Bundle Options

Contents

[How to 2](#_Toc350373601)

[Setting Properties in the New Plug-in or OSGi Bundle Project Wizard 2](#_Toc350373602)

[Output folder 2](#_Toc350373603)

[Target Platform 3](#_Toc350373604)

[UI Contributions 3](#_Toc350373605)

[Partial Dependencies 4](#_Toc350373606)

[Default values 4](#_Toc350373607)

[Sets and Graphs 5](#_Toc350373608)

[How to interpret the different Dependency Options 5](#_Toc350373609)

[Build Errors and Dependency Options 6](#_Toc350373610)

[Global Bundle commands 6](#_Toc350373611)

[Refresh on Update 6](#_Toc350373612)

[Update on Build 7](#_Toc350373613)

[Update Bundle-ClassPath on Activate/Deactivate 7](#_Toc350373614)

[Use the osgi.dev System Property 8](#_Toc350373615)

[Use one Output Folder per Source Folder 8](#_Toc350373616)

[Add the Output Folder Manually to the Bundle-ClassPath 8](#_Toc350373617)

[Set Activation Policy to Eager on Activate 9](#_Toc350373618)

[Eager Activation Policy 9](#_Toc350373619)

[Handle External Commands Automatically 9](#_Toc350373620)

[Allow Plug-Ins that makes Contributions to the Workbench 9](#_Toc350373621)

[Context Bindings 10](#_Toc350373622)

[Dynamic UI Contributions 10](#_Toc350373623)

[Standard Perspectives 10](#_Toc350373624)

[Customized Perspectives 10](#_Toc350373625)

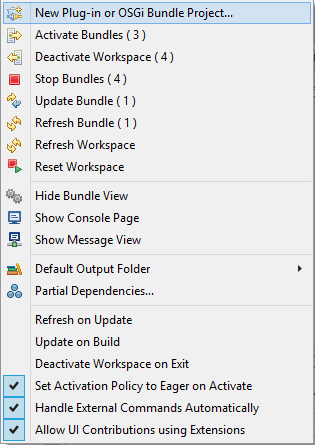
[Dynamic Menu contributions in Version 4.2 10](#_Toc350373626)

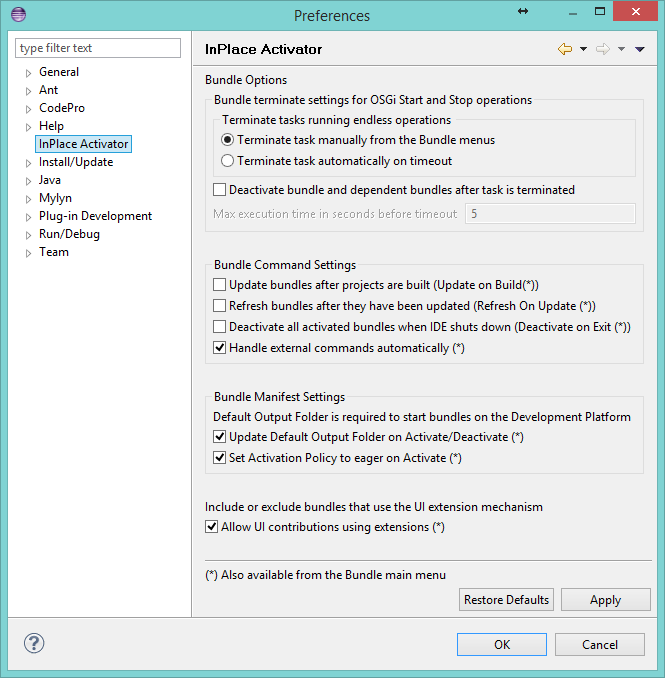
[Extensions and Extension Points 10](#_Toc350373627)

[Saving Resources before executing Bundle Operations 11](#_Toc350373628)

# How to

All options are placed on the Bundle menus. The global options are set in the Bundle main menu and the *Eager Activation Policy* option for individual bundles from the context pop-up menus in the Package Explorer, the Bundle List Page View and the Bundle Details Page View.

