



Michael Aro

Intents Registry Eclipse Plugin

Technology and Communication
2012

FOREWORD

Writing a thesis is a long process, but it is ultimately rewarding when you can hold the finished product in your hands and be proud of it. I would like to first and foremost thank my awesome family for supporting me all the way and for keeping me going whenever I was about to get stuck.

Secondly, I would also like to thank Dr Ghodrat Moghadampour for sharing the knowledge and guiding me through the complexity and beauty of software engineering over the past couple of years. I would also like to send a big kudos to Dr Friedger Muffke for giving me the opportunity and support to work on this project and use it as a bachelor thesis and giving constructive feedbacks on the project.

Personally, I would like to thank the Vaasa Polytechnic staff including but not limited to Dr Jarmo Makela, Antti Virtanen, Kalevi Ylinen, Vesa and his wife, Tarja Siren, Timo Kankaanpää, Maj-Gret Berg, Dr Ritva Rapila, Elina Luokkanen, Johan Dams, and Dr Seppo Makinen.

I would also like to thank the OpenIntents team and the Android, Eclipse and open source community. All of the people who work to make the products better either directly or with bug reports and patches, help on forums, question and answer sites, and creating documentations, libraries and tools are a big reason why these different platforms work and thrive. I am an open source activist and I fully believe in the idea of contributing back whenever you take given the possibility.

Last, and definitely not least, I wholeheartedly thank my friends for their words of encouragement and support.

Michael Aro

ABSTRACT

Author	Michael Aro
Title	Intents Registry Eclipse Plugin
Year	2012
Language	English
Pages	64 + 10 Appendices
Supervisor	Dr Ghodrat Moghadampour, VAMK
Mentor	Dr Friedger Müffke, Founder of OpenIntents and Droidcon

The thesis has been carried out with the OpenIntents organization. The purpose of this project was to build an Eclipse plugin that enables developers to use the extensible markup language (XML) data available in the OpenIntents intents registry and insert code blocks corresponding to the selected intents directly into their Android projects. It is easily accessible via the menu feature of Eclipse. In addition, developers can browse online for libraries, application source code, uris, extras and other artifacts and information needed to do their work without leaving the Eclipse integrated development environment.

The program was implemented on top of the Eclipse framework. Eclipse is the target integrated development environment (IDE) because of its modular and pluggable structure, its significant amount of users including OpenIntents developers, who can benefit from the plugin and provide feedback for further improvement. Eclipse has support for multiple languages and operating system platforms. Also part of this project is learning how to contribute to open source projects by organizations such as OpenIntents and Eclipse Foundation through open source programming.

The main achievements of this project were a developer friendly codebase on Github, a release in the Eclipse marketplace and plenty of developer and user-oriented documentation. Using this plugin is user-friendly. The intents registry plugin and Eclipse IDE complement each other by creating an efficient Android application development environment.

Keywords Plug-in, Intents, Registry, Eclipse, open source, OpenIntents

Contents

FOREWORD	2
ABSTRACT	3
LIST OF FIGURES, TABLE AND SNIPPETS	7
1 INTRODUCTION	12
1.1 OpenIntents	13
1.2 Goal of the work.....	13
2 TECHNOLOGY OVERVIEW	15
2.1 Eclipse Plugin Development and Platform API.....	16
2.1.1 Eclipse Plug-in	18
2.1.2 Structure of a plugin.....	18
2.1.3 Plug-in Directory or JAR file.....	20
2.1.4 Installing a Plug-in	22
2.1.5 Extensions and extension points	23
2.1.6 Building and Maintaining the Github Update Site.....	25
2.2 Updating the repository	26
2.3 Internationalization and Help	26
2.4 ANT	27
2.5 Android and Intents	27
2.5.1 Structure of Intent	27
2.5.2 Explicit and Implicit intents	28
2.5.3 Open Intents	28
2.5.4 Open Intents registry	29
2.5.5 Intent Filters	30
2.5.6 Android Manifest File	30

