OSGi Bundle Manifest Headers

Version 3.1 - Last revised June 20, 2005

A bundle can carry descriptive information about itself in the manifest file named META-INF/MANIFEST.MF. The OSGi R4 Framework specification defines a set of manifest headers such as Export-Package and Bundle-Classpath, which bundle developers use to supply descriptive information about a bundle. The Eclipse OSGi Framework implements the complete OSGi R4 Framework specification and all of the Core Framework services. The OSGi R4 Core Framework services include the following:

- Package Admin Service Specification
- URL Handlers Service Specification
- Start Level Service Specification
- Conditional Permission Admin Specification
- Permission Admin Service Specification

There are a number of optional services defined in the OSGi R4 specification. The optional services are not included with the Eclipse OSGi Framework implementation. For information on OSGi R4 manifest headers and services refer to the OSGi specifications.

Eclipse Bundle Manifest Headers

The Eclipse OSGi Framework supports a number of additional bundle manifest headers and directives. A bundle developer may use these additional headers and directives to take advantage of some additional features of the Eclipse OSGi Framework which are not specified as part of a standard OSGi R4 Framework.

Additional Export-Package Directives

The Eclipse OSGi Framework supports additional directives on the Export-Package header. These directives are used to specify the access restriction rules of an exported package. See osgi.resolverMode to configure the Eclipse OSGi Framework to enforce the access restriction rules at runtime.

The x-internal Directive

The x-internal directive can be used in an Export-Package header to specify whether the package is an internal package. The Plug-in Development Environment will discourage other bundles from using an internal package. If the x-internal directive is not specified then a default value of 'false' is used. The x-internal directive must use the following syntax:

```
x-internal ::= ( 'true' | 'false' )
```

The following is an example of the x-internal directive:

```
Export-Package: org.eclipse.foo.internal; x-internal:=true
```

The x-friends Directive

The x-friends directive can be used in an Export-Package header to specify a list of bundles which are allowed access to the package. The Plug-in Development Environment will discourage other bundles from using the package. The x-friends directive must use the following syntax:

```
x-friends ::= '"' ( target-bundle ) ( ',' target-bundle ) * '"'
target-bundle ::= a bundle symbolic name
```

The following is an example of the x-friends directive:

```
Export-Package: org.eclipse.foo.formyfriends; x-friends:="org.eclipse.foo.friend1, org.eclips
```

The example specifies that only the bundles org.eclipse.foo.friend1 and org.eclipse.foo.friend2 should be encouraged to use the org.eclipse.foo.formyfriends package. The x-internal package takes priority over the x-friends directive. If the x-internal directive specifies 'true' then The Plug-in Development Environment will discourage all bundles from using the package even if they are specified as a friend.

The Eclipse-LazyStart Header

The Eclipse-LazyStart header is used to specify if a bundle should be started automatically before the first class or resource is accessed from that bundle. This feature allows Eclipse to activate bundles lazily the first time they are needed. Using this model Eclipse can startup with as few active bundles as possible. The Eclipse-LazyStart header must use the following syntax:

```
Eclipse-LazyStart ::= ( 'true' | 'false' ) ( ';' 'exceptions' '=' '"' exceptions-list '"' ) ?
exceptions-list ::= a comma ',' separated list of packages
```

The 'exceptions' attribute is used to specify a list of packages which must not cause the bundle to be activated when classes or resources are loaded from them. If the Eclipse-LazyStart header is not defined in the bundle manifest then a default value of 'false' is used. The following is an example of the Eclipse-LazyStart header:

```
Eclipse-LazyStart: true; exceptions="org.eclipse.foo1, org.eclipse.foo2"
```

The example specifies that this bundle must be activated for any classes or resources that are loaded from this bundle, except the classes and resources in the packages 'org.eclipse.foo1' and 'org.eclipse.foo2'.

The Eclipse-AutoStart header has been deprecated in Eclipse 3.2. The Eclipse-LazyStart header should be used instead.

The Eclipse-PlatformFilter Header

The Eclipse-PlatformFilter is used to specify a platform filter for a bundle. A platform filter must evaluate to true in a running platform in order for a bundle to be allowed to resolve. The Eclipse-PlatformFilter header must use the following syntax:

```
Eclipse-PlatformFilter ::= a valid LDAP filter string
```

The Framework supports filtering on the following system properties:

- **osgi.nl** - the platform language setting.
- **osgi.os** - the platform operating system.
- **osgi.arch** - the platform architecture.
- **osgi.ws** - the platform windowing system.

The following is an example of the Eclipse-PlatformFilter header:

```
Eclipse-PlatformFilter: (& (osgi.ws=win32) (osgi.os=win32) (osgi.arch=x86))
```

This example specifies that this bundle can only be resolved if the platform properties are osgi.ws=win32 and osgi.os=win32 and osgi.arch=x86. In other words a platform running on an x86 architecture, using a win32 operating system and the win32 windowing system.