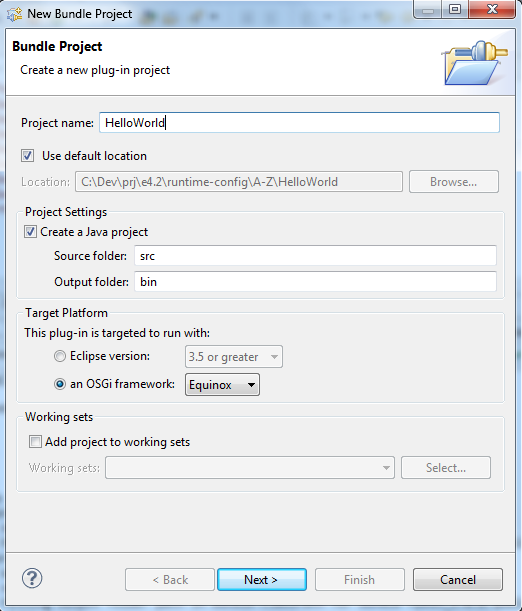




Bundle dependencies are set in the *Partial Dependencies* dialog activated from the Bundle main menu. The dialog is modeless making it possible to experiment with the different options while the dialog is open.

# Setting Properties in the New Plug-in or OSGi Bundle Project Wizard

This is a convenience entry in the Bundle main menu, opening the same wizard as the *New Plug-in Project* wizard from the File menu entry. Properties in this wizard that influence how bundles are activated and updated are commented in this section.



## Output folder

This setting specifies the default output folder and is where class files are ordinarily generated (and resource files, copied). The location you specify in this field is added to the Bundle-ClassPath header in the manifest file and used when bundles are started (See [Start and Stop Bundles](Start%20and%20Stop%20Bundles.htm).) The Bundle-ClassPath is updated automatically if the [Update Bundle-ClassPath on Activate/Deactivate](#_Update_Bundle_Class-Path) is switched on (default).

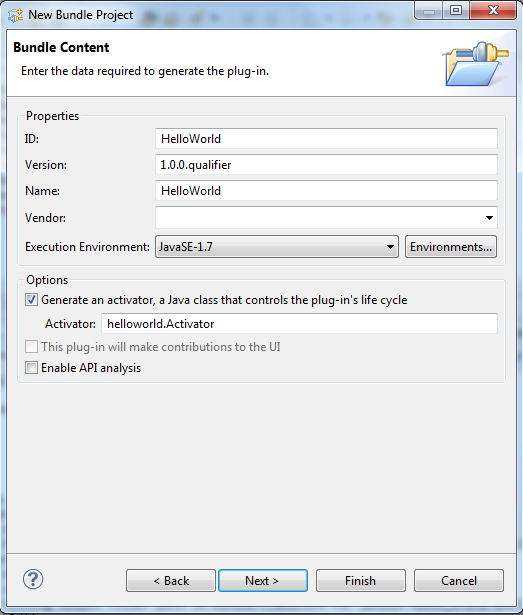
## Target Platform

The InPlace Activator is compatible with both target platforms available from the *Target Platform section* in the wizard*.*

If you select *an OSGi Framework* as the *Target Platform* a pure OSGi bundle project is created with a dependency on the imported OSGI framework (“org.osgi,framework” package). Selecting Eclipse as the target platform makes your plug-in dependent on the Eclipse runtime (“org.eclipse.core.runtime” plug-in) and indirectly of the OSGI framework and other plug-ins through the Eclipse runtime plug-in.

## UI Contributions

If you select the OSGi Framework as the target platform the *“this Plug-in will make contributions to the UI”* option will be disabled. This is logical since UI contributions make the plug-in dependent on the Eclipse runtime and the standard UI framework (“org.eclipse.ui” plug-in) in Eclipse, and your plug-in would then not become a pure OSGi bundle (See the [Target Platform](#_Target_Platform) section above). If you select Eclipse runtime as the target platform you may or may not let the plug-in make UI contributions.



When you create a plug-in with UI contributions this is typically done using extensions which may be declarative (specified in plugin.xml) or dynamic (created by code). If you use UI contributions, please check out the [Allow Plug-Ins that makes Contributions to the Workbench](#_Allow_Plug-Ins_that) option for any limitations.

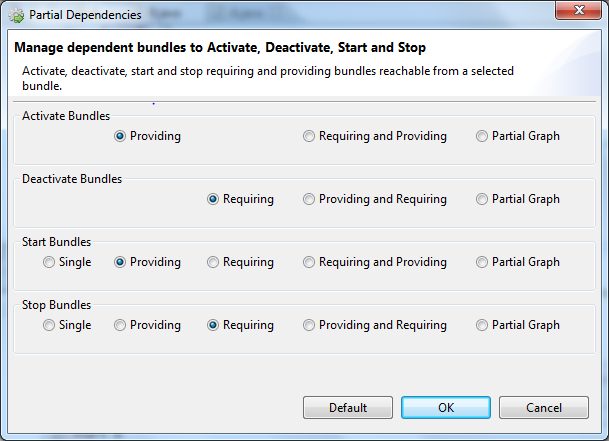
# Partial Dependencies

The dependency options determine which bundles to include when a bundle is being activated, deactivated, started or stopped from the Bundle List Page View or the Bundle Details Page View.