2.5.7	Android Manifest Editor	31
2.5.8	Default Java Editor.....	32
3	THE APPLICATION DESCRIPTION.....	33
3.1	Application Requirements	33
3.2	Functional Specification.....	34
3.3	Class Hierarchy	35
3.3.1	Main UI Classes	36
3.3.2	Handler Classes.....	38
3.3.3	Plugin Activator and Logging Classes	38
3.3.4	Model and Parser Classes	40
3.4	Description of Functions	40
3.4.1	Insert Code Block.....	40
3.4.2	Browse Intent	41
3.5	Deployment Diagram	43
4	DATA PERSISTENCE AND GUI DESIGN	44
4.1	Main GUI	44
4.2	Browser Window.....	45
5	IMPLEMENTATION.....	46
5.1	Software Methodology	46
5.2	Data Retrieval and Persistence	46
5.3	Generating Resources.....	49
5.4	View and Dialog Handler.....	50
5.5	Logging	52
6	TESTING	53
6.1	JUnit Testing	53

6.2	SWTBOT.....	54
6.3	Inserting Code Blocks	55
6.4	Browsing Libraries and Artifacts	56
7	SUMMARY	57
7.1	Publishing plugin to the Eclipse marketplace	57
7.2	Becoming a contributor / committer	58
7.2.1	Contributing to OpenIntents.....	58
7.2.2	Project mailing list	58
8	CONCLUSION.....	59
8.1	Future Work	60
8.1.1	Feature.....	60
9	REFERENCES.....	61
	APPENDIX 1	65

LIST OF FIGURES, TABLE AND SNIPPETS

FIGURE 1. ECLIPSE PLATFORM UI	15
FIGURE 2. A SCREENSHOT OF THE ECLIPSE API SPECIFICATION	17
FIGURE 3. ECLIPSE EXTENSION POINT STRUCTURE.....	19
FIGURE 4. DECLARING A NEW EXTENSION.....	20
FIGURE 5: INTENTS REGISTRY ECLIPSE PLUGIN PROJECT TREE.....	21
FIGURE 9. PROJECT PAGE	24
FIGURE 10. CODE REPOSITORY.....	25
FIGURE 11. OPEN INTENTS ONLINE REGISTRY	30
FIGURE 12. PREFERENCES DIALOG SHOWING FILE ASSOCIATION BETWEEN EDITOR AND FILE TYPES.....	31
FIGURE 13. DEFAULT EDITOR.....	32
FIGURE 14. USE CASE DIAGRAM OF THE PLUGIN.....	35
FIGURE 15. A PACKAGE DIAGRAM OF THE PLUGIN SHOWING THE RELATIONSHIPS AMONG THE PACKAGES	35
FIGURE 16. STRUCTURE OF THE MAIN VIEW	36
FIGURE 17. FILTER DIALOG CLASSES.....	37
FIGURE 18. HANDLER CLASSES	38
FIGURE 19. PLUGIN ACTIVATOR AND LOGGING CLASSES	39
FIGURE 20. PARSER AND MODEL CLASSES.....	40
FIGURE 21. SEQUENCE DIAGRAM – ADD INTENT.....	41

FIGURE 22. SEQUENCE DIAGRAM – BROWSE INTENT	42
FIGURE 23. DEPLOYMENT DIAGRAM FOR THE PLUGIN	43
FIGURE 24. ADD INTENT	45
FIGURE 25. BROWSE INTENT	45
FIGURE 26. LOCAL REGISTRY OF INTENT ACTIONS AND TITLES	50
FIGURE 27. LOGGING	52
FIGURE 28. INTENT SELECTION PAGE	55
FIGURE 29. INTENTS REGISTRY BROWSER VIEW	56
FIGURE 30. PLUGIN ON THE ECLIPSE MARKETPLACE	57
TABLE 1: ECLIPSE PLATFORM COMPONENTS	17
SNIPPET 1. PLUGIN MANIFEST	22
SNIPPET 2. PROPERTIES FILES FOR ORG.OPENINTENTS.INTENTSREGISTRY	22
SNIPPET 3. EXTENSION POINTS	23
SNIPPET 4. EXPLICIT INTENT	28
SNIPPET 5. IMPLICIT INTENT	28
SNIPPET 6. AN OPEN INTENT	29
SNIPPET 7. ANDROID MANIFEST FILE	31
SNIPPET 8. INTENT TAG	47
SNIPPET 9. PARSE METHOD.....	47
SNIPPET 10. INTENTSREGISTRYPARSER.....	48
SNIPPET 11. INTENTSREGISTRYRESOURCE	49

