

Harmony

USER MANUAL



LaBRI Software Engineering Group
<http://se.labri.fr/>

Version 0.9

Contents

1	Introduction	3
2	Try Harmony in 2 minutes	4
3	Advanced use	5
3.1	Configuration files	5
3.2	Global configuration	5
3.2.1	Database	6
3.2.2	Parallelism	6
3.2.3	Analyses	7
3.3	Source configuration	8
4	Develop new analyses with Harmony	9
4.1	Install the Harmony plug-in	9
4.2	Run Harmony within Eclipse	10
4.3	Create a new analysis	13
4.3.1	Using the wizard	13
4.3.2	Starting from scratch	15

Chapter 1

Introduction

Harmony is a framework that helps you to design and run analysis done on source code or bug repositories. Examples of such analysis are (1) the computation of source code metrics with the intent to perform a correlation with bugs distribution, (2) the measurement of software developers practices to infer their impact on software development, or (3) the computation of statistics to better understand a set of software projects.

Harmony offers the following facilities that are mandatory to perform any analysis:

- It defines an abstraction layer to design your analysis. This abstraction layer defines few simple concepts used by any analysis (Source, Event, Item, Action and Authors).
- It manages interoperability by providing connectors to many well-known repositories (Git, Mercurial, SVN, BugZilla, etc.).
- It handles parallelism that can be done when running your analysis. Such a parallelism allows a better efficiency specially when the analysis has to be ran on a large set of software projects.
- It supports the scheduling of several analyses with the objective to design analysis by composition.
- It provides a persistence facility to store the results of analysis into data bases.

This document explains how to install, run, configure Harmony and how to design your own analysis.

Chapter 2

Try Harmony in 2 minutes

This brief chapter describes the few steps required to try Harmony on your computer. Harmony is written in Java. You must have at least Java 1.7 to run Harmony. You can download it from: <http://www.java.com/fr/>. Once it is done or if you already have Java on your system follow these simple steps:

- Download the Harmony executable from this repository:
<https://code.google.com/p/harmony/downloads/list>
- Extract the archive
- Run the native launcher of the application located in the extracted folder:
 - *Harmony.exe* on Microsoft Windows
 - *Harmony* on Linux and Mac OS X

You should now have loaded all the Harmony components and see the *OSGi* prompt : ‘*osgi>*’. If you have an error check that your path is correctly set (see <http://www.java.com/fr/download/help/path.xml> for help).

- Finally just type *harmony* and wait for the default analysis to be executed.

Congratulations, you have just run your first Harmony analysis. You should have now an output directory named *out* that contains the results of this analysis. The default analysis is called *reporting* and generates several graphs on relationships between authors, events or item (see Harmony model for more explanation). The default source was used here, if you want to test Harmony on your own source repository the next chapter is for you.

Chapter 3

Advanced use

This chapter introduces the configuration of Harmony for advanced use : selection of sources and analyses, database configuration, parallelism.

3.1 Configuration files

The configuration of Harmony is achieved with two configurations files: the global configuration file presented in section 3.2 and the source configuration file presented in section 3.3. When you launch Harmony without arguments like this:

```
harmony
```

default configurations files are used in this case. You may found them under:

```
/configuration/fr.labri.harmony/
```

They are named *default-global-config.json* and *default-source-config.json*. You can tell Harmony that you want to use other configuration files by indicating the path of the files like this:

```
harmony ./my-global-config.json ./my-source-config.json
```

3.2 Global configuration

In this section we will look at the parameters of the JSON files used for the global configuration. By convention the name of these files end by 'global-config.json'. The listing 3.1 shows the default file for the global configuration.

```

1 {
2   "database":{
3     "url":"jdbc:h2:test",
4     "user":"sa",
5     "password":"",
6     "driver":"org.h2.Driver"
7   },
8   "execution":{
9     "threads":1,
10    "timeout":120
11  },
12  "source-analyses":[
13    {
14      "class":"ReportAnalysis",
15      "require-actions":true
16    }
17  ]
18 }

```

Listing 3.1: Default JSON file for the global configuration

3.2.1 Database

As shown in the listing 3.1, Harmony uses by default the H2 database engine (<http://www.h2database.com/>) in its embedded form. Even though this configuration is sufficient for testing we recommend to use a database server for running large studies with Harmony. In addition to H2, Harmony can use a MySQL server, the Community Server Edition for example: <http://dev.mysql.com/downloads/mysql/>. Once the MySQL server is installed and running, you will need to modify your global config file to set up the connection with the server. The listing 3.2 presents an example of such configuration.

```

1   "database":{
2     "url":"jdbc:mysql://localhost:3306/",
3     "user":"root",
4     "password":"strongPassword",
5     "driver":"com.mysql.jdbc.Driver"
6   }