The Eclipse-BuddyPolicy Header

The Eclipse-BuddyPolicy header is used to specify the buddy classloading policies for a bundle. The Eclipse-BuddyPolicy header must use the following syntax:

```
Eclipse-BuddyPolicy ::= ( policy-name ) ( ',' policy-name ) *
policy-name ::= ( 'dependent' | 'global' | 'registered' |
                  'app' | 'ext' | 'boot' | 'parent' )
```

The following is an example of the Eclipse-BuddyPolicy header:

```
Eclipse-BuddyPolicy: dependent
```

The Eclipse-RegisterBuddy Header

The Eclipse-RegisterBuddy header is used to specify a list of bundles for which this bundle is a registered buddy. The Eclipse-RegisterBuddy header must use the following syntax:

```
Eclipse-RegisterBuddy ::= ( target-bundle ) ( ',' target-bundle ) *
target-bundle ::= a bundle symbolic name
```

The following is an example of the Eclipse-RegisterBuddy header:

```
Eclipse-RegisterBuddy: org.eclipse.foo.bundle1, org.eclipse.foo.bundle2
```

The Eclipse-ExtensibleAPI Header

The Eclipse-ExtensibleAPI is used to specify whether a host bundle allows fragment bundles to add addtional API to the host. This header should be used if a host bundle wants to allow fragments to add additional packages to the API of the host. If this header is not specified then a default value of 'false' is used. The Eclipse-ExtensibleAPI header must use the following syntax:

```
Eclipse-ExtensibleAPI ::= ( 'true' | 'false' )
```

The following is an example of the Eclipse-ExtensibleAPI header:

```
Eclipse-ExtensibleAPI: true
```

The Eclipse-GenericCapabilty Header

The Eclipse-GenericCapability header is used to specify a generic capability of a bundle. Generic capabilities can be used to describe features of your bundle which can be required by other bundles in the system (using the Eclipse-GenericRequire header). Generic capabilities are given a name and a capability type. Capability types are defined by the bundle which is offering the capability. Capabilities can also have a set of typed matching attributes which are used to match against when resolving Eclipse-GenericRequire headers. Matching attributes may be one of the following types; [string | version | uri | long | double | set]. The set type can be used to define a set of strings as a comma separated list of strings. The Eclipse-GenericCapability header must use the following syntax:

```
Eclipse-GenericCapability ::= capability ( ',' capability ) *
capability                ::= typed-name ( ';' typed-name ) *
                              ( ';' typed-param ) *
typed-name                ::= name ( ':' capability-type )
typed-param               ::= typed-key '=' quouted-string
typed-key                 ::= name ( ':'
                              [string | version | uri | long | double | set] )
```

The following is an example of the Eclipse-GenericCapabilty header that could be used to specify a bundle with an OSGi service org.acme.stuff.SomeService implementation:

```
Eclipse-GenericCapability:
 org.acme.stuff.SomeService:osgi.service;
 version:version="1.0.1"
```

The Eclipse-GenericRequire Header

The Eclipse-GenericRequire header is used to specify a requirement on ageneric capability which is offered by another bundle (using the Eclipse-GenericCapability header). Generic requirements are given a name and a capability type. Capability types are defined by the bundle which is offering the capability. Generic requirements can specify an LDAP filter string which is used as a selection filter to resolve against matching generic capabilities. The Eclipse-GenericRequire header must use the following syntax:

```
Eclipse-GenericRequire   ::= generic-require ( ',' generic-require ) *
generic-require          ::= typed-name ( ';' typed-name ) *
                             ( ';' selection-filter '=' quoated-ldapFilter )
                             ( ';' optional '=' [true|false] )
                             ( ';' multiple '=' [true|false] )
typed-name                ::= name ( ':' capability-type )
```

The following is an example of the Eclipse-GenericRequire header that could be used to specify a bundle that depends on an OSGi service org.acme.stuff.SomeService implementation:

```
Eclipse-GenericRequire:
 org.acme.stuff.SomeService:osgi.service;
 selection-filter="(version>=1.0.1)"
```

Generic Aliases

The osgi.genericAliases option can be used to map existing OSGi manifest headers onto Eclipse-GenericCapability and Eclipse-GenericRequire manifest headers. For example, consider the following manifest headers

```
Export-Service: org.acme.stuff.SomeService
Import-Service: org.acme.stuff.SomeService
```

These headers can be mapped to Eclipse-GenericCapability and Eclipse-GenericRequire headers using the following property:

```
osgi.genericAliases=Export-Service:Import-Service:osgi.service
```

Under the covers this would translate into the following generic headers:

```
Eclipse-GenericRequire: org.acme.stuff.SomeService:osgi.service
Eclipse-GenericCapability: org.acme.stuff.SomeService:osgi.service
```

The Plugin-Class Header

The Plugin-Class header is only used to support plugins developed for the Eclipse 2.1 platform. This header is used to specify a class name that will be used to activate a plug-in using the old Eclipse 2.1 activation model. New bundles developed for Eclipse 3.0 or greater should not use this header. The following is an example of the Plugin-Class header:

```
Plugin-Class: org.eclipse.foo.FooPlugin
```