SNIPPET 12. EXECUTE METHOD OF FILTER DIALOG..... 50

SNIPPET 13. EXECUTE METHOD OF INTENTSREGISTRYVIEW 51

SNIPPET 14. REGISTRYRESOURCE JUNIT TEST CASE 53

SNIPPET 15. INTENTSREGISTRYVIEW TEST CASE 55

LIST OF ABBREVIATIONS

API	Application Programming Interface.
Bug	Flaw in software.
Bugzilla	Bug tracking software.
Committer	Committer is an individual who has been given the write access to a codebase hosted by someone.
Eclipse	Popular open source software tool platform featuring a GUI and IDE.
GUI	Graphical User Interface.
IDE	Integrated Development Environment.
Java	A computer language.
JIT	Just-In-Time.
JUnit	Unit testing framework for the Java programming language.
PDE	Plug-in Development Environment.
Plug-in	Software aimed to provide additional functionality on top of the Eclipse platform.
Git	A distributed version control system
Github	A public git repository.
UML	Unified Modelling Language.
URL	Uniform Resource Locator.
SDK	Software Development Kit.

OS	Operating System
SQL	Structured Query Language
XML	Extensible Markup Language
HTTP	HyperText Transfer Protocol
ANT	A build library
JRE	Java Runtime Environment
SWTBot	An open-source Java based GUI testing tool for SWT and Eclipse based applications
SWT	Standard Widget Toolkit
Test case	The smallest unit of a test
OI	OpenIntents

1 INTRODUCTION

The programming process includes the design, writing, testing, and maintenance of source codes with the end result of creating a file or set of files that can be executed on different platforms. In order to write a program, a code editor is needed for writing and editing the code. Integrated development environment (IDE) is an application program that can run on the desktop or in the cloud for the development of applications in different programming languages including Eclipse plugins. Typically, it consists of a code editor for editing files, compiler, debugger and a graphical user interface. There are numerous IDEs such as Cloud9, Eclipse, and Visual studio to mention a few.

Eclipse is a community for individuals and organizations who wish to collaborate on commercial-friendly open source software. Its projects are focused on building, deploying and managing software across the lifecycle. The Eclipse Foundation is a not-for-profit, member supported corporation that hosts the Eclipse projects and helps cultivate an open source community and an ecosystem of complementary products and services. The Eclipse project was originally created by IBM in November 2001 and supported by a consortium of software vendors. The Eclipse Foundation was created in January 2004 as an independent not-for-profit corporation to act as the steward of the Eclipse community. Today, the Eclipse community consists of individuals and organizations from cross section of the software industry. /35/

Plugin systems are built into IDEs for easy extensibility. The main idea is to keep the core IDE as slim as possible while plugins are added to provide additional functions or features depending on the use cases.

1.1 OpenIntents

OpenIntents is an open-source organization that design and develop open intents and interfaces to make Android applications work closely together. They provide free and open source software based on Google's Android mobile platform and an open framework. The organization was interested in making a plug-in for Eclipse that allows using Intents within the Eclipse IDE. This allows the developers to browse an online intents registry within Eclipse and insert a code block corresponding to the selected intents directly into their Eclipse project.

The plug-in is released under Apache 2.0 License and is freely available for anyone to download. This allows anyone to contribute to the open source community by modifying the source code of the plug-in. It works well on Eclipse Helios (version 3.6), Indigo (version 3.7) and Juno (version 4.2).

The plug-in is both Java and Eclipse based. It uses a simple data model wrapped into an API and then built an Eclipse based application that uses the API. It is free, open source and developed as a tool to be used by OpenIntents developers.

The intents registry Eclipse plug-in comes with a set of features, such as code insertion for intents and an easy to use web browsing interface for access to libraries.