```

Listing 3.2: Example of database configuration for a MySQL server

3.2.2 Parallelism

Harmony uses Java threads for launching the analyses. Thus it can exploit shared memory architectures such as multicore architectures. You just have to modify the number of threads you want to use (see listing 3.3) in the

execution section. Only one thread is used by source in order to minimize concurrent access to the same resources. Hence you will need multiple source to benefit from parallelism. The *timeout* parameter indicates in minutes the maximum execution time allocated to a thread for executing all the analyses asked on its associated source.

```
1 "execution":{
2   "threads":4,
3   "timeout":108
4 }
```

Listing 3.3: Example of parallel execution configuration

3.2.3 Analyses

Several analyses can be executed on the same source, these analyses can even be chained and share data. This aspect will be further discussed in chapter 4. Harmony possesses its own scheduler so you just have to declare the list of analyses as described in listing 3.4 and Harmony will take care of executing them in a correct order.

```
1 "source-analyses":[
2   {
3     "class":"ReportAnalysis",
4     "require-actions":true
5   },
6   {
7     "class":"ClocAnalysis",
8     "require-actions":false
9   }
10 ]
```

Listing 3.4: Example of multiple analyses configuration

The construction of the complete Harmony model can be quite costly and some analyses do not require this complete model. The *require-actions* parameter aims to tackle this problem by telling Harmony if it is necessary to build the actions in the Harmony model (see Chapter 4 for more information).

By default Harmony is shipped with several analyses:

- Reporting
- ScanLib
- SourceAnalyzer

3.3 Source configuration

In this section we present the parameters of the JSON files used for the configuration of the sources. By convention the name of these files end by 'source-config.json'.

The listing 3.5 shows an example of configuration file with three sources. You can add as many source as your study requires especially if you want to exploit the parallel architecture of your computer.

```
1  [
2    {
3      "url": "https://github.com/isylhdin/PED_MaisonHote.git",
4      "class": "JGitSourceExtractor"
5    },
6    {
7      "url": "https://github.com/twitter/bootstrap.git",
8      "class": "JGitSourceExtractor"
9    },
10   {
11     "url": "http://macchiato.fr/svn/",
12     "class": "SvnKitSourceExtractor",
13     "user": "myself",
14     "password": "123456789"
15   }
16 ]
```

Listing 3.5: Example of configuration file for the sources

For each source specified using the *url* attribute you must indicate an extractor that is compliant with it (*class* attribute). You can develop your own but Harmony already provides a fair number of source extractor :

- *JGitSourceExtractor* for Git repositories. This extractor is based on JGit, a Java implementation of Git.
- *GitSourceExtractor* for Git repositories. The Git native client must be installed on your machine to use this extractor.
- *SvnKitSourceExtractor* for SVN repositories.
- *Hg4jSourceExtractor* for Mercurial repositories.
- *BugzillaSourceExtractor* for Bugzilla repositories.
- *TFSSourceExtractor* for Team Foundation repositories.

Additionally if your source requires an authentication, you can use the *user* and *password* attributes as shown in the last source declaration of the listing 3.5.

Chapter 4

Develop new analyses with Harmony

This chapter explains step-by-step how to set up a development environment for Harmony. We will also show how it is possible to develop and run your own Harmony analyses. The Harmony framework is based on the OSGi component model. Even though you may use any OSGi implementation for developing and running your analyses, this documentation shows how to do it with the Equinox implementation developed by the Eclipse community.

4.1 Install the Harmony plug-in

First things first, you need to install Eclipse from <http://www.eclipse.org/downloads/>. We recommend Eclipse Classic as it is packaged with the Plug-in Development Environment.

Once Eclipse has been downloaded and unzipped, run it. Now go to the menu **Help** → **Install new software**. At the right of the "work with" field, click on the *Add...* button. On the popup type for *Name*: Harmony and for *Location*: <http://se.labri.fr/data/harmony/update-site> and click on *Ok*. At the right of the "work with" field, select in the list the Harmony update site that you have just created. Finally select the **Harmony** category, and install the selected plug-ins as shown in the figure 4.1.