From the perspective of a selected or initial bundle there is one set of bundles providing capabilities to the bundle and one set of bundles requiring capabilities from the bundle. Both the providing and requiring set may be empty meaning that the bundle has no dependencies on other bundles.

An inherent characteristic of dependencies between bundles is transitivity, which has a cascading effect activating/deactivating or starting/stopping a chain of bundles from an initial bundle that directly or indirectly has dependency relationships to other bundles.

Requiring bundles are always deactivated when a providing bundle is deactivated and opposite, providing bundles are always activated when a requiring bundle is activated. In all other cases, the behavior is defined by the dependency options.



## Default values

The default is to include providing bundles on activate and start and requiring bundles on deactivate and stop.

Caution should be taken if you select the *Single* or *Requiring* options on *Start* due to missing dependencies or *Single* or *Providing* on *Stop* due to possible stale references. See [Start and Stop Bundles](Start%20and%20Stop%20Bundles.htm# StaleReferences ) for a discussion about stale references and garbage collection.

Besides this all options are considered convenience options where you can tune them in such a way that dependent bundles are activated/deactivated and started/stopped automatically depending on your needs.

## Sets and Graphs

For those of you familiar with sets and graphs, all bundles - workspace and external - are viewed as a set, where the workspace bundles is a subset, containing one or more partitions, where each partition is a directed acyclic graph and the dependency relation between the elements or bundles in the graph is defined by the different import (Require-Bundle, …) headers in the manifest file. For each option a partial dependency order of bundles is constructed to be included in the bundle operation to execute. Thus it is possible to traverse and sort the set of workspace bundles in dependency (requiring or providing) order, based on a random set of initial bundles.

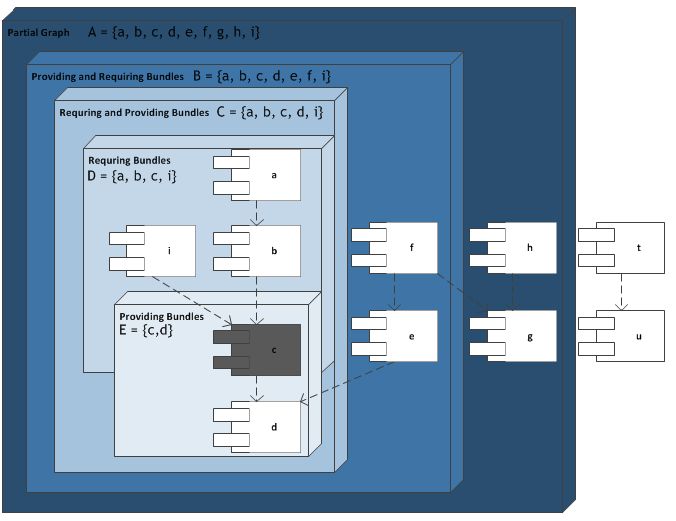
## How to interpret the different Dependency Options

Which bundles to include when a bundle is activated, deactivated, started or stopped is probably best understood by an example.

In the following example:

* The set of workspace bundles are S = {a, b, c, d, e, f, g, h, i, t, u}
* Bundle c is the selected bundle or more formally the initial set ({c}) of bundles for all options.

The figure illustrates in a graphical manner and in set notation the outcome of each option. The outcome or result set is sorted in lexical (not dependency) order.



The*Providing and Requiring Bundles* option for deactivate and stop with {c} as the initial set is interpreted as follows:

1. Start with bundle c as the initial set {c}
2. Calculate the set of providing bundles {c, d} with c as the initial bundle. Note that the initial set of bundle(s) is always part of the result set.
3. Calculate the set of requiring bundles {a, b, c, d, e, f, i} with the providing bundles {c, d} as the initial set of bundles as calculated in step 2.

The *Providing Bundles* option is the same as calculated in step 2 above.