At the heart of this software is an intents registry. This registry, much like a typical XML data source, defines all of the unique intents both actions and titles used throughout the plug-in. Using combinations of actions and titles we can define intents.

1.2 Goal of the work

- Experiment with Eclipse plug-in development: Eclipse is made up of so many plug-ins, and small runtime kernel and together they make up the Eclipse environment. Contributing to Eclipse is a very interesting challenge given the huge codebase of the software, and trying to understand the extension of

different plugins within this framework to make a new one. More importantly, I wanted to try and do something I have never done before.

- I was also interested in learning about open source programming and making contributions to free and open source software that can be used by others.
- Build an Intents Registry Eclipse plugin. Once the experimentation with Eclipse is done and there is a familiarity with the Eclipse APIs and codebase, build a plug-in on top of it.

In the subsequent chapters, we will go through the Eclipse framework and concepts as well as open intents registry. We will also get familiar with the aspect of Android system that is relevant to this plug-in. Code editors will be discussed. Once the Eclipse, Android, editors, plug-in structure and requirements for the plug-in have been described, we will move on to the design and implementation of the plug-in. This will be followed by the conclusion, future work, references and deployment of the plug-in.

2 TECHNOLOGY OVERVIEW

The tools and technologies used for this application include Eclipse framework, Java, Git, Ant, JUnit, SWTBot and xml. Eclipse for RCP and RAP developers is the IDE used.

Eclipse is a free and open-source software framework which is platform independent. It is written mainly in Java. The Eclipse libraries have been used to develop Eclipse IDE and Eclipse compiler.

The next figure shows the screen shot of the workbench window. The package explorer on the left displays the projects and files in the workspace of the user. The default java editor of type text editor in the centre displays the source code content of a file. And the bottom view shows the console for displaying contents of output streams.

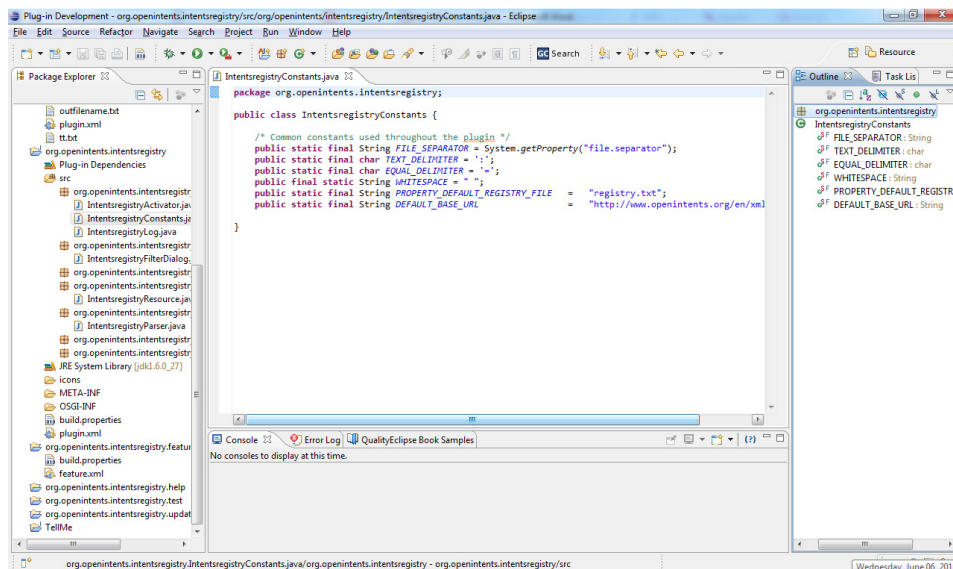


Figure 1. Eclipse Platform UI

Java is a programming language and computing platform first released by Sun Microsystems in 1995. It is the underlying technology that powers state-of-the-art programs including utilities, games, and business applications. Java runs on more

than 850 million personal computers worldwide, and on billions of devices worldwide including mobile and TV devices. /36/

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the web and elsewhere. /37/

2.1 Eclipse Plugin Development and Platform API

Eclipse is an open source community and a platform consisting of Eclipse and Eclipse-based products that can be extended or integrated into other applications by developers. The Eclipse Platform, which is a part of the Eclipse project, allows developers to extend the IDE functionality through the plug-ins. A developer can create custom editors, menus, commands, or wizards as part of a plug-in by extending the Eclipse platform. Eclipse can also be integrated with other applications, for example in a web application. Eclipse is built on the OSGi framework. The framework provides an open architecture that is modular and dynamic and required for running the Eclipse IDE.