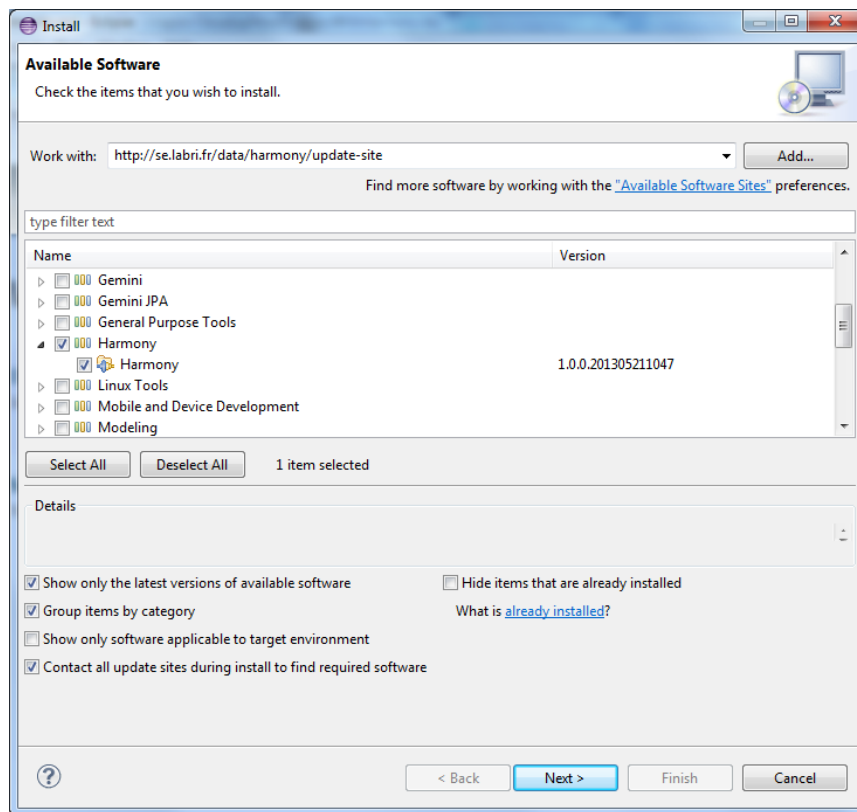


Figure 4.1: Harmony installation within the Eclipse IDE

Restart Eclipse. The Harmony plug-ins are now loaded in your Eclipse installation.

4.2 Run Harmony within Eclipse

Eclipse uses *Run configurations* in order to know what should be launched and how : which plugins, which execution environment,... A default *Run configuration* is provided in the `harmony.core` plug-in. In order to use it you have to import the `harmony.core` plug-in from your Eclipse installation into your workspace.

You can do this via the **File** → **Import** menu. Then select **Plug-ins and Fragments** (see figure 4.2).

In the **Import Plug-ins and Fragments** wizard, select **Import As** → **Project with source folders** as shown in figure 4.2.

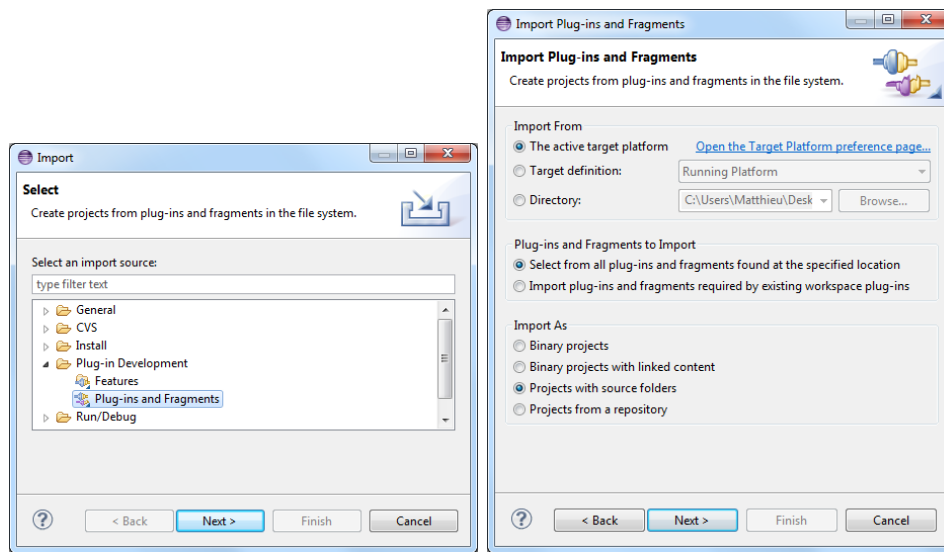


Figure 4.2: Menu for the importation of plugins in the workspace

In the next step, add the *fr.labri.harmony.core* plug-in in the list of *Plug-ins and Fragments to Import*. You may use the filter box with "fr.labri" in order to find more rapidly the *fr.labri.harmony.core* plug-in as demonstrated in figure 4.3.

