OSGi Service Platform for the Development of the Mobile and Embedded Applications

Pavlin Dobrev
Research and Development Manager
Agenda

- OSGi Overview
- OSGi R4 Extensions
- OSGi Products
- Resources
- Examples
Driving Market Challenges

**Device evolution**
- Extended connectivity
- Shorter life cycles
- Higher complexity
- Dynamic deployment

**Development**
- Shorter development cycles
- Scalable platforms
- Aftermarket device access
- Feature complexity

**Service Provider**
- New services and apps
- Content provider
- User portals
- CRM, Billing

**Infrastructure**
- Telecommunication
- Administration
- Deployment
OSGi Overview
Mission
“Our mission is to specify, create, advance, and promote 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”.

„The OSGi Alliance serves as the focal point for a collaborative ecosystem of service provider, technology, industrial, consumer and automotive electronics communities“.

Source: OSGi Homepage
OSGi – The Mission

Device evolution
- More resources available
- Extended connectivity
- Shorter life cycles
- Higher complexity

Service Provider
- New services
- Content provider
- User portals
- CRM, Billing

Development
- Shorter development cycles
- Scalable platforms
- Aftermarket device access
- Feature complexity

Infrastructure
- Telecommunication
- Administration
- Deployment
OSGi – The Mission

Image Source: OSGi Homepage
OSGi Overview
Alliance
The OSGi Alliance is an open organization
- Established in 1999, currently 44+ members
- Membership spans many industries
- Voting members treated equally
- Membership information available at www.osgi.org

Companies start work on Java Embedded Server in 1998

Open Services Gateway Initiative launched in March 1999

First Member Meeting: London – May 1999

Specification Releases:
- R1 – May 2000 (JES Framework)
- R2 – October 2001 (Gateway Management)
- R3 – March 2003 (Automotive)
- R4 - October 2005 (Core + Mobile + Vehicle)
OSGi Members

Alpine Electronics Europe GmbH
AMI-C
Aplix Corporation
Atinav Inc. *
Belgacom
BMW Group
Cablevision Systems
Computer Associates
Deutsche Telekom AG
Echelon Corporation
Electricité de France (EDF)
Ericsson Mobile Platforms AB
Esmertec
Espial Group, Inc. *
ETRI Electronics and Telecommunications Research Institute
France Telecom
Fraunhofer Inst. for Integrated Circuits IIS *
Gatespace Telematics AB *
Gemplus
IBM Corporation
Insignia Solutions *
Intel Corporation
KDDI R&D Laboratories, Inc.
KT Corporation
Mitsubishi Electric Corporation
Motorola, Inc.
NEC Corporation
Nokia Corporation
NTT
Oracle Corporation
Panasonic Technologies, Inc.
Philips Consumer Electronics
ProSyst Software GmbH
Robert Bosch GmbH
OSGi Members

Samsung Electronics Co., Ltd.
SavaJe Technologies, Inc. *
Sharp Corporation
Siemens AG
Sun Microsystems, Inc.
Telcordia Technologies, Inc.
Telefonica I+D
TeliaSonera
Toshiba Corporation

coming soon: Vodafone

* Contributor Level Members
Working Committees

- Marketing Working Committee
- Market Requirement Working Committee

Expert Groups

- Core Platform Expert Group
- Vehicle Expert Group
- Architecture Expert Group
- Mobile Expert Group
OSGi Markets: All !!

- Automotive
- Smart Home
- Mobile Phones
- Facility Management
- Consumer Electronics
- Health Care
- Industry Automation
- ..........

Vertical market positioning reflected in with R4

Strong horizontal positioning
Soon to be released for R4

- OSGi Service Platform, Mobile Specification Submitted to JCP for adoption as JSR-232 “Mobile Operational Management”
  - Jon Bostrom, Nokia, and Venkat Amirisetty, Motorola, are co-specification leads for JSR-232.
  - Early Draft Review started 7 October 2005
  - Target Release Date: 1Q2006

- OSGi Service Platform, Vehicle Specification
  - Hans-Ulrich Michel, BMW, and Olivier Pave, Siemens AG, are co-chairs of the Vehicle Expert Group
  - Liaison with ERTICO Global System Telematics (GST) Project
  - Target Release Date: 2Q2006
OSGi Overview
Market situation
**Target Markets**

