

P2 Installer

The P2 Installer is an Eclipse Rich Client Platform (RCP) application that can install products packaged in P2 repositories.

For additional information on P2, see:

- <http://www.eclipse.org/equinox/p2/>
- <https://wiki.eclipse.org/Equinox/p2>

The base installer supports:

- GUI, console, and silent operation
- Showing welcome, information, and license pages
- Choosing the install location
- Showing required and optional components for install along with disk usage
- Install P2 units
- Creating product short-cuts (and Windows add/remove)
- Extending the system PATH environment variable
- Installing Windows drivers
- Launching programs/files at the end of installation
- Uninstalling
- Packaging in a self-extracting binary

The installer can be extended with additional wizard pages and install actions.

Requirements

The P2 Installer requires:

- Eclipse SDK 3.8 or greater
- Oracle Java runtime environment (JRE) SE 7 or greater (Other JRE's are not supported)

Source

The latest source for the P2 installer can be obtained from the GitHub repository, <https://github.com/MentorEmbedded/p2-installer>. The P2 Installer is composed of one plug-in project, one feature project, and multiple platform specific fragment projects.

com.codesourcery.installer

- P2 Installer plug-in project

feature.com.codesourcery.installer

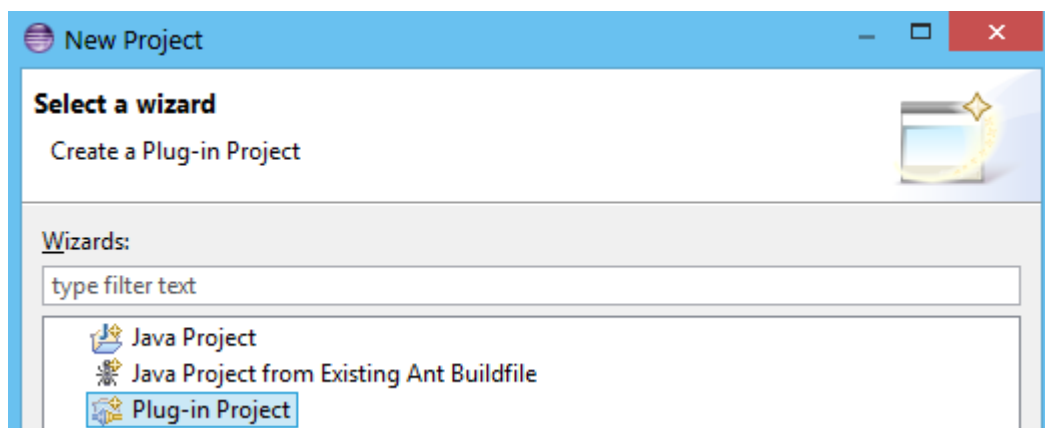
- P2 Installer feature project

The installer platform specific fragments contain a native utility (INSTMON) that is used by the installer to perform certain operations (like writing to the Windows registry). The *instmon* C/C++ Development Toolkit (CDT) project can be used to build these binaries. It contains project build configurations for different platforms. Make files in the *instmon/src* folder can also be used.

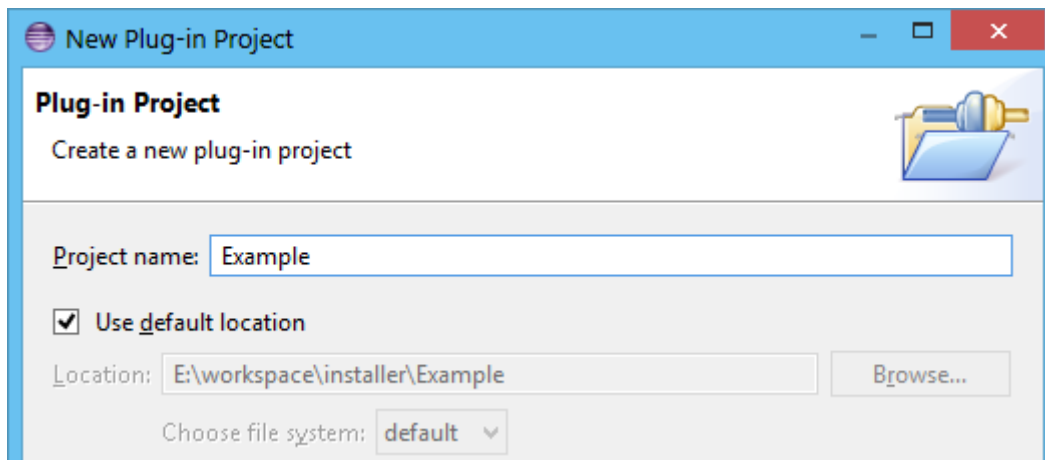
Getting Started

In order to start using the P2 Installer, import the source projects into a new Eclipse PDE workspace. Since we need P2 repositories for a product to install, we will create a simple RCP application for this purpose. You will want to use your product's P2 repositories when creating your own installer.

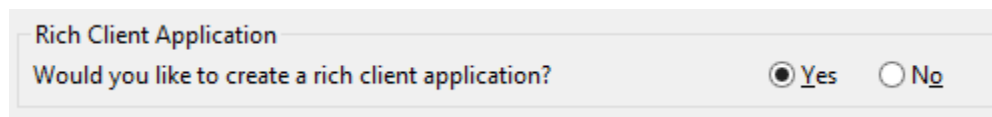
Select *File>New>Project*. Choose *Plug-in Project*. This will open the *New Plug-in Project* wizard.



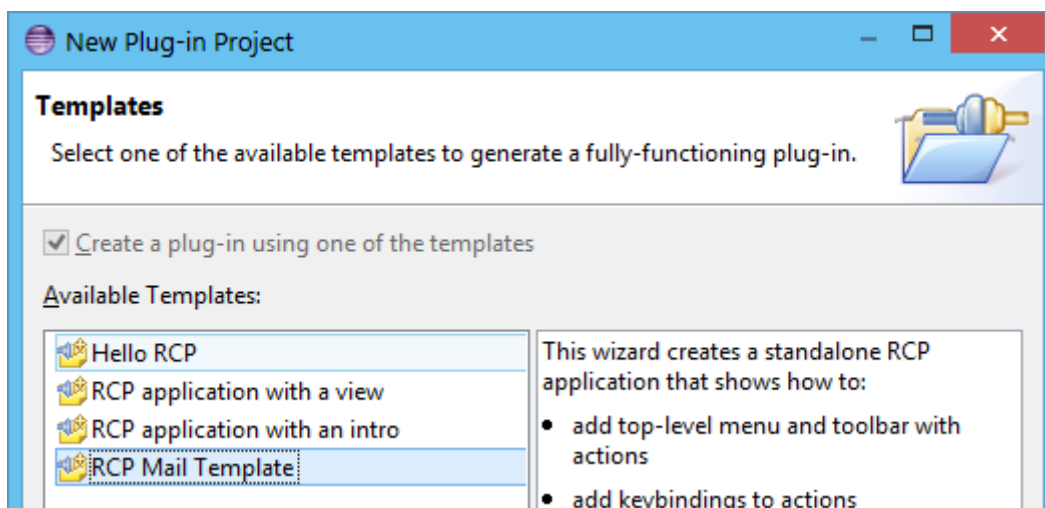
Enter the name, “Example”, for the project name and click *Next*.



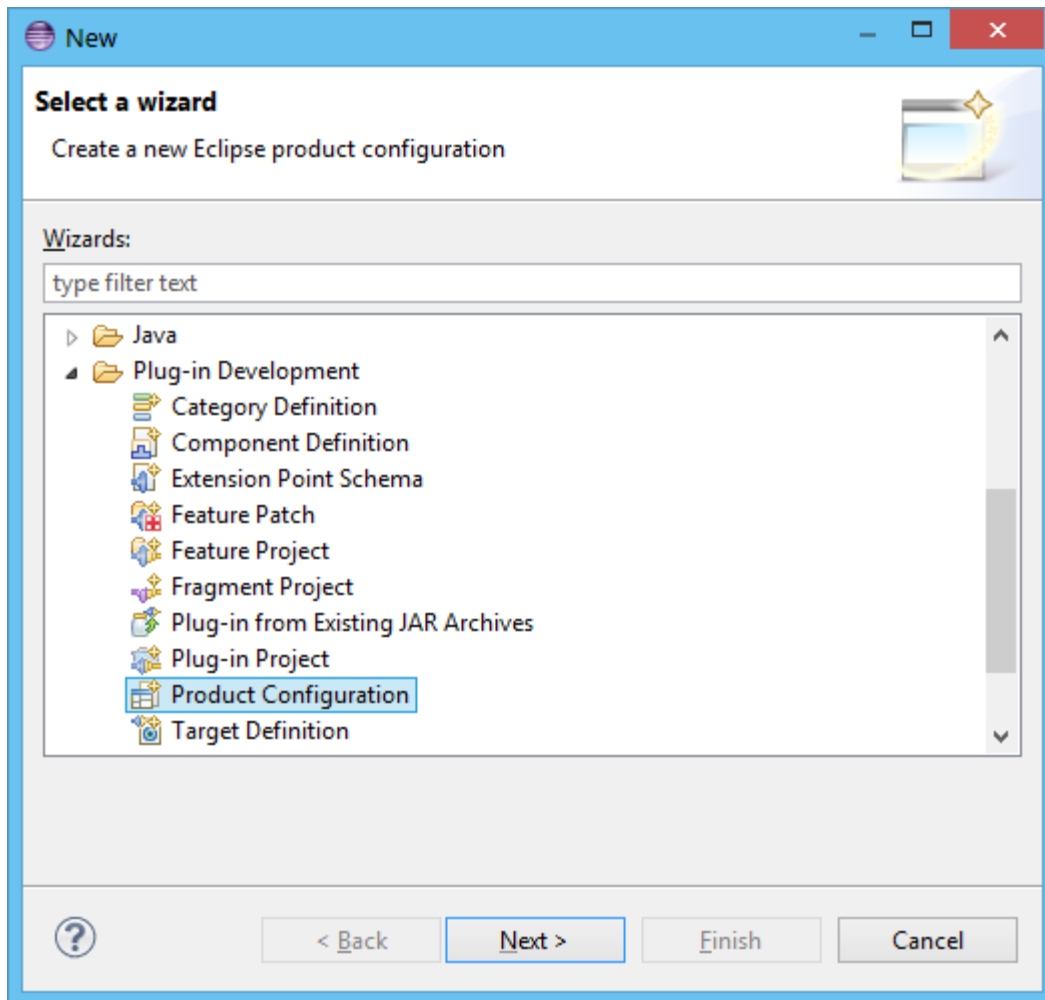
Make sure to select *Rich Client Application* on the *Content* page.



