensures to the greatest extent possible that bundles that are downloaded will have the necessary prerequisite bundles to execute.

The SMF Bundle Server is designed for development and testing environments, and not for production use. In some production environments, the application's bundles generally would be pre-loaded on the device, to avoid transfer time. In production environments where the configuration of the target device is very dynamic, such as in the Telematics or the Smart Home industry, the SMF runtime can be managed remotely using the OSGi Framework support contained in IBM's WebSphere Everyplace product portfolio. These software offerings (WebSphere Everyplace Service Provider Offering, and WebSphere Everyplace Access) allow for OSGi Framework bundle management, software inventory management and software revision management in production environments with high scalability requirements, based on open standards like SyncML DM from the Open Mobile Alliance (<http://www.openmobilealliance.org>).

The Service Management Framework is a flexible development and deployment environment for multiple platforms, with tightly integrated development support within WebSphere Studio Device Developer.