The same applies to the *Requiring and Providing Bundles* option for activate and start, except that the requiring bundles are calculated before the providing bundles:

1. Start with bundle c as the initial set {c}
2. Calculate the set of requiring bundles {a, b, c, i} with c as the initial bundle. Note that the initial set of bundle(s) is always part of the result set.
3. Calculate the set of providing bundles {a, b, c, d, i} with the requiring bundles {a, b, c, i} as the initial set of bundles as calculated in step 2.

The *Requiring Bundles* option is the same as calculated in step 2 above.

The *Partial Graph* option traverses the bundle dependencies in both directions until it is not possible to extend the graph further.

### Build Errors and Dependency Options

Start and Stop operations are not affected by bundle errors. Deactivating bundles with build errors prohibit activation of the affected bundles until the errors are corrected. Affected bundles are bundles with build errors and bundles that have requirements on those bundles. Only providing bundles of bundles with build errors are activated. To which extend providing bundles will be activated is controlled by the dependency option used. See [Bundle Project Errors](../concepts/Bundle%20Project%20Errors.htm) for a general description of error handling in workspace bundle projects.

## Global Bundle commands

When activating, deactivating, starting or stopping bundles from the Bundle main menu all possible candidate bundles in the workspace are included in the selected operation and automatically becomes the set of initial bundles. As noted, initial bundles are always included in the result set. Dependent bundles are then either in the initial set or already part of the result set. The consequence of this is that the Partial Dependency options have no effect on global bundle commands initiated from the Bundle main menu.

# Refresh on Update

Updating bundles will create new bundle revisions while the existing bundle wirings remain wired to their previous bundle revisions. This stale wiring to old bundle revisions is cleaned up by the refresh command. This option affects the method used when new revisions and wirings between dependent bundles are resolved.

When the *Refresh on Update* option is switched on, new wirings between the updated bundles and all directly and indirectly requiring bundles (requiring dependency closure) are created and the old ones are released or garbage collected by the OSGi refresh operation.

If build errors are introduced in already activated bundles and such bundles are members of the calculated requiring dependency closure to refresh a path of bundles including all requiring and providing bundles of the error bundles from the requiring dependency closure are removed. The removed bundles that are not refreshed retain their current revision with its wirings.

When the *Refresh on Update* option is switched off, existing bundle wirings remains wired to the previous revision of a bundle being updated. You may experience that the number of revisions for a bundle is greater than one when the option is switched off. Turning this option on or issuing a manual Refresh command should ensure only one revision per bundle. The number of revisions for a bundle may be found in the List *Page or the* Details Page in the Bundle View.

A simple scenario, when not refreshing on update, is two activated bundles (in state ACTIVE/LAZY or RESOLVED) where A requires and uses some capabilities from B. Change and build B. This triggers an update which creates a new current revision for B. Bundle B has now two revisions; - the old or previous one with a wiring to A; and - a new current revision. For A to be wired to B’s current revision and release the previous revision of B a refresh on B, or an another activated bundle providing capabilities to B, is necessary. One the other hand if you are updating a single bundle with no dependencies a resolve would suffice and there will be no need for an extra refresh after update.

It can be confusing when a requiring bundle (A) is wired to a previous revision of a providing bundle (B). Any compiled source changes from the latest update of the providing bundle is part of the current revision while the requiring bundle is wired to the previous (or an older) revision. This means that the requiring bundle is utilizing the functionality from the previous revision and not the changes introduced by the last update (current revision) of the provider.

# Update on Build

Activated bundle projects are by default updated automatically after a build. You can turn this switch off and control the update frequency manually by issuing an update command for all activated bundle projects that have been built since last update from the main menu or for individual bundle projects from one of the local pop-up menus or toolbars. Note that only activated bundle projects that have been built since last update are updated. See [Update Bundles Manually](Update%20and%20Refresh.htm#UpdateBundlesManually) for a description of the rules when updating individual bundles from the local bundle menus.

Bundles are refreshed after a manual update if the [Refresh on Update](#_Refresh_on_Update) option is switched on and resolved if the option is switched off. Note that activating a bundle project in an activated workspace is always updated after it is built as part of the activation process. Thus this option first becomes operative after bundle projects have been activated.

# Update Bundle-ClassPath on Activate/Deactivate

The default output folder is where the class (and resource) files are stored. The path to the generated class files are needed when a bundle project is started. The default output location is initially set in the [*Output folder*](#OutputFolder) field in the *New Plug-in Project* wizard when you create a new project and defaults to the bin folder. The default location can be changed when you create the project, from the Java Build-Path entry in the project properties dialog under the source tab, from the Java preference (Windows | Preferences | Java | Build Path) dialog after the project is created or by changing it directly in the “.classpath” file.