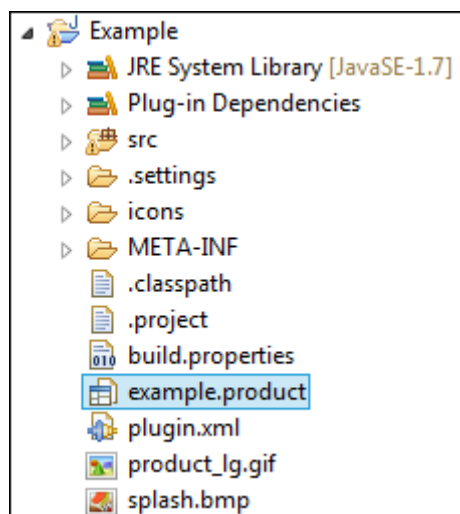
Choose the *RCP Mail Template* for our application and click *Finish*.



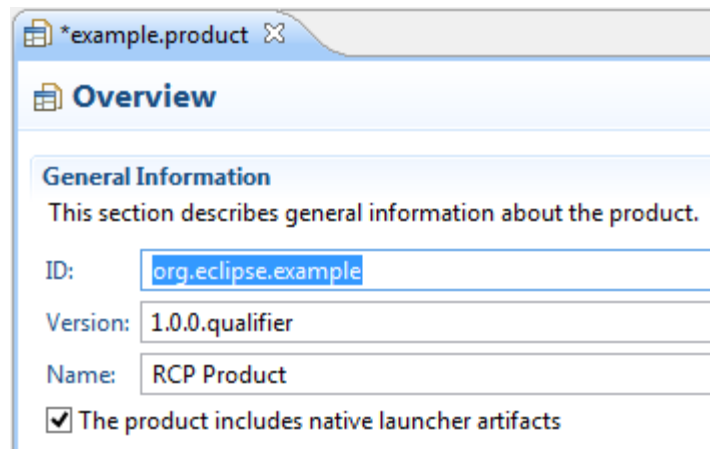
Next, we will want to create a *Product Configuration* for our application. Right-click on the *Example* project we just created and select *File>New>Other*. Choose *Plug-in Development>Product Configuration*.



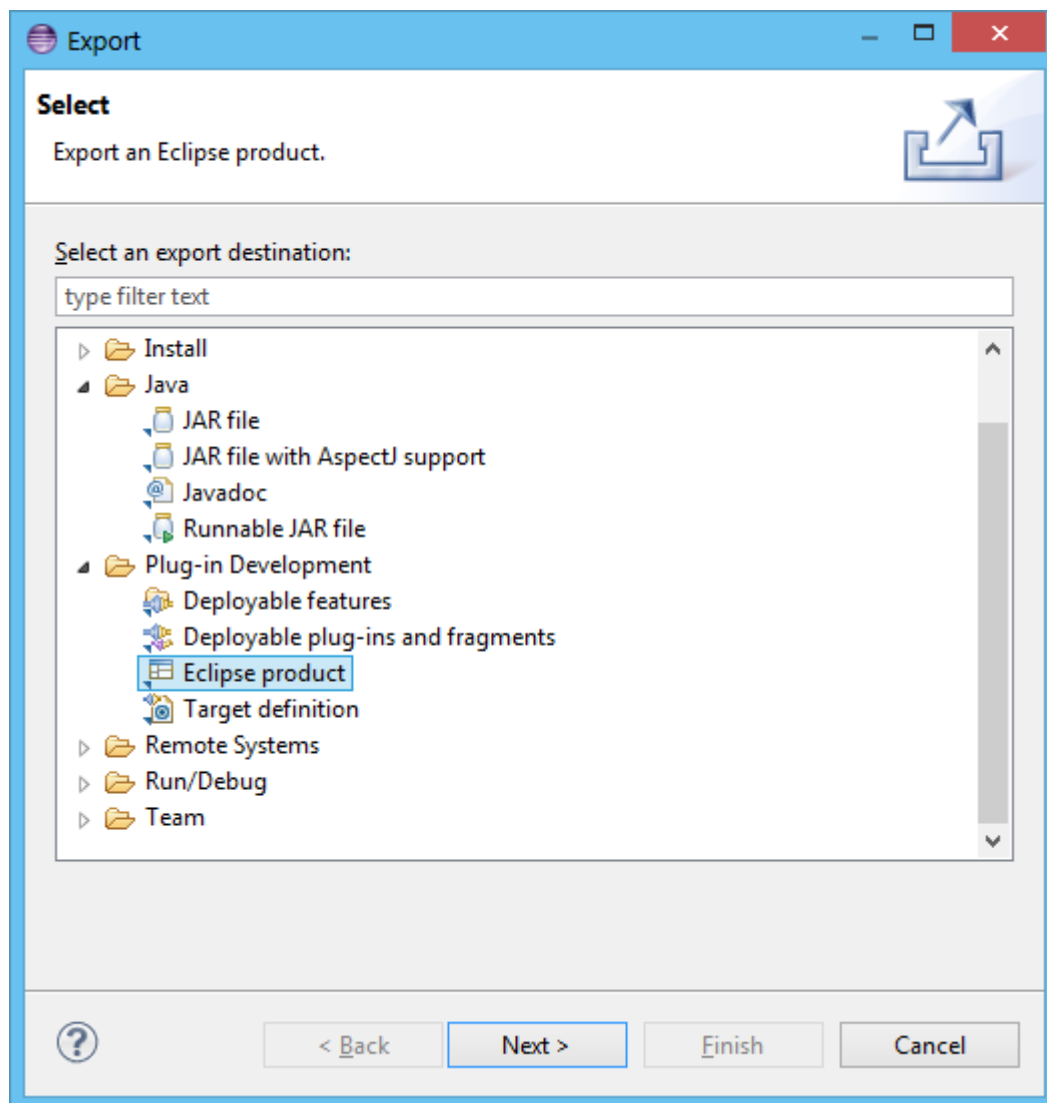
Enter “example.product” for the product filename.



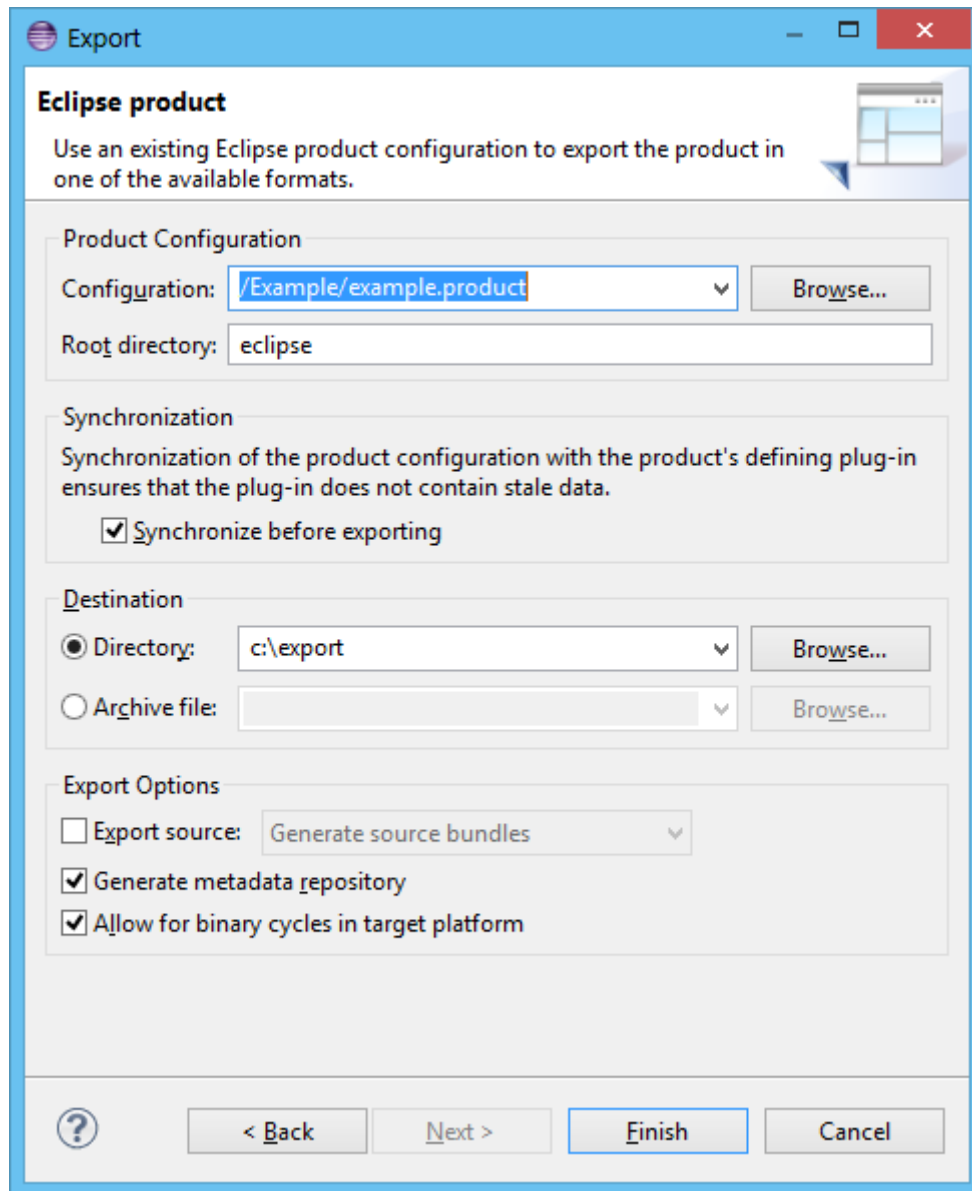
Open the product configuration file, enter “org.eclipse.example” for the *ID*, and save the file.



Now that we have created an application project, we will build, export it, and use the P2 repositories to install it. Select *File>Export* and choose *Plug-in Development>Eclipse product*.