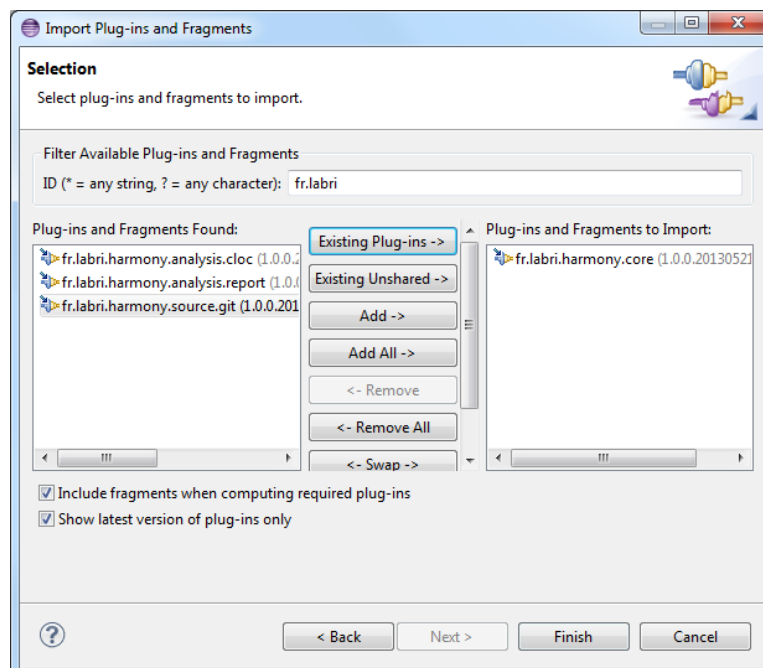


Figure 4.3: Selection of the Harmony core plugin for importation

Finally press **Finish**. A new project has been added to your workspace. You may have build errors, in such a case you need to clean the project via the **Project** → **Clean** menu.

We will be able to run Harmony in a few steps. The next thing to do is to copy the **configuration** directory –located in the **resources** directory from the **fr.labri.harmony.core** plugin– into the root of workspace, where you should have a **.metadata** directory and the plug-in you imported:

```
1 drwx-----+ 1 XXXXXX None 0 21 mai 11:27 .metadata/
2 drwx-----+ 1 XXXXXX None 0 21 mai 14:09 configuration/
3 drwx-----+ 1 XXXXXX None 0 21 mai 13:08 fr.labri.harmony.
   core/
```

Then go back into Eclipse, and open the **Run Configurations** window (via the **Run** menu). In the left menu, you should see an **HarmonyEquinox** configuration under the **OSGi Framework** node as shown in figure 4.4. Select it.

You can now see a list of plugins. The ones that are selected are the ones that will be loaded if you launch this *Run configuration*. Now, uncheck the "Show only Selected" checkbox (on the right) and type "fr.labri" in the filter box located at the top of the plugins list. Add the harmony plugins that are not imported in your workspace (i.e. reporting, cloc, and source.git) as shown in figure 4.4. Remember that if you develop new analyses, you must update your *Run configuration* by adding the new plugin otherwise you will not be able to run your analysis.

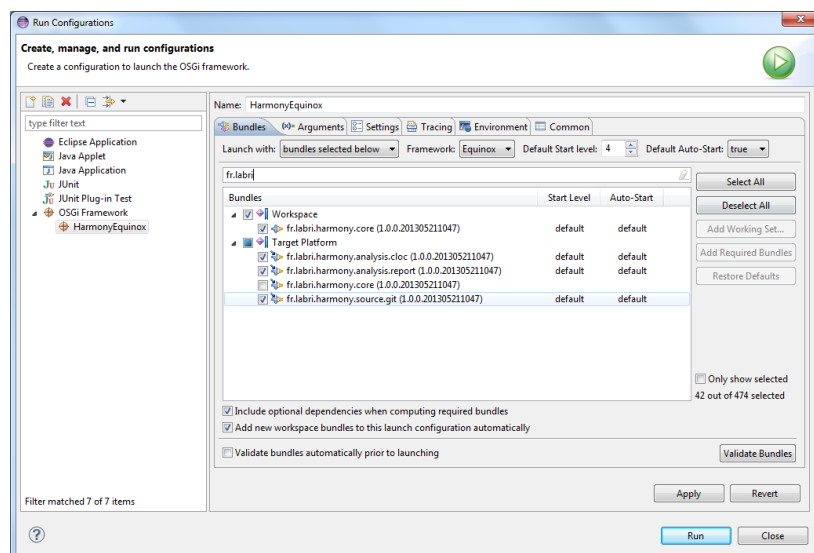


Figure 4.4: Set up of the run configuration

Finally you can hit **Run** to launch the selected plugins. Once the OSGi console is loaded, type **harmony** (see figure 4.5).



```
HarmonyEquinox [OSGi Framework] C:\Program Files\Java\jre7\bin\javaw.exe (21 mai 2013 16:14:09)
!MESSAGE Bundle fr.labri.harmony.analysis.report_1.0.0.201305211047 [5] was not resolved.
!SUBENTRY 1 org.eclipse.osgi 2 0 2013-05-21 16:14:11.217
!MESSAGE Bundle fr.labri.harmony.core_1.0.0.201305211047 [7] was not resolved.
!SUBENTRY 1 org.eclipse.osgi 2 0 2013-05-21 16:14:11.217
!MESSAGE Bundle fr.labri.harmony.source.git_1.0.0.201305211047 [12] was not resolved.
!SUBENTRY 1 org.eclipse.osgi 2 0 2013-05-21 16:14:11.217
!MESSAGE Bundle fr.labri.harmony.analysis.cloc_1.0.0.201305211047 [16] was not resolved.
[2013/05/21 16:14:18] Initializing Workspace for source https://github.com/jrfaller/test-git.git
[2013/05/21 16:14:18] Initializing existing local workspace at: C:\Users\Matthieu\Desktop\vis\temp\github.com\jrfaller\test-git.git
[2013/05/21 16:14:22] Extracting Events for source https://github.com/jrfaller/test-git.git
[2013/05/21 16:14:24] Extracting Actions for source https://github.com/jrfaller/test-git.git
[2013/05/21 16:14:26] Starting reporting analysis on https://github.com/jrfaller/test-git.git.
[2013/05/21 16:14:31] Finished execution of analyses
osgi>
```