There are many Eclipse projects namely Business Intelligence and Reporting Tools (BIRT), data tools platform, the core Eclipse project, Eclipse modeling project, Mylyn, SOA platform project, technology project, tools project, test and performance tools platform project, and the web tools platform project. Within the core Eclipse project is the Eclipse platform which provides the core APIs and services for creating plug-in extensions. It provides the runtime environment in which the plug-ins are loaded, integrated and executed. It is this platform that enables developers to build tools. One can think of the platform as universal since it is an IDE that can manage any kind of resource such Java, XML, HTML, JSP, CSS files to mention a few of them.

The following table illustrates several components of the Eclipse platform. /1/

Table 1: Eclipse Platform components

Name	Description
Ant	Eclipse/Ant integration
Core	Platform runtime and resource management
CVS	Platform CVS Integration
Debug	Generic execution debug framework
Releng	Release engineering
Search	Integrated search facility
SWT	Standard Widget Toolkit
Team/Compare	Generic team and compare support frameworks
Text	Text editor framework
User Assistance	Help system, initial user experience, cheat sheets, etc.
UI	Platform user interface
Update	Dynamic Update / Install / Field Service

The platform API is a set of interfaces that enables a developer written application or plug-in to communicate with the Eclipse platform.

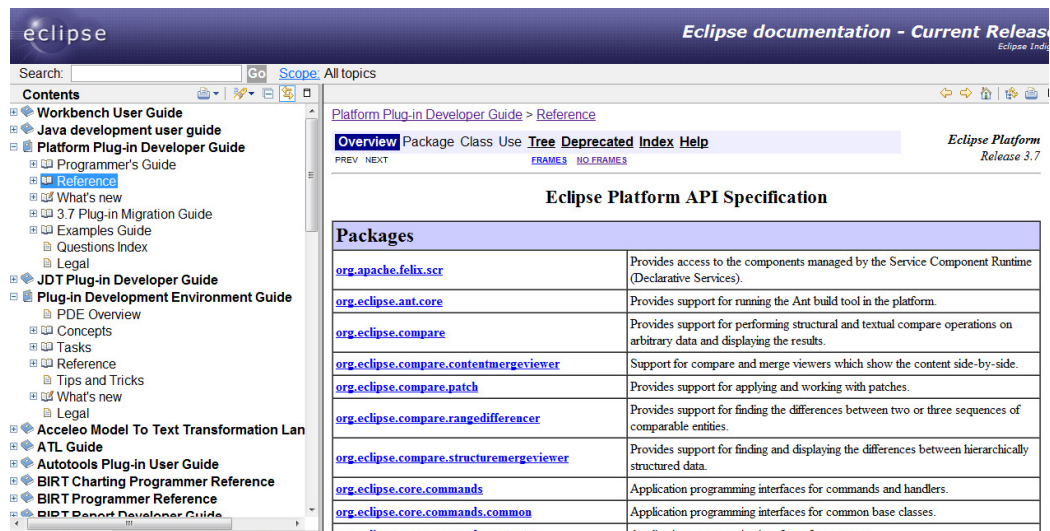


Figure 2. A screenshot of the Eclipse API specification

2.1.1 Eclipse Plug-in

A plug-in is used to group code into a self-contained, extendable and sharable unit.

Plug-ins are self-contained as each plug-in contains some portion of code. The plug-in specifies other plug-ins (or java packages) it requires in order to work as well as the set of java packages it provides. Eclipse based software typically contains multiple plug-ins, which can be added, replaced or removed to alter the functionality of the program.

A plug-in can be extended using extensions and extension points and can provide one or more extension points so other plug-ins can add to the functionality of the plug-in. It may also provide extensions to connect to other plug-ins.