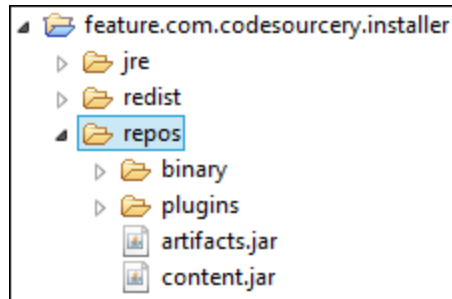


For *Configuration*, browse to the “example.product” product configuration file we created. Choose to export to a *Directory* and enter a path. Make sure the *Generate metadata repository* option is checked.



The screenshot shows the 'Export' dialog box in Eclipse. The title bar is blue with the Eclipse logo and the word 'Export'. The main area is white with a blue header bar containing the text 'Eclipse product' and a small icon of a product configuration file. Below the header, there is a section titled 'Use an existing Eclipse product configuration to export the product in one of the available formats.' The dialog is divided into several sections: 'Product Configuration' with a 'Configuration:' dropdown menu showing '/Example/example.product' and a 'Browse...' button, and a 'Root directory:' text field containing 'eclipse'. The 'Synchronization' section has a checkbox labeled 'Synchronize before exporting' which is checked. The 'Destination' section has two radio buttons: 'Directory:' (selected) and 'Archive file:'. The 'Directory:' option has a dropdown menu showing 'c:\export' and a 'Browse...' button. The 'Archive file:' option has a dropdown menu and a 'Browse...' button. The 'Export Options' section has three checkboxes: 'Export source:' (unchecked) with a dropdown menu showing 'Generate source bundles', 'Generate metadata repository' (checked), and 'Allow for binary cycles in target platform' (checked). At the bottom of the dialog, there is a row of buttons: a help button (question mark icon), '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

Now we have the P2 repositories for a simple application that we can use to install. Browse to the directory you exported the example product and copy the contents of the “repository” directory into the “repos” folder of the *feature.com.codesourcery.installer* project.



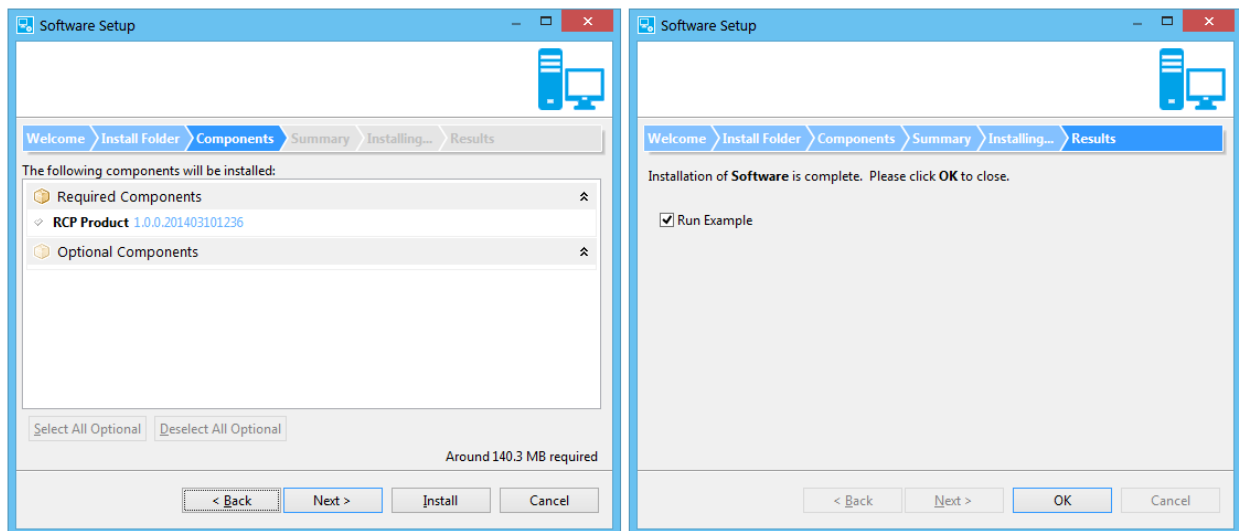
Edit the *installer.properties* file in the *feature.com.codesourcery.installer* project. Set the *eclipse.p2.requiredRoots* property to the identifier we used for our example RCP application.

```
eclipse.p2.requiredRoots=org.eclipse.example
```

Optionally, set the *eclipse.p2.launch* property to run our application after it has been installed.

```
eclipse.p2.launch=Run Example;eclipse/eclipse{exe};exe>true
```

Now run the installer using the **Installer – GUI** launch configuration.



Installer Properties

The P2 Installer is configured using a properties file. This file contains properties that specify what will be installed and the behavior of the installer. Normally, the installer looks for the *installer.properties* in the same directory as the installer launcher binary. If there is a localized

directory (for example, “*en_US*”), the installer properties will be loaded from that directory. You can also specify a different properties file to load by using the “-install.desc” command line switch.

The values of installer properties can be expressed in terms of other installer property values using the syntax, `${<property name>}`, where `<property name>` is the name of the property to substitute. For a complete list of installer properties along with their documentation, refer to the template *installer.properties* file located in the *feature.com.codesourcery.installer* project or refer to **Appendix 1**.

Exporting the P2 Installer

A stand-alone installer can be created by exporting the P2 Installer product. Before exporting, you should set up the *feature.com.codesourcery.installer* for your product.

1. Copy your products’ P2 repositories into the *repos* folder.
2. Edit the *installer.properties* file for your product installation.
3. It is advisable to copy the required JRE into the *jre* folder. This will ensure that the installer has a JRE to run and will not be dependent on the system JRE.

Select *File>Export* and choose the *Eclipse product* wizard. Choose the *feature.com.codesourcery.installer/installer.product* for the configuration to export.

The normal PDE export will produce a launcher executable to run the installer. This will be *setup* on Linux. It will be *setup.exe* and *setupc.exe* on Windows. On Linux, the *setup* binary is used to run the GUI, console, and silent installer. On Windows, *setup.exe* is used to run the GUI installer while *setupc.exe* is used to run the console and silent installers.

On Windows, the UAC will display a warning for any executable run that has **setup** or **install** in the name if it doesn’t include the proper embedded manifest. It will also attempt to run the executable using administrator privileges. For these reason, special binaries are exported from the *feature.com.codesourcery.installer/win32.win32.x86* folder and replace the ones normally generated by the PDE export. These binaries have been embedded with a manifest. The manifest can be found in:
feature.com.codesourcery.installer/win32/installer.manifest.

You can use the Windows SDK manifest tool (*mt.exe*) to embed the manifest into a new executable.

```
mt.exe -nologo -manifest "installer.manifest" -outputresource:"setup.exe;#1
```


Refer to the MSDN documentation on *mt.exe* for more information.

Running the P2 Installer

When running from the Eclipse workbench you can use the included launch configurations to run the P2 Installer.

Installer – GUI

- Runs the installer GUI wizard

Installer – Console

- Runs the installer in console mode

Installer – Silent

- Runs the installer in silent mode

Installer – Uninstall

- Runs the uninstaller. This assumes you have run the installer and have product(s) for uninstallation. The launch will prompt for an *install.manifest* file for the product to uninstall. This will be in the product installation *uninstall* directory.

Please note that when running from the Eclipse workbench, not uninstaller will be copied to the product installation *uninstall* directory.

An exported P2 Installer can be run using the *setup* executable. By default the GUI installer is run.

```
Windows
    setup.exe

Linux
    ./setup
```

To run the console installer, specify the “-nosplash -install.console” command line options.

```
Windows
    setupc.exe -nosplash -install.console

Linux
    ./setup -nosplash -install.console
```

To run the silent installer, specify the “-nosplash -install.silent” command line options. Note, the silent installer will use defaults for all options.

```
Windows
    setupc.exe -nosplash -install.silent

Linux
    ./setup -nosplash -install.silent
```

The command line options supported by the P2 Installer are:

-help

Prints usage and exits

-install.desc="URL of installer.properties"

Specifies the path to an install properties file to use.

Example:

-install.desc="file:/temp/installer.properties"

-install.manifest="Path to the install manifest file"

This option specifies the path of a product install manifest file to use for uninstallation.

Example:

-install.manifest="C:\users\user\product\uninstall\install.manifest"

-nosplash –install.console

This option runs the installer in console mode. If large amounts of text are output, like a license agreement, the console will pause after every 25th line. This limit can be changed by passing **=<limit>** where **<limit>** is the maximum number of lines to display before pausing.

Example:

-nosplash –install.console=120

-nosplash –install.silent

This option runs the installer in silent mode.

-install.D<property name>="<property value>"

Sets an installer property value where *<property name>* is the property to set and *<property value>* is the value to set. This will replace any property values loaded from the installer properties file.

-install.status=<file path>

This option creates a status file text file. The *<file path>* will be created after the installer completed and will contain either "OK", "CANCELED", OR "FAIL:" followed with an error message.

-install.data="<directory>"

This option specifies the installer data directory. If *<directory>* does not exist, it will be created. If this option is not specified, the *.p2_installer* directory in the users' home directory will be used.

Product Information

There are several installer properties that set information for the product to be installed. The unique identifier for the product is specified in the *eclipse.p2.productId* property. The unique P2 profile to use is specified in the *eclipse.p2.profileName* property. The version of the product is specified using the *eclipse.p2.productVersion* property. The installer will not allow a version of a product to be installed into a directory already containing the same version of that product. A name to display for the product can be specified in the *eclipse.p2.productName* property.