Figure 4.5: OSGi console with Harmony running

4.3 Create a new analysis

Now that your development environment is ready and that you know how to run analysis within Eclipse. We will see how you can create your own analysis. There are two ways of doing so, the easy way by using the dedicated wizard (see section 4.3.1) and the hard way by creating the project from scratch (see section 4.3.2). In both cases do not forget to add your new analysis to your *Run Configuration* (see previous section).

4.3.1 Using the wizard

Make a right-click on the *Project Explorer* view and select **New** → **Project**. As shown in figure 4.6 select the *analysis* wizard located in the *Harmony* category and click on *Next*.

Now you should see something a form similar to the figure 4.7 that ask you details about the analysis you want to create. The *project name* must follow the Java package naming convention as the project name will be used as your root package as well as your plugin unique identifier. The *Analysis class name* is the name of the class that will implement the analysis service trough the method *runOn(Source src)*. Press *Finish*.

Implement you analysis in the method *runOn(Source src)*. Take a look at existing analyses for inspiration. Follow the same procedure as the one you have followed for importing the *fr.labri.harmony.core* (see section 4.2). Finally you can run it by modifying your configuration files (see Chapter 3) and your *Run Configuration* (see Section 4.2).

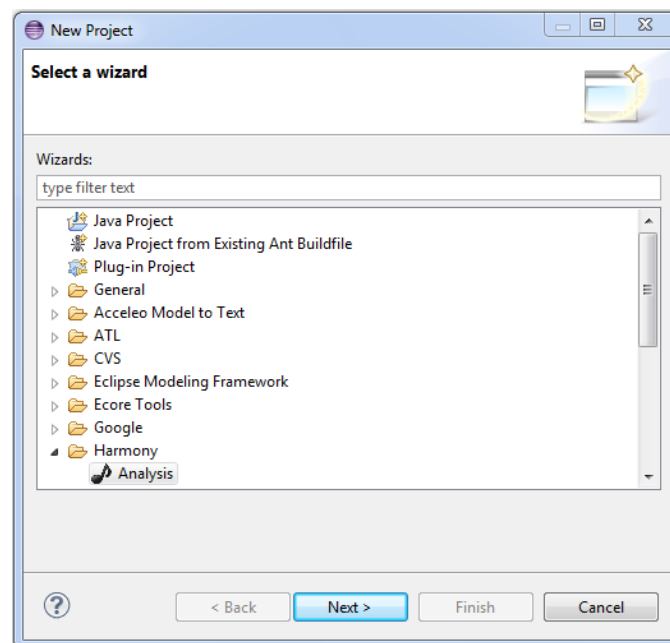


Figure 4.6: Selection of the Harmony wizard for creating a new analysis

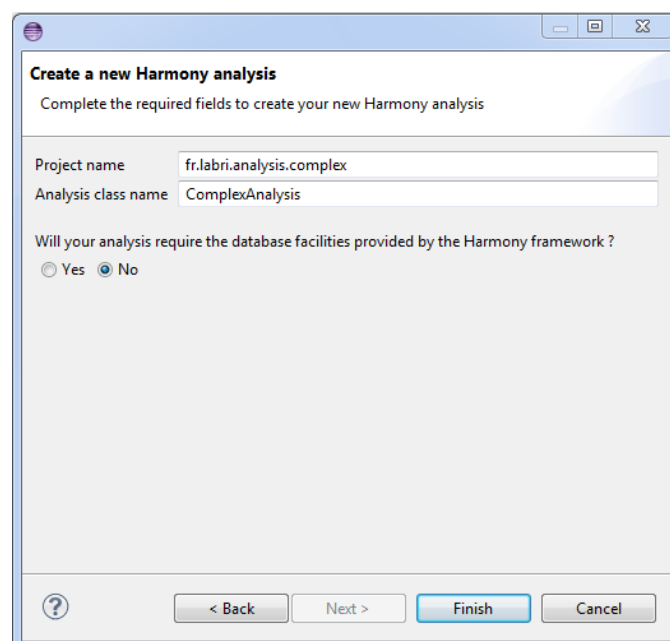


Figure 4.7: Form of the wizard for creating a new analysis

4.3.2 Starting from scratch

If for some inexplicable reason you want to create your analysis project from scratch, this section will guide you. The first step is to create a plug-in project for this analysis. Make a right-click on the *Project Explorer* view and select **New** → **Project** → **Plug-in Project** and *Next*. You a wizard similar to the one presented in figure 4.8. For demonstration purpose we will call it **dummy**. Set the target platform to use the Equinox OSGi framework and press *Next*.