- **Mobile Devices**
  - 180M handsets/ year
  - Starting in 2005, first OSGi – enabled handsets will be shipped
  - Most interesting market due to size and short life cycle

- **Automotive Infotainment/ Telematics**
  - 60M cars/ year
  - OSGi already selected by BMW, Ford, GM, VW, Renault, Hyundai
  - Commercial applications shipping 2 years to mass deployment

- **Home Networking**
  - 190M white goods, 40M DSL Modem/ STB devices per year
  - OSGi selected by Siemens, Miele, V-Zug, Motorola, Philips, Samsung, …
  - Various products shipping, long lifecycles
Is OSGi accepted from the market?

**Telematics**
- Yes deployed in BMW 5series, 6series
- Yes deployed in various fleet management systems
- Yes deployed in Bombardier trains
- Yes to be deployed in upcoming series cars

**Residential**
- Yes deployed in Shell/Motorola Home Genie package
- Yes deployed in various White Goods applications
- Yes deployed in Philips remote controller iPronto
Is OSGi accepted from the market?

Other markets
- yes deployed in spanish ADSL router
- yes deployed as facility mgmt. system at Microsoft (!)
- yes deployed as airport parking lot control system
- yes to be deployed in German Health Care system
- yes to be deployed in various other real world setups

Mobile
- yes Nokia, Motorola actively develop mobile specification
OSGi Overview
Terms and basic functions
Open Service Gateway Specification

**Open**

Open and dynamic platform

**Service**

Run-time environment for services and apps

**Gateway**

Gateway for connecting local devices and networks with wide area networks

Specified by a heterogeneous standardization consortium
OSGi – Terms & Basic Functions

OSGi Actors

Service Gateway (SG)
• (Embedded) Platform hosting an OSGi framework
• Runtime environment for services and applications

Gateway Operator (GO)
• Runs and administers Service Gateways
• Deployment and supervision of services and data

Service Provider (SP)
• Creates and provides new services to Gateway Operator
• Example: MP3 application

Content Provider (CP)
• Provides or receives data from Service Gateways (over GO)
• Example: MP3 files for MP3 service
OSGi – Terms & Basic Functions

OSGi Actors

Service Gateways
Service Gateway Architecture

Service Gateway = Device + OSGi Framework + Services

OSGi Framework = Mini Application Server
Service Gateway Architecture

mBedded Server

- Browser Bundle
- Protocol Bundles
- Security Bundles
- OSGi Bundles
- Network Mgmt Bundles
- User Interaction Bundles
- Customized Bundles
- Bundled Native Code

Java VM

Operating System / RTOS

Processor

min. J2ME CDC/FP

Other Java Applications

Other Native Applications
OSGi Framework: key facts

- Java based runtime environment for services & apps
- Primarily focused on needs of embedded systems
- Dynamic nature: installation, updates, removals of services at runtime (Life Cycle Management)!
- Dynamic Service Registry
- Open for remote management and administration
- Dynamic resolving of package dependencies
- Requires J2ME CDC/FP based runtime
OSGi Overview
Bundles & Services
Bundles

Content of Bundles:
- Java Bytecode
- Native Code (eg. shared lib)
- Resources (eg. XML files, images, language texts, etc.)
- any files possible

Description
- Container for Services, applications and libraries
- Archive file (Java jar format)
- Dynamically loadable, resolvable and runnable by OSGi framework
OSGi Bundle

- A bundle registers 0 to N services in the framework
- The fw itself is represented as the System Bundle
- Manifest file: definition of bundle interface (Import, Export, Services, Version, Activator, …)
- Interfaces between bundles: services and shared libs
- Each bundle is loaded in a separate Class Loader!
- The fw is responsible to support the bundle’s life cycle (incl. resolution of dependencies).
Bundle Life Cycle

Figure 18  State diagram Bundle

From:
OSGi Service-Platform
Release 3, Chapter 4.8.3, Page 58
OSGi Bundle: Native Code

