Equinox Framework:
A Happier OSGi R6 Implementation

Tom Watson
IBM

March 18th 2014
Agenda

• New to OSGi R6 Core
• Redesign Core Equinox and Why
• What Breaks?
OSGi R6 Core

• Finalizing specification in March 2014
• Equinox Luna release will be the RI (June 2014)
• R6 Noteworthy Additions
  • Service Scopes
  • Data Transfer Objects
  • Native Namespace
  • Extension Bundle Activators
  • WeavingHook Enhancements
OSGi R6 Core – Service Scopes

• Identify the scope of a service object
OSGi R6 Core – Service Scopes

• Identify the scope of a service object
  • Singleton – single instance used by all bundles

Singleton object that implements the service contract

service.scope = singleton
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles

Each Bundle can get access to the singleton object via the registry
OSGi R6 Core – Service Scopes

• Identify the scope of a service object
  • Singleton – single instance used by all bundles

Each Bundle can get access to the singleton object via the registry

A

B

C

Bundles

Service Registry

gets
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle

```
service.scope = bundle
```
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle

Each Bundle can get access to a bundle specific service object via the registry
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
OSGi R6 Core – Service Scopes

• Identify the scope of a service object
  • Singleton – single instance used by all bundles
  • Bundle – Service Factory used to create a service object for each bundle
OSGi R6 Core – Service Scopes

• Identify the scope of a service object
  • Singleton – single instance used by all bundles
  • Bundle – Service Factory used to create a service object for each bundle
  • Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
  - Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service

service.scope = prototype
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
  - Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service

ServiceObjects.getService()
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
  - Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service
Identify the scope of a service object

- Singleton – single instance used by all bundles
- Bundle – Service Factory used to create a service object for each bundle
- Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service
OSGi R6 Core – Service Scopes

• Identify the scope of a service object
  • Singleton – single instance used by all bundles
  • Bundle – Service Factory used to create a service object for each bundle
  • Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service

```
ServiceObjects.getService()
```
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
  - Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
  - Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service

```java
ServiceObjects.getService()
```
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
  - Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service
OSGi R6 Core – Service Scopes

• Identify the scope of a service object
  • Singleton – single instance used by all bundles
  • Bundle – Service Factory used to create a service object for each bundle
  • Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service
OSGi R6 Core – Service Scopes

- Identify the scope of a service object
  - Singleton – single instance used by all bundles
  - Bundle – Service Factory used to create a service object for each bundle
  - Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service

```java
ServiceObjects.getService();
```
Identify the scope of a service object

- Singleton – single instance used by all bundles
- Bundle – Service Factory used to create a service object for each bundle
- Prototype (new for R6) – Prototype Service Factory used to create a service object for each request for the service
Data Transfer Objects

• An object that carries data between processes
• The motivation for its use has to do with the fact that communication between processes is usually done resorting to remote interfaces (e.g. web services), where each call is an expensive operation.
• Does not have any behavior except for storage and retrieval of its own data

https://en.wikipedia.org/wiki/Data_transfer_object
OSGi R6 Core – Data Transfer Objects

• DTOs represent the state of runtime objects
  • Core Framework supplies a number of DTOs
• Easily serializable
  • Only has public fields with limited types
  • Must form a tree to simplify serialization
• Data contained is a snapshot of the state
  • DTOs do not track state changes
  • DTOs are not thread safe
OSGi R6 Core – Obtaining Core DTOs

- Core Framework DTOs obtained via bundle adapt

```java
DTOType dto = bundle.adapt(DTOType.class)
```
OSGi R6 Core – Obtaining Core DTOs

- Core Framework DTOs obtained via bundle adapt
- `BundleDTO` – for a bundle: state, id, name, version.

```java
BundleDTO dto = bundle.adapt(BundleDTO.class)
```
OSGi R6 Core – Obtaining Core DTOs

- Core Framework DTOs obtained via bundle adapt
- BundleDTO – for a bundle: state, id, name, version.
- BundleRevisionDTO – for a bundle revision: requirements, capabilities

```java
BundleRevisionDTO dto
    = bundle.adapt(BundleRevisionDTO.class)
BundleRevisionDTO[] dtos
    = bundle.adapt(BundleRevisionDTO[].class)
```
OSGi R6 Core – Obtaining Core DTOs

- Core Framework DTOs obtained via bundle adapt
- BundleDTO – for a bundle: state, id, name, version.
- BundleRevisionDTO – for a bundle revision: requirements, capabilities
- BundleWiringDTO – for a bundle wiring: required and provided wires

```java
BundleWiringDTO dto = bundle.adapt(BundleWiringDTO.class);
BundleWiringDTO[] dtos = bundle.adapt(BundleWiringDTO[].class);
```
OSGi R6 Core – Obtaining Core DTOs