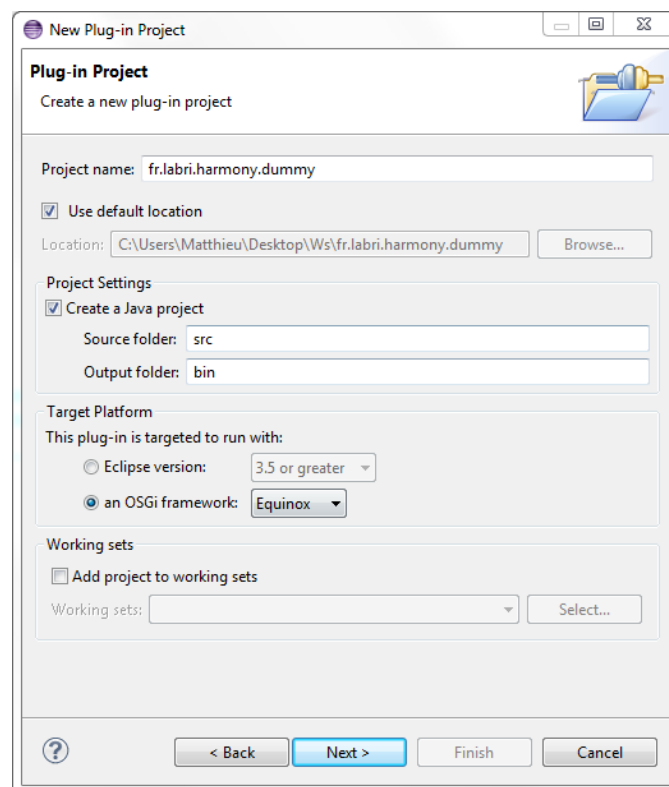


Figure 4.8: First page of the plug-in wizard

Press next. You should now see the **Content** form as shown in figure 4.9, uncheck the **Generate an activator...** checkbox. Finally press *Finish* to create your new project.

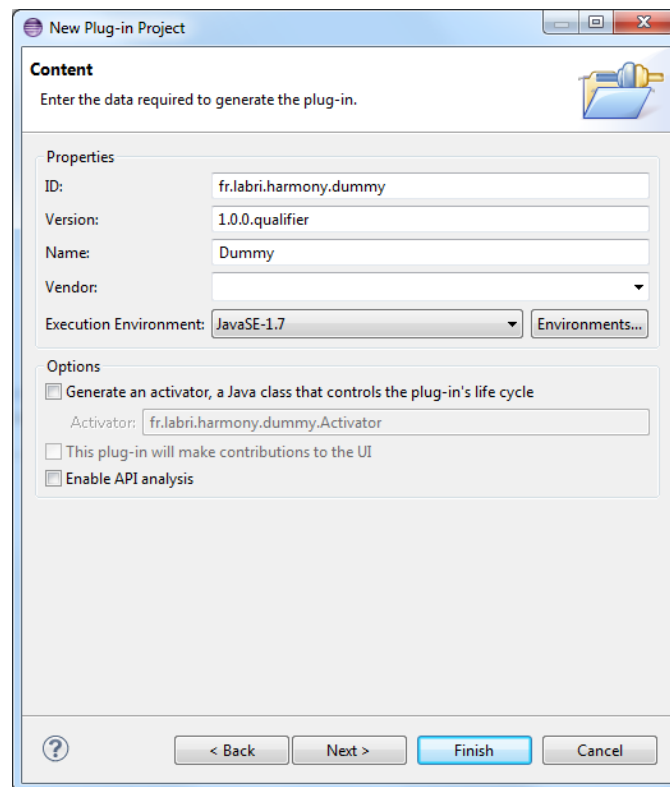


Figure 4.9: First page of the plug-in wizard

In order for your analysis to use the harmony framework, you need to add the corresponding dependencies to your plug-in configuration. If it is not already the case open the `MANIFEST.MF` file located in the `META-INF` folder. Open the **Dependencies** tab, and add the following imported packages:

```
1 fr.labri.harmony.core
2 fr.labri.harmony.core.analysis
3 fr.labri.harmony.core.config.model
4 fr.labri.harmony.core.dao
5 fr.labri.harmony.core.log
6 fr.labri.harmony.core.model
7 javax.persistence
```