- A bundle can contain native libraries
- Framework checks the OS, CPU, language, etc.
- The life-cycle of the native libraries related to the life cycle of the corresponding bundle
- Java Code can access the native libraries by using JNI (Java Native Interface)
The two meanings of the word “service”

Service in terms of application qualifier
- A service may contain 0 to N software components (installed on fw and on backend)
- Example: Mobile Sales Support Tool

Service in terms of a registered interface in fw
- A software component with 1 or more Java interfaces that has been registered as a “Service” within the service registry of the framework.
Services Example

Service
GPS Service

Service
Database

Application
e.g.
POI Application
Service

- A service may come and go and runtime !!!!
- Consequently, services need to be tracked
- A service may have 1 to N interfaces
- A service might be registered multiple times with a different set of properties
- A service might be instantiated multiple times
- Service must be acquired from the Service Registry
OSGi – Bundles & Services

Service Registry

OSGi Framework
Service Registry

- Dynamic database holding all service references
- A service can only be obtained from the registry
- Support of Filter of service object search
- Provides service object references to the client
- Support of ServiceFactory for multiple service instanciation
- Contains ranking algorithm for identification of service that fit’s best to a client’s request.
OSGi Overview
Remote Management
OSGi Actors

Service Gateways

Back-end

Services, Data

INTERNET

Service Provider

Gateway Operator

Wide Area Network

Service Gateways
Remote Management Ref. Architecture:

OSGi – Backend

OSGi Service-Platform
Release 3.0, chapter 3.1.2,
page 22
Defined by OSGi:

- The remote management operations must be performed by a Management Agent

- Initial Provisioning Spec. (optional) defines the first connection between the backend and the gateway

- Remote Management Reference Architecture (fig. on the previous page)

- Management Agent Bundles are defined by the gateway operator
Not defined by OSGi:

- The protocol(s) between backend and service gateway
- Implementation recommendation for the management agent
- Security architecture
- Goal: Not to restrict the specific needs of the gateway operators and service providers
OSGi Overview

Excurs: MIDP versus OSGi Model
Main differences

<table>
<thead>
<tr>
<th>MIDP Model</th>
<th>OSGi R3 Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLDC based</td>
<td>CDC 1.0 / FP 1.0 based</td>
</tr>
<tr>
<td>Static set of APIs</td>
<td>APIs and application code updates &amp; upgrades possible at runtime</td>
</tr>
<tr>
<td>Focus: MIDlet application model</td>
<td>Focus: component model, not application model</td>
</tr>
<tr>
<td>Lightweight applications</td>
<td>Almost unlimited Java core features</td>
</tr>
<tr>
<td>Market dedication: mobile phones</td>
<td>Horizontal market orientation</td>
</tr>
</tbody>
</table>
Overview of stacks

- **MIDlets only**
- **Only MIDlet at a time**

- **Multiple apps (types, instances)**
- **Core APIs are not static**

### Core APIs
- MIDP
- OSGi
- MEG
- Other

### Frameworks
- CLDC
- CDC
- native device management framework
- FP
- JNI

### Stacks
- OS & Drivers
- MIDlet
- Mgm
- Other

- **MIDlet(s)**
- **MEGlet(s)**
- **Other(s)**
OSGi Overview

OSGi Specification Details
Base platform: the framework

- Bundle & service administration (life cycle management, dependencies)
- Service registry
- Event transport
- Boot management
- Permissions

Runtime environment for OSGi services
The Package Admin Interface

- Bundles can come and go at any time. This requires the bundle dependencies and states to be recalculated.
- PackageAdmin is a service interface registered by fw.
- It provides information to a Management Agent about a bundle’s exported packages.
Package Management

- Lazy and eager update
- Package sharing:
  - Classes
  - Resources
Permission Administration

- The framework keeps a central **store of permissions**
- A bundle has dedicated or default permissions
- The PermissionAdmin service (registered by fw) provides read/write access to the permissions store
- Bundle permissions can be modified **before, during or after** a bundle has been installed.