- Core Framework DTOs obtained via bundle adapt
- BundleDTO – for a bundle: state, id, name, version.
- BundleRevisionDTO – for a bundle revision: requirements, capabilities
- BundleWiringDTO – for a bundle wiring: required and provided wires
- ServiceReferenceDTO – for a registered service: properties, using bundles

```java
ServiceReferenceDTO[] dtos = bundle.adapt(ServiceReferenceDTO[].class)
```
OSGi R6 Core – Obtaining Core DTOs

• Core Framework DTOs obtained via bundle adapt
• BundleDTO – for a bundle: state, id, name, version.
• BundleRevisionDTO – for a bundle revision: requirements, capabilities
• BundleWiringDTO – for a bundle wiring: required and provided wires
• ServiceReferenceDTO – for a registered service: properties, using bundles
• FrameworkDTO – for the framework: list installed bundles, services, launch props

```java
FrameworkDTO dto
    = systemBundle.adapt(FrameworkDTO.class)
```
OSGi R6 Core – Obtaining Core DTOs

- Core Framework DTOs obtained via bundle adapt
- BundleDTO – for a bundle: state, id, name, version.
- BundleRevisionDTO – for a bundle revision: requirements, capabilities
- BundleWiringDTO – for a bundle wiring: required and provided wires
- ServiceReferenceDTO – for a registered service: properties, using bundles
- FrameworkDTO – for the framework: list installed bundles, services, launch props
- BundleStartLevelDTO – for a bundle start-level and start options

BundleStartLevelDTO dto
    = bundle.adapt(BundleStartLevelDTO.class)
OSGi R6 Core – Obtaining Core DTOs

- Core Framework DTOs obtained via bundle adapt
  - BundleDTO – for a bundle: state, id, name, version.
  - BundleRevisionDTO – for a bundle revision: requirements, capabilities
  - BundleWiringDTO – for a bundle wiring: required and provided wires
  - ServiceReferenceDTO – for a registered service: properties, using bundles
  - FrameworkDTO – for the framework: list installed bundles, services, launch props
  - BundleStartLevelDTO – for a bundle start-level and start options
  - FrameworkStartLevelDTO – for the framework start-level and initial bundle start-level

FrameworkStartLevelDTO dto
  = systemBundle.adapt(FrameworkStartLevelDTO.class)
OSGi R6 Core – Native Namespace

- Describes the native environment the framework is running
- Framework provides a capability in the `osgi.native` namespace
- `Bundle-NativeCode` headers are converted into requirements in the `osgi.native` namespace
- Last requirement type in OSGi to be transformed into the generic requirements/capabilities model
OSGi R6 Core – Extension Bundle Activators

• **Allows** `system.bundle` **fragments** to hook into the initialization and shutdown process of the framework

• **Use case** involves scenarios that require hooks to be in place before any other bundles are resolved

• **Equinox regions** uses this to ensure the regions are established before any bundle resolution operations occur
OSGi R6 Core – Weaving Hook Enhancements

• **New** *WovenClassListener*
  • Listen to class weaving process to detect new dynamic imports
  • Needed by isolation engines, such as subsystems, to provide access to imports added by *WeavingHook* implementations

• **Package import permissions are auto-granted to Woven bundles for package imports added by a *WeavingHook***
  • With security enabled a woven bundle may be prevented from wiring to a package required by a dynamic import added by a *WeavingHook*
Equinox – Moving Forward

• Equinox Framework in need of a redesign
  • One unhappy implementation

• Dependencies are modelled differently from the generic dependency model of the OSGi specification
  • Requires adapting Equinox resolver types to OSGi wiring types
  • Needed to call out to resolver and weaving hooks

• Hard to reason about locking strategy
  • OSGi Resolver and Weaving Hooks introduced more callouts to user code
  • Too many abstraction layers!

• Hard to prototype core OSGi changes for the reference implementation
  • Dependency Resolution (e.g. Native Namespace)
  • Creating DTO snapshots
Equinox – Redesign the Core

- Implement a core module container
Equinox – Redesign the Core

- Implement a core module container
- Responsible for managing resources
Equinox – Redesign the Core

• Implement a core module container
• Responsible for managing resources
  • Lifecycle
Equinox – Redesign the Core

- Implement a core module container
- Responsible for managing resources
  - Lifecycle
  - Metadata – Capabilities and Requirements
Equinox – Redesign the Core

- Implement a core module container
- Responsible for managing resources
  - Lifecycle
  - Metadata – Capabilities and Requirements
  - Resolution
Equinox – Redesign the Core

- Implement a core module container
- Responsible for managing resources
  - Lifecycle
  - Metadata – Capabilities and Requirements
  - Resolution
  - Persistent settings

Container

Database
Equinox – Redesign the Core

- Implement a core module container
- Responsible for managing resources
  - Lifecycle
  - Metadata – Capabilities and Requirements
  - Resolution
  - Persistent settings
- Uses an OSGi R5 Resolver service
  - Currently Apache Felix
Equinox – Redesign the Core