You should obtain a configuration similar to the one presented in figure 4.10

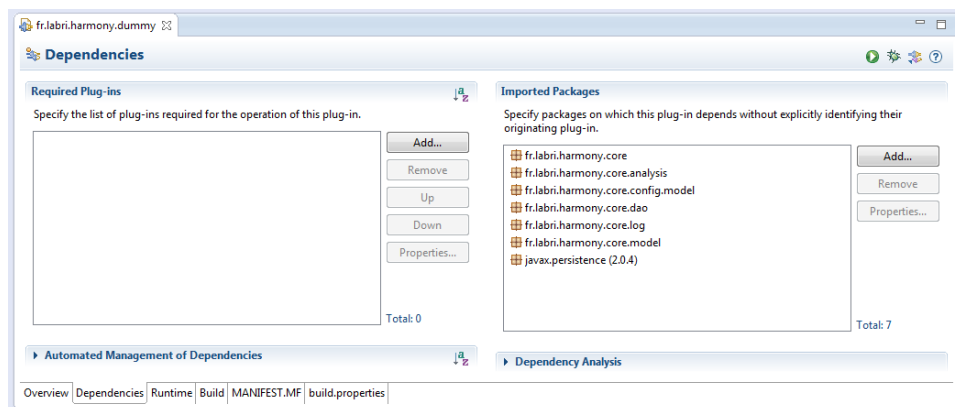


Figure 4.10: First page of the plug-in wizard

You now have to add your analysis main class that will implement the *AbstractAnalysis* class defined in the Harmony Core. To do so, use the Eclipse *new class* wizard, and check the **Constructors from superclass** checkbox as shown in the figure 4.11.

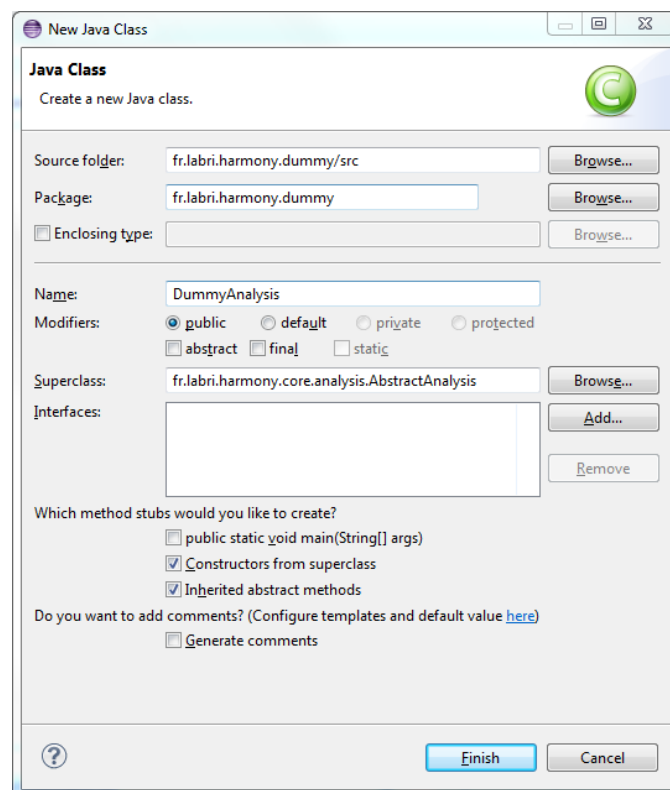


Figure 4.11: Creation of the main class of class of your analysis

This is very important, as **both constructors from AbstractAnalysis are mandatory**. Also add as your superclass:

fr.labri.harmony.core.analysis.AbstractAnalysis

Press *Finish*. You should have something looking like the class presented in the listing 4.1. The method *runOn(Source src)* is where you must implement your analysis. Take a look at existing analyses for inspiration. Follow the same procedure as the one you have followed for importing the *fr.labri.harmony.core* (see section 4.2).

```
1 package fr.labri.harmony.dummy;
2
3 import java.util.Properties;
4
5 import fr.labri.harmony.core.analysis.AbstractAnalysis;
6 import fr.labri.harmony.core.config.model.AnalysisConfiguration
7 ;
8 import fr.labri.harmony.core.dao.Dao;
9 import fr.labri.harmony.core.log.HarmonyLogger;
10 import fr.labri.harmony.core.model.Source;
11
12 public class DummyAnalysis extends AbstractAnalysis {
13
14     public DummyAnalysis() {
15         super();
16     }
17
18     public DummyAnalysis(AnalysisConfiguration config, Dao dao,
19         Properties properties) {
20         super(config, dao, properties);
21     }
22
23     @Override
24     public void runOn(Source arg0) {
25         HarmonyLogger.info("Hello, I am Dummy!");
26     }
27
28 }
```

Listing 4.1: DummyAnalysis lass

You now have a class that implements the analysis service so you need to inform the OSGi runtime about your implementation. Create an empty folder called *OSGI-INF* at the root of your project. Then create a new component definition : make a right-click on your plug-in project and select **New → Component Definition**. Complete the form by taking example of the figure 4.12, for example the component definition should be in the newly created *OSGI-INF* folder. Finally press *Finish*.