- Note: permission concept is based on *The Java Security Architecture*
OSGi based logging: the Log Service

- The LogService provides means to write log messages
- The LogReaderService can retrieve log entries
- Log changes can be tracked by a listener
- Categorization of log entries by Log Levels
- Persistency is not specified

- Important tool for keeping logging information
Configuration Admin & ManagedService

- Centralized management of bundle configurations
- ConfigurationAdmin Service:
  - Maintains repository of bundle configs
  - Provides API to access (read/write) this data
  - Delivers configurations to bundles automatically
- Runtime configuration changes of bundles!
- ManagedServices:
  - Service registered by bundle to receive config data
  - Used by ConfigAdmin to pass config data
- Persistency layer not defined by OSGi
Track your services!

- The dynamic environment of OSGi requires services to be tracked
- OSGi provides a utility API called `ServiceTracker`
- Features:
  - get service references for specified service(s)
  - receive callbacks in case of service state changes
  - filtering is possible, of course.

- Partly shields the dynamics.
OSGi Overview

Benefits and disadvantages
At development time

- Standardized technology
- Availability of ready products and functions
- Modularity based on a good component model
- Reusable components
- Parallel development and portability
- Complementary to other existing standards
- Fast and secure programming environment

Reduction of development time and costs !!
At run time

- Standardized platform base technology
- Dynamic installation/update of software components
- Offering of new services
- Offering of services from different SPs
- Remote administration and configuration
- Remote diagnostics
Some general disadvantages

- Added overhead for RAM and flash
- Little amount of vertical services defined
- Complex specification process which involves many participants with different interests
- Missing standard GUI framework model
## Typical hardware configurations

<table>
<thead>
<tr>
<th>Profile</th>
<th>CPU</th>
<th>RAM</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum (Record)</td>
<td>50 MHz</td>
<td>8 MB</td>
<td>4 MB</td>
</tr>
<tr>
<td>Low End</td>
<td>50 MHz</td>
<td>16 MB</td>
<td>8 MB</td>
</tr>
<tr>
<td>Medium</td>
<td>200 MHz</td>
<td>32 MB</td>
<td>16-32 MB</td>
</tr>
<tr>
<td>High End</td>
<td>400 MHz</td>
<td>&gt; 32 MB</td>
<td>&gt; 32 MB</td>
</tr>
</tbody>
</table>
OSGi Release 4 Extensions
OSGi R4 Documentation Plan

Core Specification
- Framework Implementer’s spec
- Bundle Programmer’s guide

Service Compendium
- Service
  - Service
  - Service
  - Service

Mobile
- Ref Arch / Guide
  - Constraints
  - Service
  - Service
  - Service

Vehicle
- Ref Arch / Guide
  - Constraints
  - Service
  - Service
  - Service

Other
- Ref Arch / Guide
  - Constraints
  - Service
  - Service
  - Service
Core Specification Defines The

- Framework – rewritten and updated
  - Module Layer
  - Lifecycle Layer
  - Service Layer
- Framework Services
  - Package Admin - updated
  - Start Level
  - Conditional Permission Admin - updated
  - Permission Admin - updated
  - URL Handlers
Framework Layering

**L3** – Provides a publish/find/bind service model to decouple bundles

**L2** - Manages the lifecycle of bundle in a bundle repository without requiring the VM be restarted

**L1** - Creates the concept of modules (aka, bundles) that use classes from each other in a controlled way according to system and bundle constraints

**L0** -
- OSGi Minimum Execution Environment
- CDC/Foundation
- J2SE
Service Compendium

- Log Service - updated
- Http Service - updated
- Device Access
- Configuration Admin - updated
- Preferences Service - updated
- Metatype - updated
- Wire Admin
- User Admin
- IO Connector

- Initial Provisioning
- UPnP Device - updated
- Declarative Services - new
- Event Admin - new
- Service Tracker - updated
- XML Parser
- Position
- Measurement and State
- Execution Environments - updated
Service Programming Model With R4 – Declarative Services

- Simplifies the service oriented programming model (XML description)
- Assists bundle developers in their work
- Handles the dynamic of the service objects