Install Location

The *eclipse.p2.rootLocation* property can be used to set the default install directory for a product. If this property uses “~”, the location will be based on the user’s operating specific home directory.

```
eclipse.p2.rootLocation=~/${eclipse.p2.productName}
```

P2 Repositories

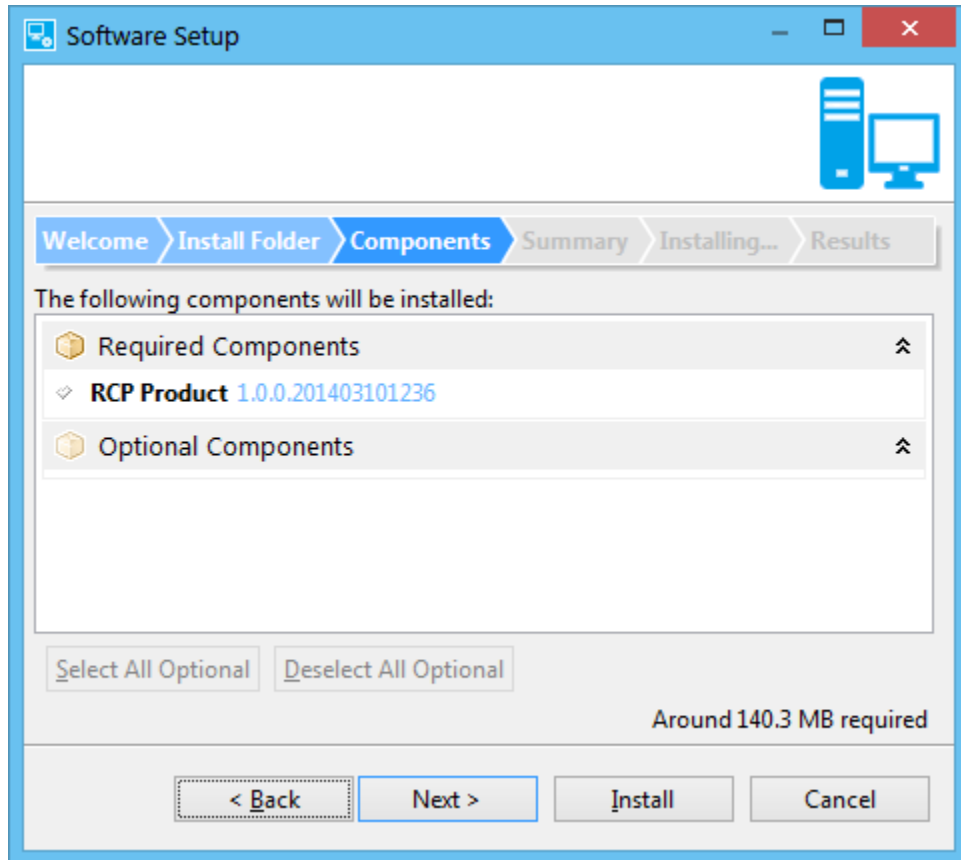
The P2 repositories containing the P2 units to be installed are listed in the *eclipse.p2.repos* property.

```
eclipse.p2.repos=file:./repos
```

By default, this property is set to the */repos* folder. This folder is located in the *feature.com.codesourcery.installer* project and is exported with the installer.

Alternatively, the *eclipse.p2.metadata* property can be used to specify P2 meta-data repositories and the *eclipse.p2.artifacts* property can be used to specify the P2 artifact repositories.

The P2 root installable units that will be installed for the product are specified in the *eclipse.p2.requiredRoots* and *eclipse.p2.optionalRoots*. Any units specified in the *eclipse.p2.optionalRoots* property can be selected by the user on the install wizard **Components** page. Optional roots that should be checked for install by default can be specified in the *eclipse.p2.optionalRootsDefault* property.

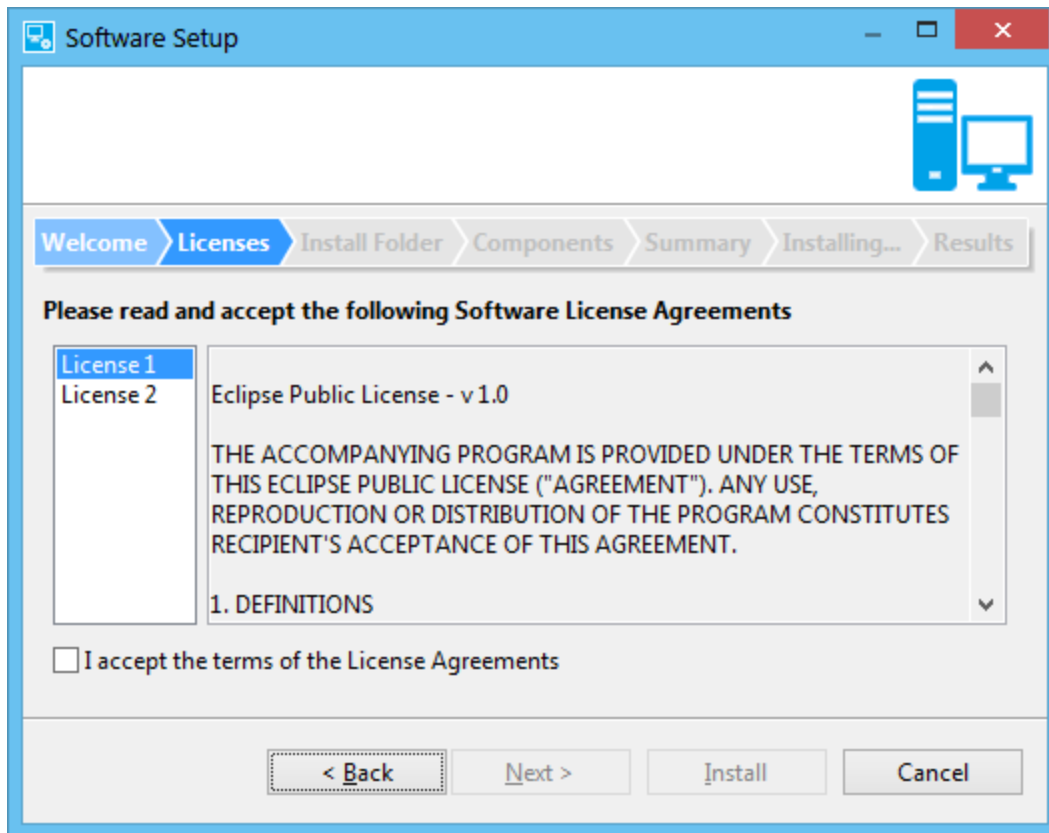


License Information

The P2 Installer wizard can display a page for the user to accept any required license agreements. License information can be included in files by using the *eclipse.p2.license* property.

```
eclipse.p2.license=license1.txt:License 1,license2.txt:License 2
```

The files must be included in the *build.properties* as root files in order to be available in the exported installer.



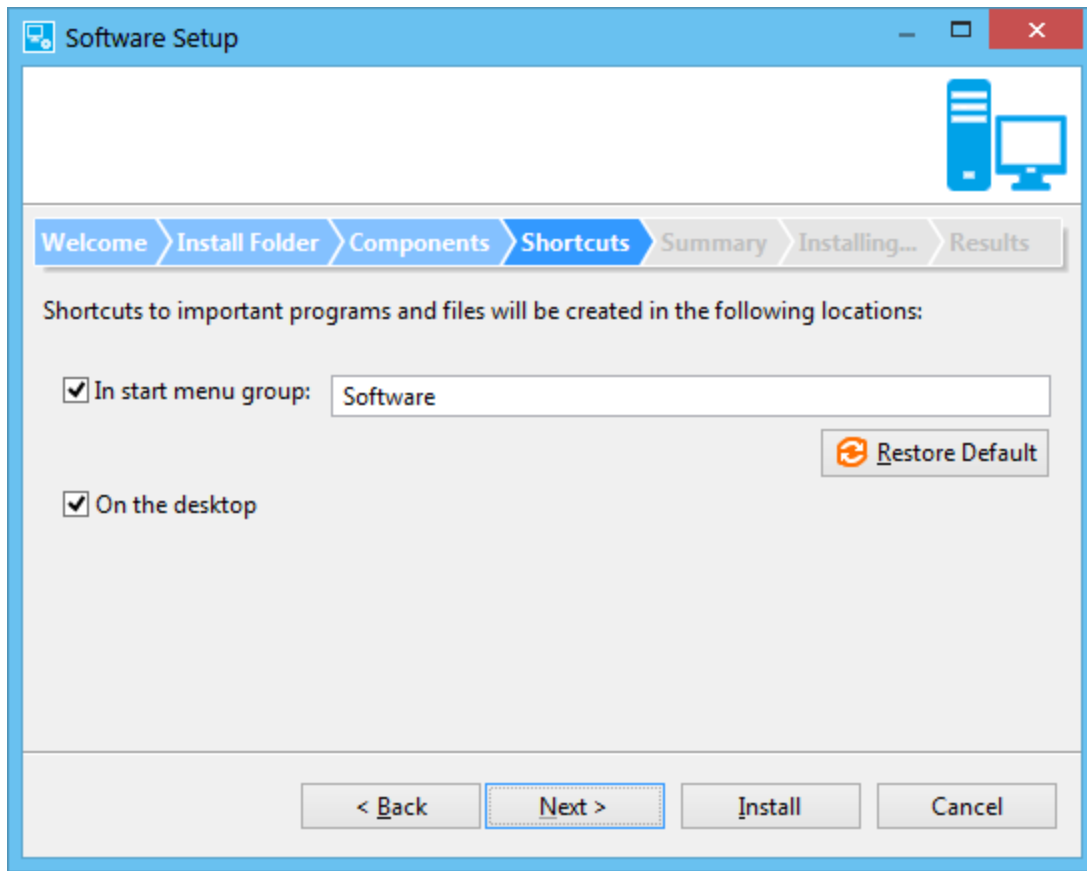
Licenses can also be displayed from the P2 root installable units by setting the *eclipse.p2.licenseIU* to *true*.

```
eclipse.p2.licenseIU=true
```

Product Shortcuts

The P2 Installer can create shortcuts to launch the product using the *eclipse.p2.links* property. Shortcuts can be created in the programs area, desktop, or launcher (Ubuntu Unity only) and can include arguments. On Linux, short-cuts are created using symbolic links. All *program* shortcuts are created relative to the folder specified in the *eclipse.p2.linksLocation* property. It is usually set to a directory based on the product name. On Windows, this folder will be created in the start menu. On Linux, this folder will be created in the user's home directory.

```
eclipse.p2.linksLocation=${eclipse.p2.productName}
eclipse.p2.links=programs;${eclipse.p2.productName};eclipse/eclipse{exe};;;
desktop;${eclipse.p2.productName};eclipse/eclipse{exe};;;
```



