Dynamic Workspace Bundles

Contents

[A Dynamic Workspace for Plug-ins and Bundles 1](#_Toc349215661)

[How Workspace bundles are updated 2](#_Toc349215662)

[Target and development platform 2](#_Toc349215663)

[Bundle Versions 2](#_Toc349215664)

[Bundle Dependencies 2](#_Toc349215665)

[Activate Providing Bundles 3](#_Toc349215666)

[Deactivate Requiring Bundles 3](#_Toc349215667)

[Update Bundles 3](#_Toc349215668)

[Dependency Rules 3](#_Toc349215669)

[Bundle States and Transitions 4](#_Toc349215670)

[Introduction 4](#_Toc349215671)

[Transitions and Bundle Operations 4](#_Toc349215672)

[Bundle Project 4](#_Toc349215673)

[Workspace and Composite Commands 4](#_Toc349215674)

[Project CRUD Operations 5](#_Toc349215675)

[Implicit and Explicit Commands 5](#_Toc349215676)

[The Bundle State Diagram 5](#_Toc349215677)

# A Dynamic Workspace for Plug-ins and Bundles

A workspace plug-in project or bundle project is installed and started from an originating plug-in source project after Eclipse has been started. The term bundle project (or workspace bundle) is used when referring to both the project and the bundle at the same time as one coherent unit. A deployed plug-in originates from a jar bundle and is typically installed and started during startup of Eclipse. Both workspace and deployed plug-ins are dynamic in the sense that they may change state after they are installed. Workspace bundles dynamically change its runtime behavior when source projects are updated or changed during an Eclipse session. Deployed bundles require a manual re-installation or an installation of a new version to change its runtime behavior.

A workspace bundle can be dependent on and require capabilities from deployed bundles, but a deployed bundle, with a few exceptions, cannot require capabilities from a workspace bundle. The exceptions include call backs from deployed bundles to workspace bundles at runtime and workspace fragment bundles attached to a deployed bundle. From this follows that deployed bundles provide capabilities to workspace bundles and a workspace bundle does not provide capabilities to deployed bundles during an Eclipse session.

Workspace bundle projects can be viewed as a set of plug-ins in a workspace region disjoint from the set of deployed plug-ins providing capabilities to the workspace plug-ins. A workspace bundle first becomes a jar bundle when it is deployed.

# How Workspace bundles are updated

It is not possible to update jar bundles dynamically within an Eclipse session as opposed to workspace bundles. For workspace bundles the build system in combination with the dynamics of OSGi is utilized is to resolve and start (execute) plug-ins automatically after they have been built.

The tight loop between source code changes and execution is especially efficient when doing Rapid Application Development (RAD) testing small incremental changes in plug-ins. The functionality is in general useful for any tool or project (e.g. design tools or code generators) in need for a runtime environment embedded in the development workspace.

# Target and development platform

Traditionally to activate and execute workspace plug-ins projects in Eclipse the Run command is used to launch a new instance of the Eclipse IDE, called the target platform. By default this activates all jar bundles defined in the runtime configuration of the target together with the plug-in projects in the workspace

The target platform is known as the execution or runtime environment for the plug-ins under development. The source or development platform is where the bundles are developed and built. The InPlace Activator treats the target and development platform the same and thus the same instance of the IDE (workbench) is used both for the target and development platform.

# Bundle Versions

Versions become important when bundles are packed in jars, deployed, installed and resolved by an external OSGi container. In a deployed scenario different versions of a bundle may run at the same time where each version usually only has one revision. In the same manner different versions of workspace bundles with the same symbolic name, but different location identifiers, may run simultaneously. This may be the case if you copy a project to a different location, changes the version and then activates the bundle.

# Bundle Dependencies

A new revision is assigned to a bundle each time it is installed and updated and removed when the bundle is uninstalled. Old revisions are removed when a bundle is refreshed.

A requiring bundle dependency implies that a bundle directly or indirectly requires some capabilities provided by another bundle (called the requiring dependency closure).

For example, an Export-Package clause is a capability and an Import-Package clause is a requirement. During the resolving phase the requirements are resolved to matching capabilities by creating a Bundle Wire. A wiring is created each time a bundle revision is resolved, holding the wires to other bundle wirings as well as maintaining the run time state of the bundle

If the contract between two bundles is broken because exported packages are withdrawn or the API of interfaces in exported packages is broken this will result in compile time errors and the involved bundles will not resolve. In this case, if any, the current revision and wiring is used.

When installing, updating and executing bundles directly from compiled source files, the last resolve holds the current revision and its wires of the bundle. Every time a bundle is updated after a change, a new revision called the current revision of the bundle is created and when resolved new wires are bound to that revision. Earlier revision(s) and the wires of the bundle are released during the update process when bundle is refreshed after an update.