**R3 Programming Model**

META-INF/Manifest.mf

Bundle-Activator: com.velingrad.Hello

```java
@com.velingrad.Hello.java
public class Hello implements BundleActivator {
    public void start(BundleContext bc) {
        ServiceTracker tracker = new ServiceTracker(bc,
            "org.osgi.service.log.LogService", null);
        tracker.open();
        LogService log = (LogService) tracker.getService();
        if (log != null) {
            log.log(LogService.LOG_INFO, "Hello Velingrad");
        } else {
            // ??? what to do here
        }
    }
    public void stop(BundleContext bc) {
    }
}
```

**R4 Programming Model**

META-INF/Manifest.mf

Service-Component: OSGI-INF/activator.xml

```xml
@OSGI-INF/activator.xml
<?xml version ="1.0" encoding="UTF-8"?>
<component name="example.Hello">
    <implementation class="com.velingrad.Hello"/>
    <reference name="LOG" interface="org.osgi.service.log.LogService"/>
</component>
```

```java
@com.velingrad.Hello.java
public class Hello {
    protected void activate(ComponentContext cc) {
        LogService log = (LogService) cc.locateService("LOG");
        log.log(LogService.LOG_INFO, "Hello Velingrad");
    }
}
```
Generic Event Model

The problem in R3:

- Usual event pattern for applications:
  - listening parties register services with dedic. interf.
  - event source references all services with such dedicated interfaces and passes the event over

- Since services (event source service, event listener services) may come and go, this results in extra programming efforts and potential failures
Generic Event Model

The solution in R4:
- Central Generic Event service
- Simple topic based publish/subscription model
- Synchronous and asynchronous event delivery
- Support for wildcards
- Easy binding to native event sources/consumers
Mobile Architecture Overview

OSGi Service Platform
Mobile Service Platform - OSGi specifics for mobile devices

MEG is actively working to define Mobile Service Platform (MEG R4). The official release is scheduled for the first quarter of 2006.

MEG R4 will be based on OSGi R4 Framework which has been extended from the R3 Framework with...

..and for the Mobile Service Platform the enhancements are:

- Security and Policy Framework
- Permissions and Signatures
- Generic Events Mechanism

- Deployment model and infrastructure
- Application model and lifecycle
- Device management functionality
Deployment Package

- Deployment Package
  - Based on JAR Format
- Manifest describes the resources and associates them with a Resource
- Processor
- Fix Packages
- Provide only updated contents
The basic OSGi architecture is management protocol agnostic

- Provides a model where many parties can participate

What is missing is an abstraction to manage a device in detail

The OMA DM protocol is dominant in the mobile device market

- Will be supported by a wide range of devices

The MEG therefore supports the OMA DM management model with the Dmt Admin Service
The management tree organizes all available management objects in the device as a hierarchical tree structure where all nodes can be uniquely addressed with a URI.

- ./SyncML/DMAcc/xyzInc
DMT Basics

- Introduced in support of the SyncML DM (now OMA DM) protocol
- DMT is a tree of interior and leaf nodes
- All nodes in the data tree have names
- Only leaf nodes have values
- Base value types:
  - Integer
  - String
  - Boolean
  - Binary
  - XML
- (b64 | bin | bool | chr | int | node | null | xml | date | time | float)
DMT Basics

- OMA DM defines five possible operations on the nodes of DMT
  - Add,
  - Get,
  - Replace,
  - Delete,
  - Execute

- ACL of a node specified which entity is permitted to perform the different operations on that node.

- The DMT is dynamic and it is not required to be stored
ProSyst Plugin Structure – OSGi Mobile Management Tree

- **OMA DM Client**
  - OMA_ALERT
  - DmtAdmin

- **DMT Admin**
  - DmtPlugin
  - AlertSender

**ProSyst CU Model**

- **DMT CU**
  - CU
  - CU

**DMT Plugins**

- **Standard OMA DM Objects**
  - DmtPlugin
- **Config Plugin**
  - DmtPlugin
- **Log Plugin**
  - DmtPlugin
- **Monitoring Plugin**
  - DmtPlugin
- **Deployment Plugin**
  - DmtPlugin
- **Download Plugin**
  - DmtPlugin
- **Application Plugin**
  - DmtPlugin
- **Policy Plugin**
  - DmtPlugin
- **CU Plugin**
  - DmtPlugin

**Management Services**

- **Config Admin**
- **Log Service**
- **Monitor Admin**
- **Deployment Admin**
- **Download Agent**
- **Application Management Framework**