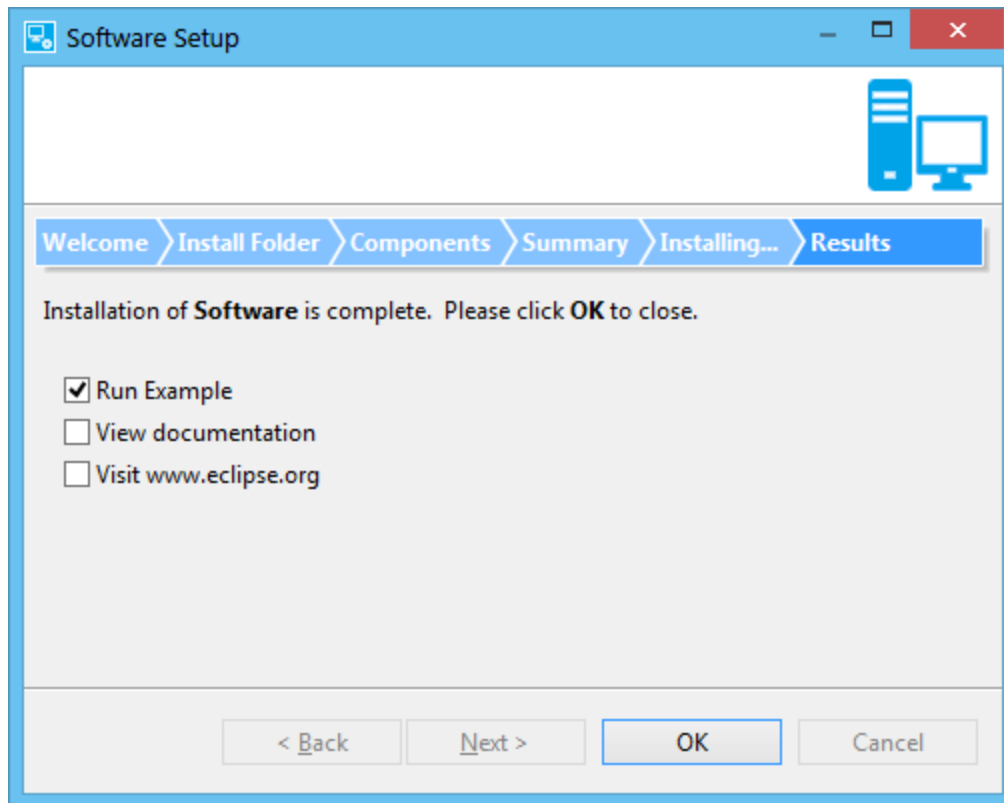
A shortcut can also be created to uninstall the product by pointing to the `uninstall/uninstall{exe}` target,

```
eclipse.p2.links=programs;;Uninstall  
${eclipse.p2.productName};uninstall/uninstall{exe};;
```

Product Launching

The P2 Installer can be set to launch the product at the end of a successful installation by using the `eclipse.p2.launch` property.

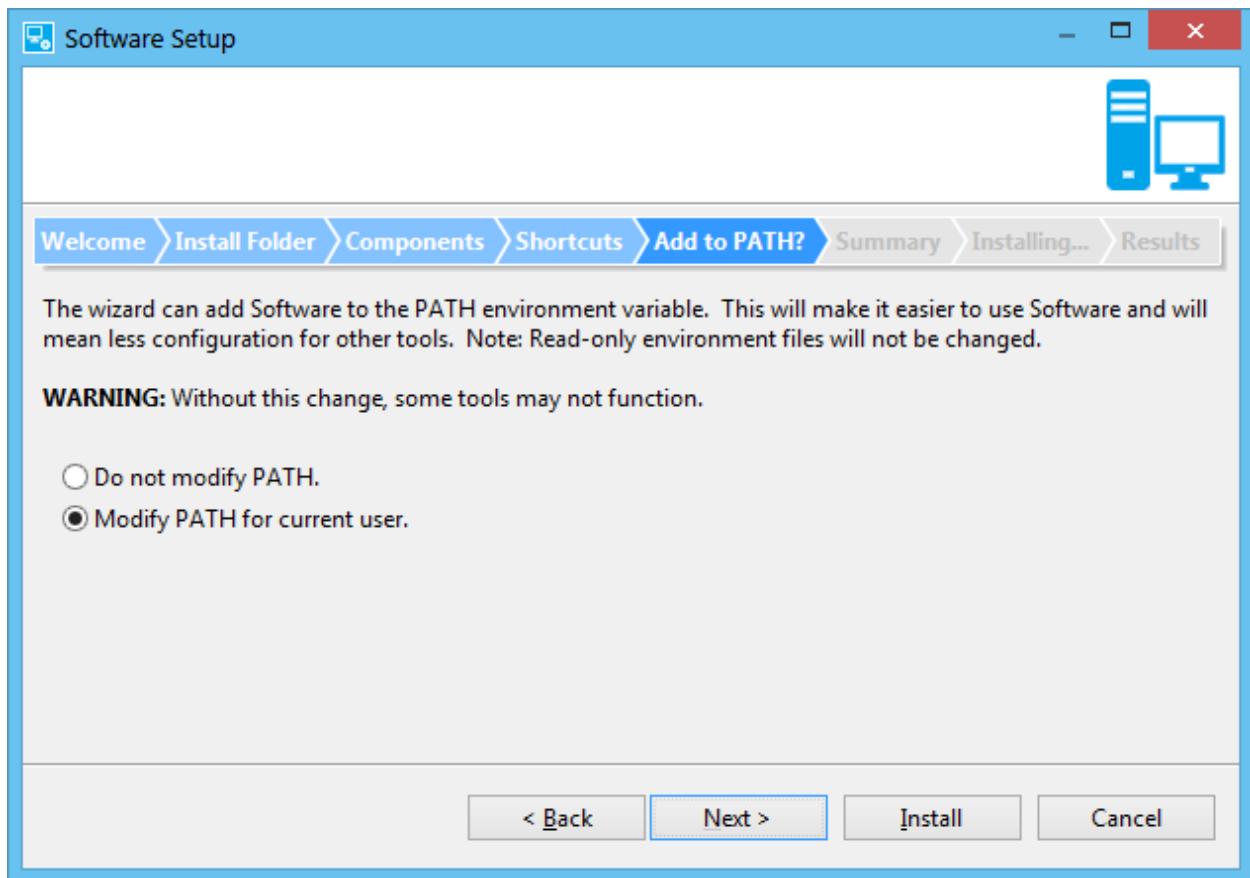
```
eclipse.p2.launch=Run Example;eclipse/eclipse{exe};exe>true,\  
View documentation;docs/doc.pdf;file;false,\  
Visit www.eclipse.org;http://www.eclipse.org/html;false
```



PATH Environment

Some products may require that their binaries be available on the system path. The *eclipse.p2.env.paths* property can be used to add product directories to the system PATH environment variable.

```
eclipse.p2.env.paths=bin
```



Uninstaller

The P2 Installer creates an *uninstall* directory after installation and writes a manifest (*install.manifest*) file in the directory that records information about what was installed. It will also copy the installer binaries to run as the uninstaller. The files copied are specified in the *eclipse.p2.uninstallFiles* property. Note, these files will only be available in an exported installer so running the uninstall launch configuration from a workbench will not be able to copy the files.

On Windows, the installer will also create information in the *Add/Remove Programs*.

Add-ons

The *eclipse.p2.addons* property can be used to specify additional remote P2 repositories containing add-on components for a product. The P2 Installer will query these repositories and include all root installable units found on the **Components** wizard page as optional.

Add-ons that provide additional features and functionality for **Software** can be downloaded and installed.

☐ Skip add-ons

☒ Install add-ons.

Username:

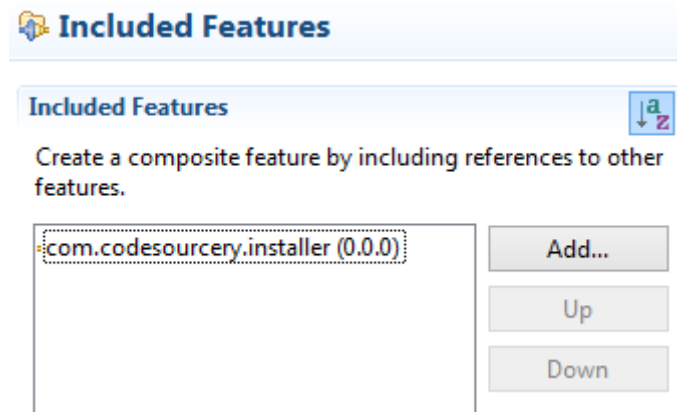
Password:

If the add-on sites require a log-in, set the `eclipse.p2.requiresLogin` property to **true**. Additional site credential handling can be done using the `com.codesourcery.installer.verifiers` extension point and `IInstallVerifier.verifyCredentials()` method. The description for a product's add-ons can be set using the `eclipse.p2.addons.description` property.

Creating a Custom Installer

Although the installer can be used by itself, you will probably want to create your own feature and plug-ins in order to add additional install wizard pages and custom install actions.

Create an additional feature and plug-in project for your installer. In the feature project, edit the `feature.xml` and add the installer `com.codesourcery.installer` feature.



If you will be including any files with the installer like information or license, add these as root files in the `build.properties`.

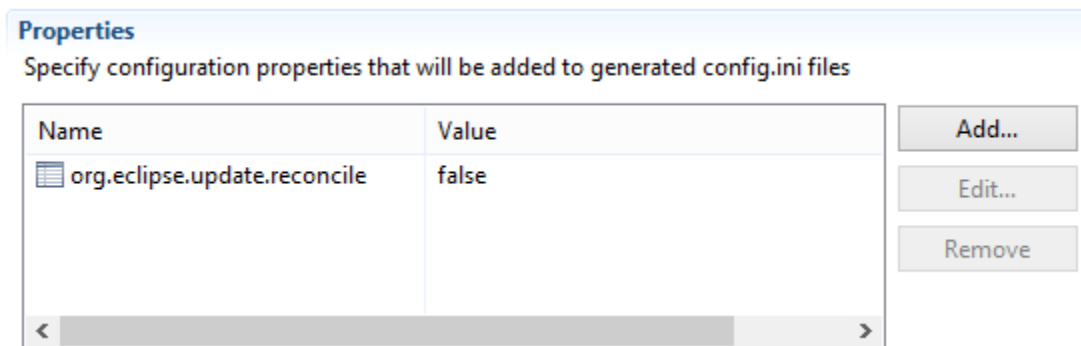
```
root=file:license.txt
root=file:information.txt
```

Add a project configuration to your feature project (`File>New>Other` and choose `Plug-in Development>Product configuration`). This will be used to export your installer application. Edit

the product file. Select the **Dependencies** tab and add your feature along with the *com.codesourcery.installer* feature. On the **Splash** tab, select your plug-in for the splash screen or you can use splash screen included with the *com.codesourcery.installer* plug-in.