A workspace bundle may have many revisions at the same time. Other bundles may require capabilities or services from different revisions of a bundle. A bundle changes from the usage of an older revision to a newer revision of a bundle if possible when the bundle is refreshed. Bundles may be refreshed manually from the UI and are always refreshed automatically when deactivated (unresolved or uninstalled) and after update if the [Refresh on Update](../tasks/Setting%20Bundle%20Options.htm#RefreshOnUpdate) option is switched on (default).

## Activate Providing Bundles

If bundle A is dependent on bundle B, and bundle A is activated, bundle B has to be activated to. If you activate a providing bundle, as in this case bundle B, this does not imply activation of the requiring bundle (bundle A).

## Deactivate Requiring Bundles

When deactivating bundles the rule is reversed. That is, deactivating a providing bundle (bundle B) the requiring bundle is also deactivated (bundle A), but when deactivating a requiring bundle (bundle A), bundle B is unchanged.

## Update Bundles

When a bundle is updated, the bundle and all requiring bundles are stopped in correct dependency order. Then the bundle is updated and refreshed if the [Refresh on Update](../tasks/Setting%20Bundle%20Options.htm#RefreshOnUpdate) option is switched on. The refresh operation releases any existing wires and new wires are created. When the option is off the bundle is resolved after update instead of refreshed. Lastly all providing bundles are started in correct dependency order.

## Dependency Rules

The activation/deactivation is in principle controlled by three basic dependency rules:

* **Directed**  
  Dependencies between bundles goes only in one direction; - From a requiring bundle to a providing bundle. The requiring (dependent) bundle is dependent on the providing (independent) bundle.
* **Acyclic**   
  States that a dependency relationship cannot be circular. If bundle A requires capabilities from a providing bundle B, bundle A is said to be dependent on B. Thus bundle B cannot at the same time be dependent on bundle A. This is also true for transitive dependencies, as stated in the next bullet.
* **Transitive**  
  Dependency relationships spans across bundles. If bundle A is dependent on bundle B and bundle B is dependent on bundle C, bundle A is also dependent on bundle C.

When a bundle is activated or deactivated the direction of the dependency and the acyclic and transitive properties of the relationships between bundles controls which additional bundles are activated or deactivated.

By design the InPlace Activator automatically enforce changes in life cycle of bundles that have relationships to other bundles that have their life cycle changed. For instance when a bundle that requires capabilities form another bundle is activated the providing bundle is also activated.

When an Eclipse session ends all bundles are uninstalled before Eclipse shut down. When starting a new session, activated bundles from earlier sessions are by default started but bundles in state RESOLVED at shutdown return to state RESOLVED at next startup.

# Bundle States and Transitions

## Introduction

As mentioned workspace bundles originates from a plug-in project in an Eclipse workspace. Deployed (or jar) bundles have different characteristics compared to workspace bundles. Due to these differences, the OSGi settings and commands are tailored for handling dependencies and how they affect life cycle changes of workspace bundles in a development environment where bundles are updated and activated/deactivated repeatedly.

This section presents an overview of the rules and a summary of bundles states and transitions and how to interpret them. A list of relevant terms is presented first.

## Transitions and Bundle Operations

There is a one-to-one relationship between a transition and an OSGi command or operation. Every OSGi command used in this section has a corresponding transition in the state diagram.

### Bundle Project

A project is an Eclipse plug-in project or an OSGi framework bundle project. There is a bi-directional one-to-one relationship between a project and an activated workspace bundle. A workspace bundle is a singleton and referred to as a bundle project when the relationship between a project and its bundle is of importance.

### Workspace and Composite Commands

A workspace command is a simple or a composite OSGi command. Composite commands consist of a combination of one or more OSGi commands, and possibly one of the Eclipse build commands. Activate, Deactivate and Reset are composite commands.

### Project CRUD Operations

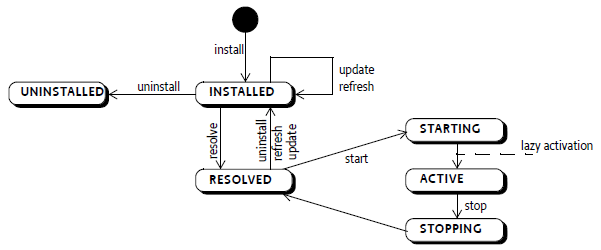
All workspace and project commands that manipulate a project as a whole or the content in a project associated with a bundle are considered CRUD operations. The operations are Create (create, open, import project), Rename (rename, move project), Update (save, save all file(s)) and Delete (delete, close project). All CRUD operation triggers a workspace bundle command, except for Update, which must be followed by a build command to trigger a workspace bundle command.

### Implicit and Explicit Commands

Bundle commands are divided into explicit (activate, deactivate, reset, refresh, start, stop) and implicit (install, uninstall, update, resolve) commands. Explicit commands are executed directly by you as a developer. Implicit commands are executed automatically when a project CRUD or an explicit command is issued by the developer.