If you are using the default location for the output folder, which is also the default setting in Eclipse, and the *Update Bundle-ClassPath on Activate/Deactivate* option is switched on, the Bundle Class-Path header in the manifest file is updated automatically with the default output location when the bundle is activated and removed when deactivated.

If this option is turned off, no action is taken to update the Bundle-ClassPath on activation. If the Bundle-ClassPath was not updated with the output folder when the bundle was activated it is possible to update the Bundle-ClassPath for already activated bundles from the main menu. For full control you can add and remove the default output location in the Bundle-ClassPath on any – deactivated and activated - bundle project from one of the local menus in the Bundle View.

## Use the osgi.dev System Property

If you don’t like the idea of updating the Bundle-ClassPath when activating and deactivating bundle projects you can set the osgi.dev system property in the configuration.ini file to the default output folder. The configuration.ini file is located in the ECLIPSE\_HOME/configuration folder where ECLIPSE\_HOME is the folder where you installed Eclipse.

The setting in the configuration.ini can then be specified as *osgi.dev=bin* where bin (or what you have changed it to) is the default output folder. When the osgi.dev system property is set you can turn the *Update Bundle-ClassPath on Activate/Deactivate* option off.

It is also possible to reference a properties file as a parameter to the osgi.dev system property. The output folder of each activated bundle is then updated with the bundle symbolic name as the key and the output folder as the attribute in the specified property file (e.g. Windows: osgi.dev=file\:c\:/dev/prj/dev.properties). This is more dynamic and useful when bundles have different output folders or the output folder of a bundle is changed during an Eclipse session.

## Use one Output Folder per Source Folder

If you later override the default location by setting an output folder per source folder you must manually add the new output location to the Bundle-ClassPath or use the osgi.dev system property. This feature makes it possible to arrange generated class files for different source folders in different output folders, and not necessarily the default output folder. This means that the generated class files for the project may end up scattered across several folders, rather than all in the default output folder, which is the default.

See the "*Allow output folders for source folders*" option accessible from the project properties dialog in the “Java Build Path” menu entry under the source tab, to associate specific output folders to source folders.

## Add the Output Folder Manually to the Bundle-ClassPath

You add an output folder to the Bundle Class-Path manually either directly in the manifest file or in the *Classpath* section from the Runtime tab in the Plug-in Manifest Editor of the bundle. To bring up the manifest editor select the newpprj_wiz.gif tool bar button in the Bundle View for the selected bundle project or open the manifest.mf file in Package or project Explorer. If you for instance use output/bin as the output folder for one of your source folders, the class path header should at least include the following entry: *Bundle Class-Path: output/bin*.

# Set Activation Policy to Eager on Activate

By default the New Plug-in Project wizard in Eclipse creates a bundle project with lazy activation policy. A standalone bundle with lazy activation will enter state <<LAZY> (or STARTING) when activated and the start method in the Activator class will not be executed unless some other bundle force one of the classes in the lazy bundle to load.

By switching this option on, the bundle will activate eagerly and enter state ACTIVE, causing the start method in the Activator class to execute.

This and the [Eager Activation Policy](#_Eager_Activation_Policy) option respond to the OSGi Bundle-ActivationPolicy header in the manifest file and ignore the older Eclipse-LazyStart and the deprecated Eclipse-AutoStart headers.

This option simply manipulates the Bundle-ActivationPolicy header in the manifest file.

# Eager Activation Policy

This option is set on a per bundle basis from one of the Bundle context menus. The option toggles between lazy and eager activation policy. Toggling this option on an activated bundle updates the bundle after changing the policy setting of the bundle, according to the chosen policy.

This option is overruled by the [*Set Activation Policy to Eager on Activate*](#SetActivationPolicyToEagerOnActivate)option when a deactivated bundle is being activated.You can fine tune the policy for each bundle with this option by switching off the *Set Activation Policy to Eager on Activate* option.

# Handle External Commands Automatically

To keep a consistent state between workspace bundles the InPlace Activator listens to OSGi commands from the framework, third party bundles and external tools as for instance the Host OSGi Console. When this option is turned off a modal dialog is provided where you have the option to choose what action the InPlace Activator should take when an uninstall command is executed in an activated workspace. When turned on, default actions are performed silently by the InPlace Activator. See [External Bundle operations](External%20Bundle%20operations.htm) for a description of possible actions.