In order to avoid “Bundle ... has already been installed” warnings in the Eclipse log file when running, the “*org.eclipse.update.reconcile=false*” property should be added in the **Configuration** tab.



See: <http://www.eclipse.org/forums/index.php/t/158631/> for more information.

Branding

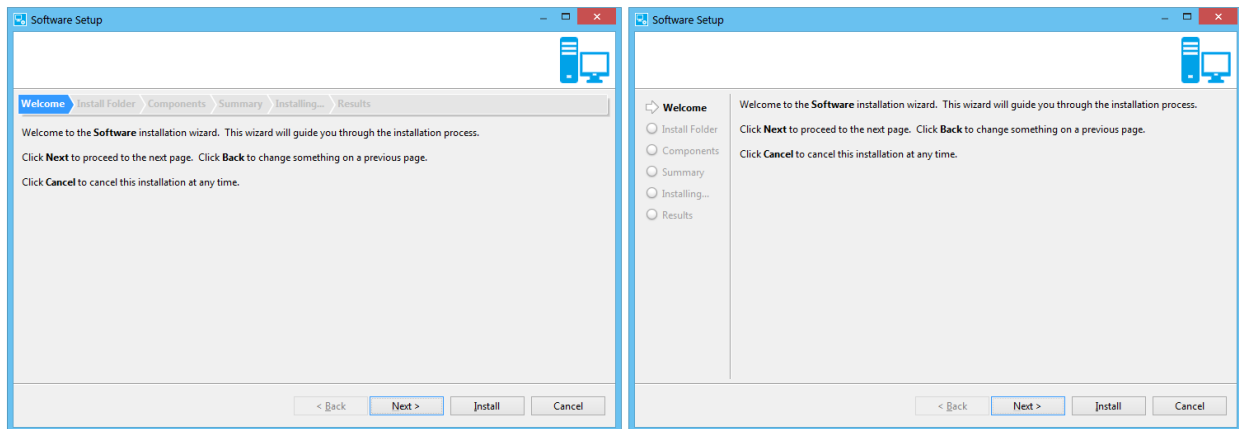
Branding for the P2 installer is done using a combination of installer properties and extension points. The title for the installer window can be set using the *eclipse.p2.windowTitle* property. A message can be displayed in the title area using the *eclipse.p2.title* property.

```
eclipse.p2.windowTitle=${eclipse.p2.productName} Setup  
eclipse.p2.title=Setup will install software for ${eclipse.p2.productName}.
```



The page navigation bar in the install wizard can be changed using the `org.eclipse.p2.wizardNavigation` property. The bar can be set to not show (“none”), show at the top (“top”), or show at the left side (“left”).

```
org.eclipse.p2.wizardNavigation=top
```



To change the splash screen, you can replace `splash.bmp` in the `com.codesourcery.installer` project, but it is recommended that you create your own feature and plug-in projects (see **Creating a Custom Installer**).

The `com.codesourcery.installer.icon` extension point can be used to replace the image used in the installer wizard title area. For example, you could replace it with a `logo.png` image located in your installer plug-in’s `icons` folder with the following declaration in the plug-in’s `plugin.xml`,

```
<extension point="com.codesourcery.installer.icon">
  <icon image="icons/logo.png" />
</extension>
```

Debugging an Exported P2 Installer

If you run the installer from the Eclipse workbench, debugging can be directly. In order to debug an installer that has been exported, do the following:

1. Add the following additional options after the `-vmargs` option in the `setup.ini` file located in the installer directory:

```
-Xdebug  
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8000
```

2. Create a new *Remote Java Application* launch configuration. Specify the `com.codesourcery.installer` project.

3. Run the exported installer and then run the launch configuration to attach and debug.

Extending the P2 Installer

The P2 Installer is composed of modules. These modules provide pages that are displayed in the install wizard along with actions that get performed during installation. The installer includes one module that provides the standard wizard pages and install actions.

A new install module is contributed using the `com.codesourcery.installer.modules` extension point. The Java class for a module must implement the `IInstallModule` interface or can extend `AbstractInstallModule`.

```
<extension  
  point="com.codesourcery.installer.modules">  
  <module  
    class="org.eclipse.example.InstallModule"  
    id="org.eclipse.example.InstallModule">  
  </module>  
</extension>
```

By default, the installer will load all registered modules. You can configure the installer to only use certain modules by listing them in the `eclipse.p2.modules` installer property.

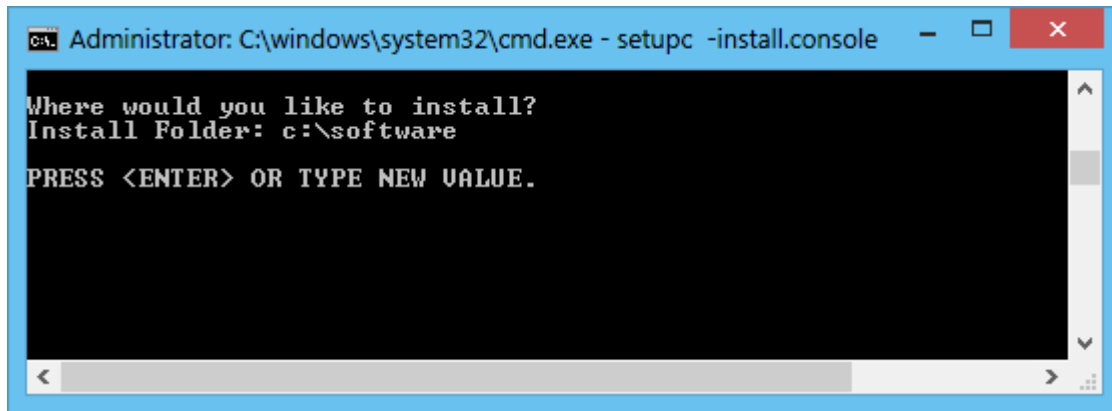
Install Wizard Pages

An install module contributes wizard pages through the method,

```
IInstallWizardPage[] getInstallPages()
```

The wizard pages must extend the `InstallWizardPage` class. A wizard page is passed an `IInstallData` to save its data and read data from other pages.

If a wizard page will be included when the installer is run in console mode, it should implement `IInstallConsoleProvider`. There are two classes that can aid in console presentation. The `ConsoleListPrompter` class can be used to display a list of items to select. The `ConsoleYesNoPrompter` can be used to display a standard yes/no prompt.



Install Actions

An install module can contribute actions to perform during installation through the method,

```
IInstallModule.getInstallActions(IProvisioningAgent, IInstallData)
```

This method is passed an *IInstallData* that can be used to read data from wizard pages. An action must implement the *IInstallAction* interface.

An action performs its operation when the *IInstallAction.run()* method is called. The method is passed an *IInstallMode* that specifies the installation mode. This can be installing, uninstalling, or upgrading.

The installer calls the *IInstallAction.save()* method to store the action data in the install manifest. When a product is uninstalled, the installer calls the *IInstallAction.load()* method to load the action data.

An action can call *Installer.getInstallPlatform()* to get an *IInstallPlatform* interface that has methods to perform certain platform specific operations (like writing/reading the Window registry, installing Windows drivers, etc.).

Packaging the P2 Installer (SFX)

After the installer has been exported, it can be packaged into small self-extracting binary, or a single executable file, which the user executes to launch the installer. The self-extracting binary is supported on Linux and Windows hosts. Support for SFX on Mac is not included at this time.

User Host Requirements

Linux

The Linux self-extraction script is based on a concept described in the Linux Journal by Jeff Parent (<http://www.linuxjournal.com/node/1005818>). The p2 installer SFX script itself can be found in the source:

```
sfx/scripts/src/linux/p2_sfx.h
```

This script will extract and launch the installer on any modern Linux distribution which includes standard core utilities such as tar with gzip compression (-z argument), df, readlink (-e argument), awk, tail, and others.

The extraction script has been tested using bash and dash shells.

Windows

The Windows self-extracting program will extract and launch the installer on any recent Windows version such as Windows 7 or 8. The SFX for Windows uses unzip.exe from <http://www.info-zip.org/>. By default, this is automatically bundled with SFX for Windows with no additional software needed to be installed by the user.

The algorithm used by SFX for Windows is a rather simplistic usage of the *incbin* assembly directive to link binary blobs to an executable and then extract them on the user's machine. It uses zip and unzip to create and extract the binary blobs. The source code for Windows SFX can be found here:

```
sfx/src/windows/p2_sfx.*
```

SFX Command Line Arguments

The following command line arguments are supported by sfx for the user on both Windows and Linux hosts:

- -silent
 - Performs a silent installation
- -console
 - Performs a text console based installation
- -location=[path]
 - Specifies the path for the installation

- `-noplash`
 - Suppresses the splash screen
- `-x [path]`
 - Extracts the installer to the path specified but does not start the installer.

All other arguments are passed directly to the installer.

By default, the extractor will extract the installer to the system temporary directory. A different directory can be used by setting its path in the `P2_INSTALLER_TEMP_PATH` environment variable.

The SFX will put any logs in the user's home location in a directory named `.p2_installer`. On Linux SFX typically won't add a log unless the JRE crashes on start up. On Windows however, SFX is a Microsoft GUI application and not a console application. Therefore it is not capable of printing any output (including help) to the command line. This means on Windows, all status and errors are printed directly to the log file which by default is named `p2_installer_extraction.log`. If the SFX fails on a Windows machine, check the log file.

Creating the SFX

The creation of the SFX binary for both Linux and Windows must be performed on a Linux host and requires a bash shell. In addition, the creation of the Windows SFX requires the Minimalist GNU tools for Windows to be installed.