## The Bundle State Diagram

The state diagram is copied from the OSGi Service Platform Release 4, Version 4.3 specification and gives an overview of the possible life cycle states and transitions that a bundle undergoes. The OSGi specification gives a detailed explanation of each command (transition) and state.



You can inspect the current state and the last transition executed on a bundle in the Details View and the Properties Page for a bundle. In the following, the states and transitions are discussed from the perspective of workspace bundles and the different commands (transitions) provided by the InPlace Activator.

#### Explicit Composite Commands triggered from the UI

* **Activate**A bundle is either in state UNINSTALLED or INSTALLED when deactivated. If in state UNINSTALLED it is first installed, then resolved and lastly started. If in s state INSTALLED, the bundle is first updated, then resolved and started. A started bundle enters state ACTIVE or STARTING (<<LAZY>>) according to the declared activation policy in the manifest file.
* **Deactivate**  
  A bundle is moved to state UNINSTALLED if no other bundles are activated, and moved to state INSTALLED if there are other activated bundles in the workspace. A bundle can be deactivated from any activated state (RESOLVED, STARTING/<<LAZY>>, STOPPING and ACTIVE). If the bundle is in state ACTIVE/STARTING it is first stopped, moving the bundle to state RESOLVED. In an activated workspace the bundle is refreshed (unresolved), moving the bundle to state INSTALLED. In a deactivated workspace (implies that this is the last bundle in the workspace to deactivate) the bundle is uninstalled moving the bundle to state UNINSTALL.
* **Reset**When an activated bundle is reset it is first uninstalled. When in state UNINSTALLED it is installed, resolved and started if it was in state ACTIVE or STARTING before the Reset command was issued.

#### Explicit OSGi Commands triggered from the UI

* **Stop**When a bundle is in state ACTIVE or STARTING it can be stopped. When stopped it is moved to state RESOLVED.
* **Start**When a bundle is in state RESOLVED it can be started. A started bundle enters state ACTIVE or STARTING (<<LAZY>>) according to the declared activation policy in the manifest file. If the bundle has an eager activation policy and both requires and use some capabilities from a lazy activated bundles, the lazy activated bundle is started and moved to state ACTIVE due to “on demand” class loading. This presumes that the dependency option for starting bundles includes starting providing bundles.
* **Refresh**A bundle in state RESOLVED, ACTIVE, STARTING and STOPPING may be refreshed. Refresh on bundles in state INSTALLED is not put in use. Bundles are stopped, moving them to state RESOLVED before refreshed. Refresh first unresolves the bundle before it resolves it again constructing new wires. The resolving process is an iterative process that searches through the active workspace and the deployed bundles. If possible a bundle always returns to the same state it had before being refreshed.  
    
  It is a goal to minimize the number of revisions for a bundle to one. The refresh operation is available to manually refresh bundles with more than one revision.
* **Update**If the [Update on Build](../tasks/Setting%20Bundle%20Options.htm#UpdateOnBuild) option is off a bundle is updated manually from the UI. If the option is on (default) [update](#UpdateImplicit) is an implicit command.

#### Implicit Commands triggered by the InPlace Activator

* **Install**When a bundle is installed a unique bundle object is created and all remaining life cycle operations are performed upon this object. Every bundle is uniquely identified by its location string and is assigned a unique bundle identifier and a new revision.  
    
  Multiple bundles with the same symbolic name and version are not allowed. Only the first in a set of duplicate bundles are installed.
* **Uninstall**Bundles are uninstalled when the workspace is deactivated (when there are no activated bundles in the workspace). The Framework removes any resources related to the bundle and uninstalls the bundle from the persistent storage of the Framework.   
    
  The InPlace Activator removes all dependencies when uninstalling the bundle. When the bundle is uninstalled, this implies that the workspace is deactivated, and there are no requirements left from the workspace or deployed bundles on the uninstalled bundle. The bundle is moved to state UNINSTALLED when uninstalled.
* **Update**When a project is built, the updated version of the bundle is read from the input stream and installed, resolved and possibly started. If the [*Refresh on Update*](../tasks/Setting%20Bundle%20Options.htm#RefreshOnUpdate) option is switched on the bundle is refreshed instead of explicit resolved after update. A new revision and wires is also created for the bundle as part of the update process. If possible the bundle returns to the state it was in before the update operation. The update command is not run when the bundle is deactivated (state UNINSTALED or INSTALLED).
* **Resolve**The resolve process, which also is part of the update process, checks that all specified dependencies for the bundle is satisfied starting with the dependency related headers in the manifest (e.g. imports and exports), creates a wiring between the bundles involved and assigns the revision, created when the bundle was installed or updated, to the wired bundle.   
    
  The resolve operation moves the bundle from state INSTALLED to RESOLVED. When resolved all Java classes that the bundle needs are available. This state indicates that the bundle is either ready to be started or has stopped. Resolve is invoked each time a bundle is Activated, Reset, Updated or Refreshed.