- Implement a core module container
- Responsible for managing resources
  - Lifecycle
  - Metadata – Capabilities and Requirements
  - Resolution
  - Persistent settings
- Uses an OSGi R5 Resolver service
  - Currently Apache Felix
- Concurrency in mind from the start
  - Properly handle call outs (hooks) while holding no locks
- Tools may use it for modelling OSGi wiring (PDE)
Equinox – Redesign the Core

• Implement a core module container
• Responsible for managing resources
  • Lifecycle
  • Metadata – Capabilities and Requirements
  • Resolution
  • Persistent settings
• Uses an OSGi R5 Resolver service
  • Currently Apache Felix
• Concurrency in mind from the start
  • Properly handle call outs (hooks) while holding no locks
• Tools may use it for modeling OSGi wiring (PDE)
• Does not deal with bundle content, class loading, service registry, event listeners, security
Equinox – Redesign the Framework

Unhappy Equinox

Happy Container

I care about that big $X$

bundle content
class loading
service registry
event listeners
security
Equinox – Redesign the Framework

• Uses the container as the core
• Keeps complicated concurrency concerns in one place
  • Uses reentrant read/write lock for the module 'database'
  • For long running operations optimistic locking is used. For example, resolution.
Equinox – Redesign the Framework

• Uses the container as the core
• Keeps complicated concurrency concerns in one place
  • Uses reentrant read/write lock for the module 'database'
  • For long running operations optimistic locking is used. For example, resolution.
• Implements Framework bits outside of the container
  • Bundle content
  • Class loading
  • Service registry
  • Event listeners
  • Security
Equinox – Configuration

- Don't access config statically
- Get rid of static class `FrameworkProperties`
- Instead make available an instance of `EquinoxConfiguration`
- Greatly simplifies and optimizes embedding and starting multiple Framework instances
  - A special class loader was needed to isolate static config data for each framework instance

Happy Equinox
Equinox – Hooks

- **Equinox Hooks are still supported**
  - Much of the internal details changed requiring an overhaul of the Equinox Hooks SPI
  - ALL existing Equinox Hook implementers are broken and will need migration

- **Unhappy Equinox had too many Hooks**
  - Three class loading hooks – now one, but no ability to override bundle class loader implementation
  - Bundle File hook – removed, use wrapper hook instead
  - Adaptor hook – removed, no more adaptor
  - Bundle Watcher – removed, was a glorified bundle listener
  - Added ability to specify a bundle activator that is called during framework start/stop
  - Improved Storage Hook to handle hooks being added/removed better
Equinox – Old State/Resolver

- Equinox Container has no implementation of old Equinox resolver API
  - Still exports the Equinox resolver API
  - No longer registers the PlatformAdmin service

Happy Equinox Container
Equinox – Old State/Resolver

- Equinox Container has no implementation of old Equinox resolver API
  - Still exports the Equinox resolver API
  - No longer registers the PlatformAdmin service
- Equinox Compatibility fragment
  - Hooks into Equinox to register a PlatformAdmin implementation
  - Attempts to mirror the Equinox wiring state in the PlatformAdmin implementation
  - Has some limitations modeling the running state
  - Useful for offline modeling, for example: PDE
Equinox – Plugin Converter

- Equinox Container has no implementation of the Plugin Converter
  - Support for Eclipse 2.0 style plugins
  - Still exports the plugin converter API
  - No longer registers the `PluginConverter` service
Equinox – Old State/Resolver

• Equinox Container has no implementation of the Plugin Converter
  • Support for Eclipse 2.0 style plugins
  • Still exports the plugin converter API
  • No longer registers the `PluginConverter` service

• Equinox Compatibility fragment
  • Hooks into Equinox to register a `PluginConverter` implementation
  • Converts Eclipse 2.0 style plugins into proper bundles
Equinox – What Breaks?

- All Equinox Framework Hook implementations must be migrated
- Some limitations for the compatibility resolver
  - Disconnected from the running wiring state
  - Cannot be used to diagnose resolution errors
  - Cannot be used to disable bundles at runtime
Equinox – Current Status

• Work is being done in the git master branch
  • http://git.eclipse.org/c/equinox/rt.equinox.framework.git/log/?h=master
  • http://git.eclipse.org/c/equinox/rt.equinox.bundles.git/log/?h=master

• Implementation has been in Luna since M1

• Fully compliant with the current suite of compliance tests for OSGi R6
  • Will be the reference implementation of the Core Framework for R6

• Successful if nobody notices
  • Except for the poor Equinox Adaptor Hook implementors
Evaluate This Session

1. Sign-in: [www.eclipsecon.org](http://www.eclipsecon.org)

2. Select session from schedule

3. Evaluate: +1 0 -1
Trademarks

IBM and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.