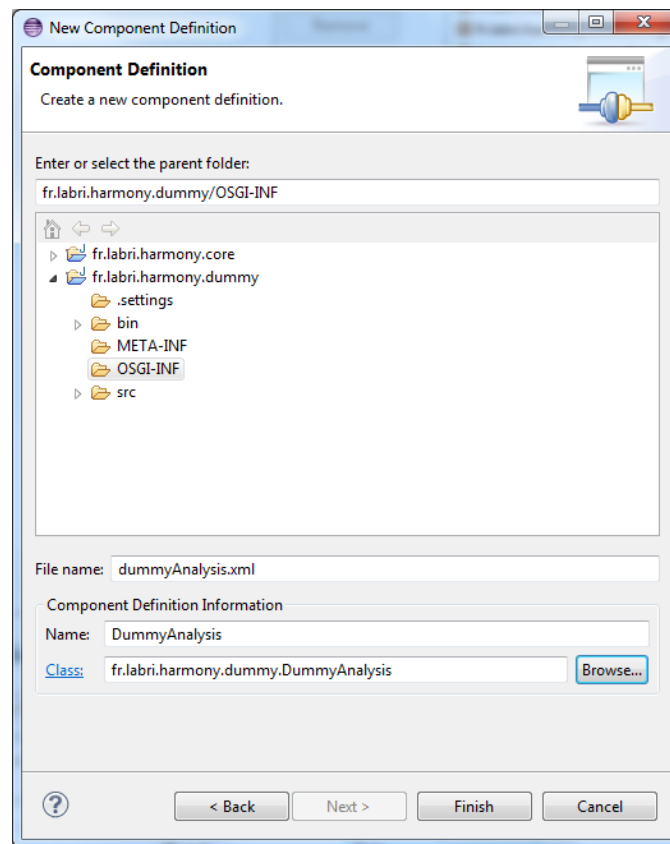


Figure 4.12: Wizard for creating a new component definition

Open the newly created file and go the *Service* tab as shown in the figure 4.13. Click on the *Add* button of the *Provided Services* section and select `fr.labri.harmony.core.analysis.Analysis`, press *Ok*.

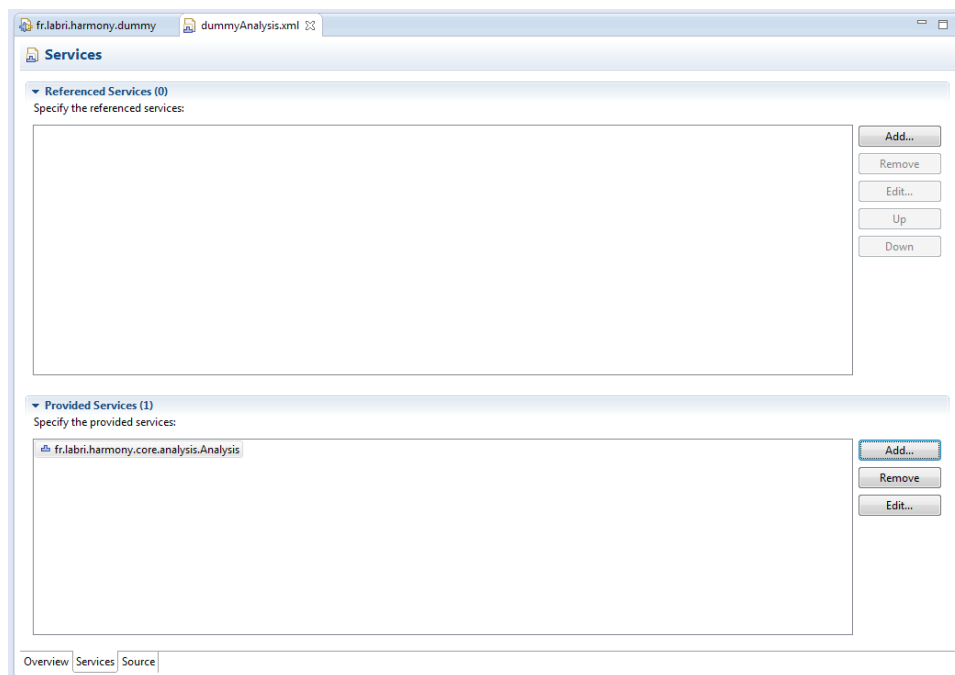


Figure 4.13: Provided services

Your analysis project is now set up. You can run it by modifying your configuration files (see Chapter 3) and your *Run Configuration* (see Section 4.2).