# Allow Plug-Ins that makes Contributions to the Workbench

Eclipse was initially not designed to dynamically activate and deactivate bundles in the middle of a session; - that is between startup and shutdown of the IDE or an application. The rewriting of the kernel in Eclipse 3.0 supporting OSGi as the runtime architecture with the Equinox implementation, opened up for dynamic loading and unloading of bundles. This dynamic behavior revealed some issues regarding context binding, perspectives and the extension mechanism.

This option is provided to avoid accidental activation and loading of bundles that require capabilities from the Workbench or more specific the Eclipse UI plug-in (“org.eclipse.ui”) and derived UI plug-ins.

When this option is switched off, bundles that make contributions to the UI will not be activated. This also includes bundles that require capabilities from UI enabled bundles. When switched on all plug-in projects are candidates to be activated.

## Context Bindings

[[Bug 295662]](https://bugs.eclipse.org/bugs/show_bug.cgi?id=295662), clutter up the context bindings in Eclipse when plug-ins using the command framework is resolved and unresolved after startup. This bug is resolved internally in the InPlace Activator as a workaround until an official fix comes along. If you use the OSGi Console or any other tool to resolve or unresolve bundles using the command extension point dynamically, you may receive binding errors in the console. Note that this option is not relevant for other UI libraries (e.g. JFC/Swing).

## Dynamic UI Contributions

### Standard Perspectives

There is annoying feature - not necessarily a bug - when loading and unloading UI dependent bundles. Every time a bundle that contributes with a visual UI component is unloaded while in a standard perspective a modal dialog pops-up asking for resetting the perspective. To avoid this, the workaround is to create a custom perspective using the *Window | Save Perspective as …* menu and save the perspective with a name of your own choice. If you use this custom perspective while activating/deactivating bundles, the Reset Perspective dialog won’t show up.

### Customized Perspectives

If you customize a perspective using the *Window | Customize Perspective …* menu entry, and afterwards resolves or unresolves a bundle that makes contributions to the main menu using the command extension point an unhandled event loop exception is raised when the menu manager tries to contribute the new menu/tool bar item to the main menu. This is the case in version 3.x but not in 4.2.

### Dynamic Menu contributions in Version 4.2

Lastly and for some unknown reason Eclipse 4.2 does not update the main menu and tool bar dynamically. This works in 3.8 and earlier versions (from version 3.0).

## Extensions and Extension Points

When working with dynamic plug-ins, plug-ins that implements extensions points should remove all references to classes and resources belonging to an extension contributor when the contributed extension is being unresolved. Runtime dependencies (e.g. referenced classes) are removed when the contributor is unresolved and the extension markup or definition is removed when the contributor is uninstalled.

The Eclipse stack is dynamic aware, but third party plug-ins may not be. Using non dynamic aware plug-ins cause stale references to classes in the contributing plug-ins when they are unresolved. [IExtensionTracker](http://help.eclipse.org/juno/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/core/runtime/dynamichelpers/package-summary.html?resultof=%22%49%45%78%74%65%6e%73%69%6f%6e%54%72%61%63%6b%65%72%22%20%22%69%65%78%74%65%6e%73%69%6f%6e%74%72%61%63%6b%22%20) is provided for implementing dynamic aware extension points and is also useful for tracking additions and removal of extensions in other contexts.

When dealing with non-dynamic extension points it is possible to dynamically add and remove an extension and its references from inside the contributing bundle by using [IExtensionRegistry](http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fcore%2Fruntime%2FIExtensionRegistry.html) and ExtensionRegistry. The ExtensionRegistry class is internal and relevant methods for adding and removing extension contributions is at the time of writing experimental and part of an interim API that is expected to change in the future.

See the [Start and Stop Bundles](Start%20and%20Stop%20Bundles.htm) section regarding extensions and the OSGi Stop command.

# Saving Resources before executing Bundle Operations

The “Save automatically before build” (Window | Preferences | General | Workspace) and “Save required dirty editors before launching” (Window | Preferences | Run/Debug | Launching) options are inspected before a bundle operation is executed.

“Save automatically before build” saves all dirty editors across all open projects. Hidden projects are not included. The “Save automatically before build” option is inspected first. If this option is off, the “Save required dirty editors before launching” is inspected.

It is recommended to switch the “Save automatically before build” on.