**OSGi Core**

- **Framework**
- **Package Admin**
- **Permission Admin**
- **Conditional Permission Admin**
Generic Application Model - A generic model that is intended to abstract different application models so they can be treated as one

- Provides for third party screen managers
- Provides for rich GUIs
  - Icons, help, etc.
  - Can monitor the state of running instances
- Interacts with JSR 211 Content Handlers

Foreign Application Model - defines how non-OSGi Applications can access and provide services

- Header usage
- Access to Framework class
The OSGi Vehicle Profile shares its architecture with the Mobile Profile

The Vehicle Profile provides specific vehicle oriented services

The Vehicle Profile uses many more of the Core Compendium Services because it is more mature

It is likely the vertical profiles will come closer in the future

- Start Level Service
- URL Handlers
- Package Admin Service
- Permission Admin Service
- Log Service
- Http Service
- Device Access
- Configuration Admin Service
- Metatype(2) Service
- Preference Service
- User Admin Service

- Wire Admin Service
- IO Connector Service
- Declarative Services
- Event Admin Service
- Power Management Service
- Diagnostic Service
- Service Tracker Utility
- XML Parser Utility
- Position Utility
- Measurement and State Utility
The power management service makes power management pluggable.

The system power state can be set externally:
- Full Power
- PM Active
- Suspend
- Sleep
- Power off

is mapped to different device power state:
- D0-D3 power states

Power manager can take device specific capabilities in consideration.

An observer bundle can follow the transitions in the system and device power state.
OSGi Products
Products available on the market
OSGi Products - ProSyst

**mBedded Server**
Embedded Software

**mBedded Builder**
Development IDE

**mPower Remote Manager**
Remote Management Software

**Deploy and Deliver**
**Develop and Debug**
**Manage and Maintain**
Siemens VDO & OSGi Alliance

- Siemens VDO is involved in OSGi Alliance since 2001

is the Siemens VDO Platform

- multimedia system based on OSGi Service Platform
- used in BMW 1, 3, 5 and 6 Series, X3
- used in current developments for other Car makers
WCTME 5.7.2 tools

MIDP mobile handhelds

Advanced mobile handhelds, tablets, laptops, desktops

WCTME 5.x IDE

Enterprise MIDP Tools

Web Services for MIDP

WME CLDC/MIDP

WME CDC/F/PBP/PP, OSGi MEE, J2SE

WSDD

eSWT

Extension Services (Includes Web Services)

SMF Bundle Development Kit

Application Tools for Extension Services

J2EE Tooling

J2EE

WSSD/AD

Eclipse

“Embedded” WCTME 5.7.2
WebSphere ED (Client) 6.0

Enterprise and ISV Applications

- Web Application View with SWT Browser Component
- Application Launcher/Switcher
- Preference Pages

Middleware Shell
- JFace
- SWT
- Generic UI frame
- Update Manager
- Help UI (with Lucene)

Interaction Services
- Core Extension Point Framework

Access Services
- MicroBroker
- JMS
- Transaction Container
- JDBC
- XML Parsing
- Web Services
- SyncML

Managed Client Services
- OSGi Framework
- OSGi Services

Framework
- Class Library
- Virtual Machine
Relation to JCP

- The relation to the JCP is troublesome
- Several JSRs overlap with JSR 232
- JSR 277 Modularization
  - However, long way off from J2ME
- JSR 271 MIDP 3.0
  - Is addressing some of the solutions that MEG provides
- JSR 246 OMA DM Access
  - Based on JSR 232 Dmt Admin, but slightly different
  - Needs to be merged
- JSR 249/248 MSA CDC/CLDC
  - Must select JSR 232 to make MEG viable
Resources

- http://www.osgi.org
- http://www.osgi.org/blog/index.html
- http://dz.prosyst.com
- http://eclipse.org/equinox
Thank you! For further information please contact us!