The scripts that can be used to build the SFX executable binary can be found here:

sfx/scripts/build

Building SFX

The script that can be used to build the Linux SFX executable is

sfx/scripts/build/build_linux_p2_sfx.sh.

The script that can be used to build the Windows SFX executable is

sfx/scripts/build/build_windows_p2_sfx.sh.

To brand SFX for your product, you can modify these scripts directly or use them as a template.

The arguments for the SFX build scripts are as follows:

`--installer-dir <directory where the exported installer to be packaged by SFX can be found. This argument is required>`

--scratch-dir <directory where temporary objects and files can be written during SFX generation. This argument is optional. The script will use the current directory if not specified>
--output-file <path and file name of generated SFX binary. This argument is required>
--silent <Turn off messages from this script. This argument is optional>

The Linux extraction script (p2_sfx.h) contains the following tags:

```
#@PREPROCESS@  
  
and  
  
#@POSTPROCESS@.
```

These tags can be replaced with any script code you deem necessary by writing your own SED scripts.

The windows extraction program behavior and splash screen may be changed by modifying the code in the functions found in build_windows_p2_sfx.sh.

Windows SFX build

The script that can be used to build the Windows SFX executable is *sfx/scripts/build/build_windows_p2_sfx.sh*. You can modify this file directly or use it as a template. It takes similar input arguments as the build script for Linux SFX, but it also adds the following:

--compiler <path to mingw compiler. Required>
--resource-compiler <path to mingw resource compiler. Required>

SFX JRE Requirement

The SFX for Linux and Windows requires a JRE to be present in the following directory before the installer will successfully launch on the user's computer:

```
$installer_dir/jre/bin
```

The easiest way to accomplish this is to add a JRE to the installer exported by eclipse prior to building SFX.

Repositories

SFX requires that all P2 repositories for the product be in zip files. Directory repositories are not supported.

Windows Firewall

On Windows, P2 can sometimes trigger a firewall warning during installation. This is due to the *BackupStore* class in the *org.eclipse.equinox.p2.touchpoint.natives* bundle. The *genUnique()* method uses *InetAddress* methods to generate a unique string. This can be avoided by patching the *org.eclipse.equinox.p2.touchpoint.natives* plug-in and changing the *genUnique()* method to use Java 1.6 UUID. The *genUnique()* method implementation can be replaced with the following:

```
private String genUnique() {
    long timePart = (System.currentTimeMillis() << 5) | (msCounter++ & 31);
    return Long.toHexString(timePart) + "_" + UUID.randomUUID().toString();
}
```

Signing

You may want to digitally sign all of your installer executables. This can possibly avoid Antivirus software reporting the binaries as suspicious. You can check various Antivirus reports against files using a tool like: <https://www.virustotal.com/>

Appendix 1: Installer Properties Template

```
#####
##
# Copyright (c) 2014 Mentor Graphics and others.
# All rights reserved. This program and the accompanying materials
# are made available under the terms of the Eclipse Public License v1.0
# which accompanies this distribution, and is available at
# http://www.eclipse.org/legal/epl-v10.html
#
# Contributors:
#   Mentor Graphics - initial API and implementation
#####
##

#####
##
# This file specifies properties used by the installer. The file is first
# searched for in a locale directory in the installer directory. If a locale
# directory is not found, it will be looked for in the installer directory.
# Property values can be expressed in terms of other property values using
the
# following syntax:
#   ${<property name>}
# where
#   <property name> is the name of the other property
#####
##

#####
##
```

```

# P2 Repositories (Required)
#   Specify the URL for P2 repositories.  Separate multiple repositories with
#   a comma.
#   By default, the installer expects repositories to be located in the
#   /repos
#   folder of the feature.com.codesourcery.installer feature.
#
#   To specify an archived repository use the following syntax,
#   "jar:file:<full or relative path to archive file>!/\"
#   For Example:
#   jar:file:./repos/repo-file.zip!/
#####
##
eclipse.p2.repos=file:./repos

#####
##
# P2 Meta-data Repositories (Required)
#   Specify the URL for the P2 meta-data repositories.  Separate multiple
#   repositories with a comma.
#####
##
eclipse.p2.metadata=${eclipse.p2.repos}

#####
##
# P2 Artifact Repositories (Required)
#   Specify the URL for the P2 artifact repositories.  Separate multiple
#   repositories with a comma.
#####
##
eclipse.p2.artifacts=${eclipse.p2.repos}

#####
##
# Add-ons (Optional)
#   Optional add-ons sites.  This property can specify the URL of
#   repositories
#   containing add-ons that can be selected to install.  Separate multiple
#   repositories with comma.
#   If this property is specified, the install wizard will display a page
#   that
#   gives the user the option to install additional components.  If add-ons
#   are
#   selected, the installer will query all root installable units from the
#   available repositories.
#   If this property is specified then the "eclipse.p2.addons.description"
#   property must also be specified.
#####
##
#eclipse.p2.addons=

#####
##
# Add-ons Login (Optional)
#   Specify if the add-on sites require a login.  This property is only
#   applicable

```

```

#   if the "eclipse.p2.addons" property is specified.  If this property is
set to
#   "true", the installer will prompt for a log-in user name and password.
#####
##
eclipse.p2.addons.requiresLogin=false

#####
##
# Add-ons Description (Optional)
#   Optional description to display for add-ons.
#   This property is only applicable if the "eclipse.p2.addons" property is
#   specified.
#   Text can be made bold using the <b> tag.  For example,
#       "This is <b>bold</b> text."
#   Text can be made italic using the <i> tag.  For example,
#       "This is <b>italic</b> text."
#####
##
eclipse.p2.addons.description=Add-ons that provide additional features and
functionality for <b>${eclipse.p2.productName}</b> can be downloaded and
installed.

#####
##
# Update Site (Optional)
#   Optional update sites for the product.
#   Each update site specified will be added and available in the 'Install
New
Software' dialog.
#   Separate multiple update sites with a comma.  An optional name for the
update
#   site can be specified using a semicolon.
#       <update site 1 URL>;<update site 1 name>;<update site 2 URL>;<update
site 2
#       name>;...
#   Example:
#       http://www.mysite1.com;Update Site 1,http://www.mysite2.com;Update Site
2
#####
##
#eclipse.p2.update=

#####
##
# Installer Title (Optional)
#   Optional property to specify the title area text that is displayed for all
#   wizard pages in the installer.  If this property is not specified, no title
#   will be displayed.
#####
##
#eclipse.p2.title=Software Setup

#####
##
# Window Title (Optional)
#   Specify the window title of the installer.  If this property is not

```

```

# specified, the product name will be used.
#####
##
eclipse.p2.windowTitle=${eclipse.p2.productName} Setup

#####
##
# Product Information (Optional)
# Optional file containing additional information to display for the
product.
# If this property is specified, the installer will display an information
page
# with the contents of the file. The path to the file is relative to the
# installer properties file.
# Text in the file can be made bold using the <b> tag. For example,
# "This is <b>bold</b> text."
# Text in the file can be made italic using the <i> tag. For example,
# "This is <i>italic</i> text."
#####
##
#eclipse.p2.information=./information.txt

#####
##
# Product Install Location (Required)
# Specifies the default root install location. Use '~' to specify a
location
# based on the operating system specific home directory.
#
# Note: Do not use this property in the values for other properties as it
can
# be changed by the user during installation.
#####
##
eclipse.p2.rootLocation=~/${eclipse.p2.productName}

#####
##
# P2 Install Location (Required)
# Specifies the location for the P2 (i.e. Eclipse) installation directory
# relative to the product location (eclipse.p2.rootLocation).
#####
##
eclipse.p2.installLocation=eclipse

#####
##
# Uninstaller (Optional)
# This property specifies the installer files/directories to copy into the
# /uninstall directory. If this property is not specified, the uninstaller
# will not be copied to the product installation directory. Also, no
# add/remove entry will be created on Windows host.
#
# If the file is an executable binary, append {exe} that will be replaced
with
# ".exe" on Windows and nothing on Linux.

```

```
#####
##
eclipse.p2.uninstallFiles=setup{exe}:uninstall{exe},setup.ini:uninstall.ini,configuration,features,jre,mon,plugins,.eclipseproduct

#####
##
# P2 Flavor (Required)
#   As part of a product build, PDE/Build automatically generates default
#   configuration meta-data to set start levels and config.ini property. This
#   meta-data is commonly referred to as Configuration Units (CUs). In
#   particular, start levels are set using CU fragments on the IU for the
bundle
#   being started. The flavor is used as a qualifier when generating the CU's
#   name based on the IU.  For example, with "p2.flavor = tooling",
#   'toolingwin32.win32.x86org.eclipse.core.runtime' will be the name of the
CU
#   that configures the org.eclipse.core.runtime bundle on windows. It may be
a
#   good idea to use a flavor based on your product id to avoid conflicts
with
#   other meta-data, particularly if your product has particular needs with
#   respect to start levels.
#####
##
eclipse.p2.flavor=tooling

#####
##
# Product Identifier (Required)
#   Specifies a unique identifier for the product
#####
##
eclipse.p2.productId=\${eclipse.p2.productName}

#####
##
# Product Name (Required)
#   Specifies a name for the product.
#####
##
eclipse.p2.productName=Software

#####
##
# Product Vendor Name (Optional)
#   Specifies the vendor name for the product.  This will appear in the
#   add/remove entry on Windows host.
#####
##
#eclipse.p2.productVendor=Eclipse.org

#####
##
# Product Version (Required)
#   Specifies the version of the product
```

```
#####
##
eclipse.p2.productVersion=1.0.0

#####
##
# Product Help URL (Optional)
# Specifies the URL for product help. This will appear in the add/remove
# entry on Window host.
#####
##
#eclipse.p2.productHelp=http://eclipse.org/

#####
##
# P2 Profile (Required)
# Specifies the P2 Profile to use for installation.
#####
##
eclipse.p2.profileName=SGXXProfile

#####
##
# Remove Profile IUs (Optional)
# This property specifies if all installable units included in the profile
# should be removed when the product is uninstalled or upgraded. This will
# include any installable units that are installed into the profile after
the
# initial installation (add-ons).
# If this property is set to "true" all IUs in the profile will be removed.
# If this property is set to "false" only the root IUs recorded during
# installation will be removed.
#####
##
eclipse.p2.removeProfile=true

#####
##
# Launch (Optional)
# Specifies items to launch after installation is complete. If this
property
# is specified, the installer will display a page to choose items to launch
# after installation.
# Separate multiple items with a comma.
# The format for an item is:
# <name>;<path>;<type>;<default>
# where
# <name> is the name to display
# <path> is the path to the file that should be launched. This path can
be
# a file relative to the product installation director
# (eclipse.p2.rootLocation), or an HTML address.
# If the item is an executable binary, append {exe} that will be
replaced
# with ".exe" on Windows and nothing on Linux.
# <type> is
# exe - Executable to run
```

```

#     file - File to open
#     html - Web page to open in the browser
#     <default> is "true" to perform or "false" to not perform by default.
#
# Example:
#     Visit Eclipse.org?;http://www.eclipse.org/;html;true
#
#####
##
#eclipse.p2.launch=

#####
##
# Short-cuts (Optional)
#     Specifies short-cuts/links to create.
#     If this property is specified, the installer will display a page to
choose
#     short-cuts.
#
#     Separate multiple short-cuts with a comma.
#     The format for links is:
#     <folder>;<link path>;<link name>;<link target>;<icon path>;<arguments>
#     where,
#     <folder> is
#     programs - Programs folder. This folder is relative to the
#     eclipse.p2.linksLocation folder.
#     desktop - The desktop folder
#     launcher - The Unity launcher panel (Ubuntu Linux only)
#     <link path> - The folder/directory to create the short-cut in. This
path
#     is relative to the <folder> location and can be omitted.
#     <link name> - The name for the short-cut.
#     <link target> - The path to the target file for the short-cut relative
#     to the root install location. If the link target is an executable
#     binary,
#     append {exe} that will be replaced with ".exe" on Windows and nothing
on
#     Linux.
#     <icon path> - The path to the icon that will be used to display the
link.
#     <arguments> - A comma-separated list of command-line arguments that
will
#     be passed to the executable when the link is activated.
#
# Example:
#     programs;Mentor Graphics/CodeBench#CodeBench;bin/eclipse{exe};;;
#     desktop;;CodeBench;bin/eclipse{exe};bin/icon.png;;
#
#     Created Short-cut directories will be removed on uninstallation if they
are
#     empty.
#####
##
#eclipse.p2.links=

#####
##