Pavlin Dobrev
ProSyst Labs EOOD
Vladajska Str. 48
Sofia 1606, Bulgaria
Tel. +359 2 952 35 81
Fax +359 2 953 26 17

p.dobrev@prosyst.com

www.prosyst.com
dz.prosyst.com – ProSyst Developer Zone (free registration)
OSGi Programming
Theory and Hello World Example

These slides are not intended to be presented. They will be available for all users that download the presentation.
Purpose of the Hello World bundle

• Demonstrate the Bundle lifecycle

• Demonstrate how a bundle is developed, packed and deployed manually

• Demonstrate how a bundle is developed, packed and deployed using the OSGi plug-in
Basic Interfaces
Bundle: the BundleActivator interface

- Implemented by some class in bundle
- Used by framework to start and stop a bundle
- Must be assigned through manifest
- Interface signature:

```java
package org.osgi.framework;

public interface BundleActivator {
    // start method
    public void start (BundleContext) throws Exception;
    // stop method
    public void stop (BundleContext) throws Exception;
}
```
Bundle: the BundleContext interface

- Implemented by framework
- Represents the execution environment of the bundle
- Acts as a proxy between framework and the bundle
- Instantiated by framework on bundle start

Doc: OSGi R3 Spec, chapter 4.23.5, page 98
Bundle: the BundleContext interface

- What bundles can do with BundleContext:
  - Register services in the framework
  - Retrieve services from the framework
  - Subscribe to framework events
  - Obtain a persistent storage area
  - Interrogate other bundles
  - Install new bundles in the framework
Bundle: the Bundle interface

- Implemented by framework
- One Bundle object instantiated for each bundle (by fw)
- Represents the bundle and its state
- Used to observe and control a bundle’s life-cycle
- Can list all registered services
- Can list all used services
Never Forget!

Whatever you do – keep in mind that bundles and service come and go at runtime!
Creating a Hello World bundle manually

**Step 1:** Create a working directory: `/osgi_test/`

**Step 2:** Create a `BundleActivator` Implementation class
- Create the package directory: `/osgi_test/examples/hello`
- Create in it a Java file containing the `BundleActivator` implementation:

```java
public class Activator implements BundleActivator {
    public void start(BundleContext bc) throws Exception {
        System.out.println("Bundle Started !!");
    }

    public void stop(BundleContext bc) throws Exception {
        System.out.println("Bundle Stopped !!");
    }
}
```
Creating a Hello World bundle manually

Step 3: Create a Manifest file:
• Location of the file: /osgi_test/META-INF/Manifest.mf
• Content of the Manifest

  Bundle-Activator: examples.hello.Activator
  Bundle-Category: examples
  Bundle-Vendor: Nokia
  Bundle-Version: 1.0
  Bundle-Name: Hello World Bundle
Creating a Hello World bundle manually

**Step 4:** Compile the `Activator.java` with `javac`

```
javac -classpath %MBS%/lib/frameworklib.jar examples/hello/*.java
```

**Step 5:** Pack the bundle with the `jar.exe` from the JDK

```
jar cmf META-INF/Manifest.mf hello.jar examples/hello/*.class
```

**Step 6:** Optionally write a script file that will atomize these two steps

**Step 7:** Deploy the bundle on the framework using the mBS console:

```
fw>$ install -s /osgi_test/hello.jar
```
ProSyst OSGi Eclipse Plug-in

New OSGi Project or Bundle

OSGi Bundles

Install Bundle

Make Bundle jar file

Code Generate

Java Editor. Code of the Bundle Activator

Framework Console
Creating a Hello World bundle with the Eclipse plug-in

**Step 1:** Create Bundle with the plug-in:
- Right-click on the Bundles tree and choose New/Bundle
- Specify the Bundle Name & click the Finish button

**Step 2:** Create the BundleActivator implementation class:
- Right-click on the Hello Bundle node in the Bundles tree and choose Code Generate/Bundle Activator
- Implement the start & stop methods

**Step 3:** Edit the manifest file
- The Bundle-Activator header is generated automatically
- Add the additional headers
Creating a Hello World bundle with the Eclipse plug-in

Step 4: Generate a bundle JAR file
• Right-click the Hello Bundle node and choose Pack
• Select a jar file name & location and click the Finish button

Step 5: Install the bundle in the framework, and start/stop it
• Right-click the Hello Bundle node and choose Install Bundle