```

```

# Short-cuts Location (Optional)
#   Specifies the folder program short-cuts/links. The folder is relative to
#   the user's home directory on Linux and Start Menu on Windows.
#   This property is required if the "eclipse.p2.links" property is set.
#####
##
#eclipse.p2.linksLocation=${eclipse.p2.productName}

#####
##
# Short-cut Defaults (Optional)
#   Specifies the short-cuts that will be selected by default. Separate
#   short-cut folders to be selected by a comma.
#   If this property is not specified, all short-cuts will be created by
#   default.
#   This property is only applicable if the "eclipse.p2.links" property is
#   set.
#
#   Example:
#       eclipse.p2.links=desktop,programs
#####
##
#eclipse.p2.linksDefault=desktop,programs,launcher

#####
##
# PATH (Optional)
#   Specifies paths to append to the system PATH environment variable. The
#   paths are relative to the product installation directory
#   (eclipse.p2.rootLocation). Separate multiple paths with a comma.
#   If this property is specified, the install wizard will display a page to
#   choose if the path should be modified.
#
#   Example:
#       eclipse.p2.env.paths=bin
#####
##
#eclipse.p2.env.paths=

#####
##
# Required Root Installable Units (Required)
#   Specifies the identifiers of the root installable units (IU) that will be
#   installed. IU's that are specified will be listed on the Components page
#   of
#   the install wizard under the "Required" section.
#   Any IU's listed must be present in the P2 repositories specified in the
#   "eclipse.p2.repos" property.
#   Separate multiple IU's with a comma.
#
#   Example:
#       eclipse.p2.requiredRoot=org.eclipse.example,org.eclipse.extras
#####
##
#eclipse.p2.requiredRoots=

```



```
#####
##
# Optional Root Installable Units (Optional)
# Specifies the identifiers of the root installable units (IU) that can be
# optionally installed. IU's that are specified will be listed on the
# Components page of the install wizard under the "Optional" section and
# can
# be selected for install.
# Any IU's listed must be present in the P2 repositories specified in the
# "eclipse.p2.repos" property.
# Separate multiple IU's with a comma.
#####
##
#eclipse.p2.optionalRoots=

#####
##
# Default Optional Root Installable Units (Optional)
# Specifies the optional root installable units that should be selected by
# default. Any IU's listed will initially be checked on the install wizard
# Components page.
# Separate multiple IU's with a comma.
# Note: Any IU's found in an existing installation that is being upgraded,
# will be selected for install automatically regardless if they are not
# listed
# in this property.
#####
##
#eclipse.p2.optionalRootsDefault=

#####
##
# Component Sorting (Optional)
# The following optional properties can be specified to sort components by
# name on the install wizard components page. By default components will
# appear in the order that they are listed in the "eclipse.p2.roots"
# property.
#####
##
#eclipse.p2.sortRoots=false
#eclipse.p2.sortOptionalRoots=true

#####
##
# License Agreement (Optional)
# Specifies files containing license agreements for the product. If this
# property is specified, the installer will display a page for the user to
# accept the license agreements.
# Multiple files should be separated with a comma. An optional name to
# display for the license agreement can be entered separated with a ':'.
# The format is:
# <path to license file 1>:<optional name>,
# <path to license file 2>:<optional name>,
# ...
# <path to license file 3>:<optional name>
#
# Example:
```

```

# eclipse.p2.license=./license.txt
# eclipse.p2.license=./license1.txt, ./license2.txt
# eclipse.p2.license=./license1.txt:License Agreement A,\
# ./license2.txt:License Agreement B
#
# The file paths are relative to the installer.properties file.
# If no name is provided for the license agreements, the licenses will be
# named "License Agreement 1", "License Agreement 2", etc.
#
# To include a license file, place it in the
# feature.com.codesourcery.installer project and modify the
build.properties
# to include it as an exported root file.
# For example, if the license file is license.txt, the following would need
# to be added in the build.properties,
#     root=file:installer.properties
# The property would be set to:
#     eclipse.p2.license=./license.txt
#####
##
#eclipse.p2.license=

#####
##
# License IUs (Optional)
# Specifies if license information from selected component IU's should be
# displayed on the license page. If this property is set to "true", license
# information found in the IU's will be displayed.
#####
##
#eclipse.p2.licenseIU=true

#####
##
# Wizard Navigation (Optional)
# Specifies the type of wizard page navigation bar. This property can be
one
# of the following values:
# none - Don't show a page navigation bar
# top - Show a page navigation bar at the top of the wizard
# left - Show a page navigation bar on the left of the wizard
#####
##
org.eclipse.p2.wizardNavigation=top

#####
##
# Page Order (Optional)
# Specifies the order that install wizard pages should be displayed.
Separate
# page names with a comma. The pages will be displayed in the order that
they
# are listed in this property. Any page not listed will be displayed after
the
# pages specified. Any page listed that does not exist will be skipped.
# The names of common pages are:
#     welcomePage, licensePage, informationPage, installFolderPage,

```

```

# componentsPage, shortcutsPage, and pathsPage.
# Note: Some wizards pages are designed to be shown after certain other
pages.
#####
##
#eclipse.p2.wizardPages=

#####
##
# P2 Progress Filter (Optional)
# Specifies a set of regular expressions to filter the progress messages
# reported from P2 during installation. Multiple properties can be
specified
# and the first match will be applied.
#
# The format of the property names is:
# eclipse.p2.progressFind0
# eclipse.p2.progressFind1
# ...
# eclipse.p2.progressFindn
#
# For each eclipse.p2.progressFind property, there should be a
corresponding
# eclipse.p2.progressReplace property that specifies the expression used to
# replace the progress message. The eclipse.p2.progressFind expressions
can
# contain groups that are referenced by the eclipse.p2.progressReplace
# expressions.
#
# Example:
# eclipse.p2.progressFind0=\
# (.*?) (([A-Za-z0-9]+\)\.\.([A-Za-z0-9]+\)\.\.([A-Za-z0-9]+\)\.)(.*?)
#####
##
eclipse.p2.progressFind0=(.*?) (([A-Za-z0-9]+\)\.\.([A-Za-z0-9]+\)\.\.([A-
Za-z0-9]+\).jar)(.*))
eclipse.p2.progressFind1=(.*?) (([A-Za-z0-9]+\)\.\.([A-Za-z0-9]+\)\.\.([A-
Za-z0-9]+\)))(.*?)

#####
##
# P2 Progress Filter Replacement (Optional)
# Specifies a set of regular expressions to replace the progress messages
# from P2.
#
# The format of the property names is:
# eclipse.p2.progressReplace0
# eclipse.p2.progressReplace1
# ...
# eclipse.p2.progressReplacen
#
# For each "eclipse.p2.progressFind" property, there should be a
corresponding
# "eclipse.p2.progressReplace" property that specifies the expression used
to
# replace the progress message. The "eclipse.p2.progressFind" expressions
can

```

```

#   contain groups that are referenced by the eclipse.p2.progressReplace
#   expressions.
#
#   Example:
#       eclipse.p2.progressReplace0=Installing $2
#####
##
eclipse.p2.progressReplace0=Reading $2
eclipse.p2.progressReplace1=Installing $2

#####
##
# P2 Properties (Optional)
#   Specifies additional properties used by P2.
#   The property, "org.eclipse.update.install.features", set to "true" will
#   cause the P2 update manager features to be installed.
#####
##
org.eclipse.update.install.features=true

#####
##
# Installer Modules (Optional)
#   The installer can be extended with modules that implement the
#   com.codesourcery.installer.IInstallModule interface or extend the
#   com.codesourcery.installer.AbstractInstallModule class. Modules can
#   contribute additional install wizard pages and install actions.
#   Installer modules are registered using the
#   "com.codesourcery.installer.modules" extension point. The installer
#   includes one module, "com.codesourcery.installer", that contributes the
#   common wizard pages and install actions.
#   By default, all registered modules are loaded. This property can be used
#   to only include certain modules. The identifiers for modules that should
#   be loaded should be separated with a comma.
#####
##
#eclipse.p2.modules=com.codesourcery.installer.generalModule

```