A plug-in can be shared by exporting it to a directory or jar file and added to other applications. They can be grouped into features which can be distributed and installed into applications.

Eclipse plug-ins are based on OSGi bundles. OSGi is used to manage the plug-ins in an Eclipse application. A plug-in must contain a manifest file with valid OSGi headers for plug-in name and version. Extensions and extension points functionality added by Eclipse in addition to OSGi. To use extensions you must provide a plugin.xml file. PDE provides a full featured project and editor for creating and editing these files. /2/

2.1.2 Structure of a plugin

Eclipse is not a single monolithic program, but rather a small kernel containing a plug-in loader surrounded by many plug-ins. The loader is an implementation of OSGi specification and provides the environment in which plug-ins execute. An example of how plug-ins depend on one another is shown in Figure 3 below.

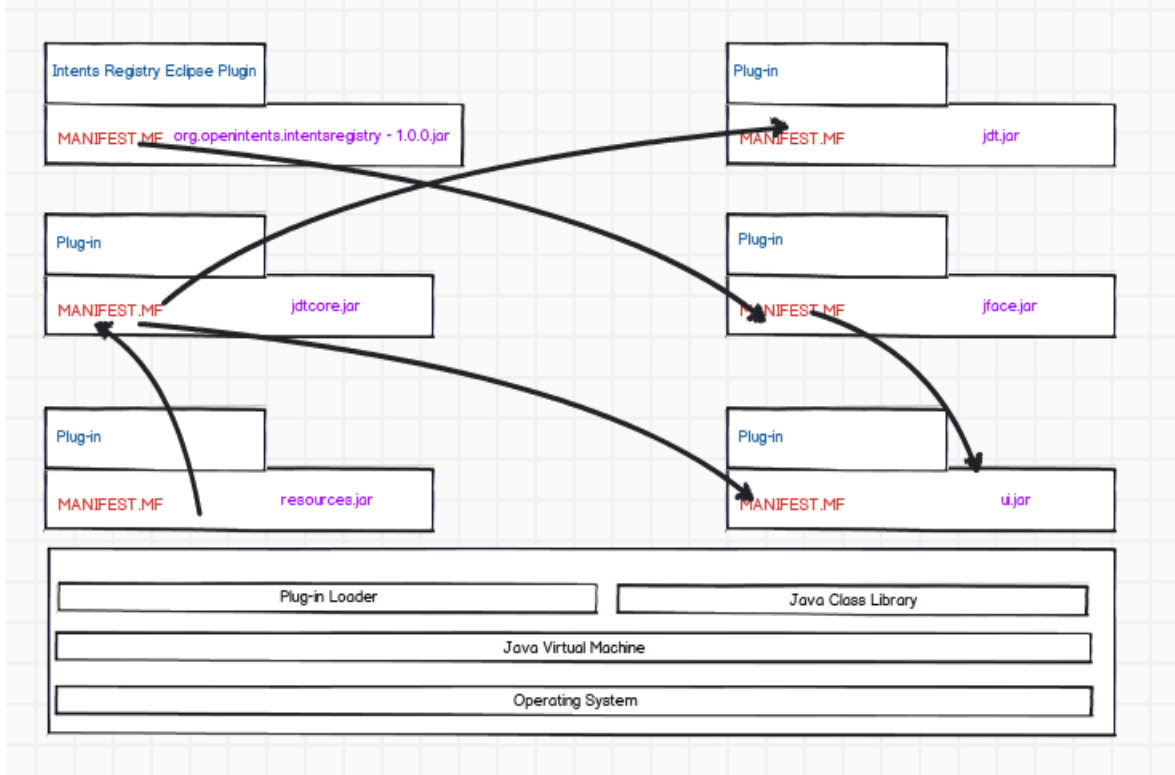


Figure 3. Eclipse Extension Point Structure

The dependencies of a plug-in are declared in the MANIFEST.MF while the services are declared in the plugin.xml file. The behavior is in the code. Plug-ins are loaded when needed, a term referred to as lazy-loading. The benefits of this include reduced startup time and memory usage.

The plug-in loader reads the manifest and plugin.xml file for each plugin on startup. A structure is built based on this information.

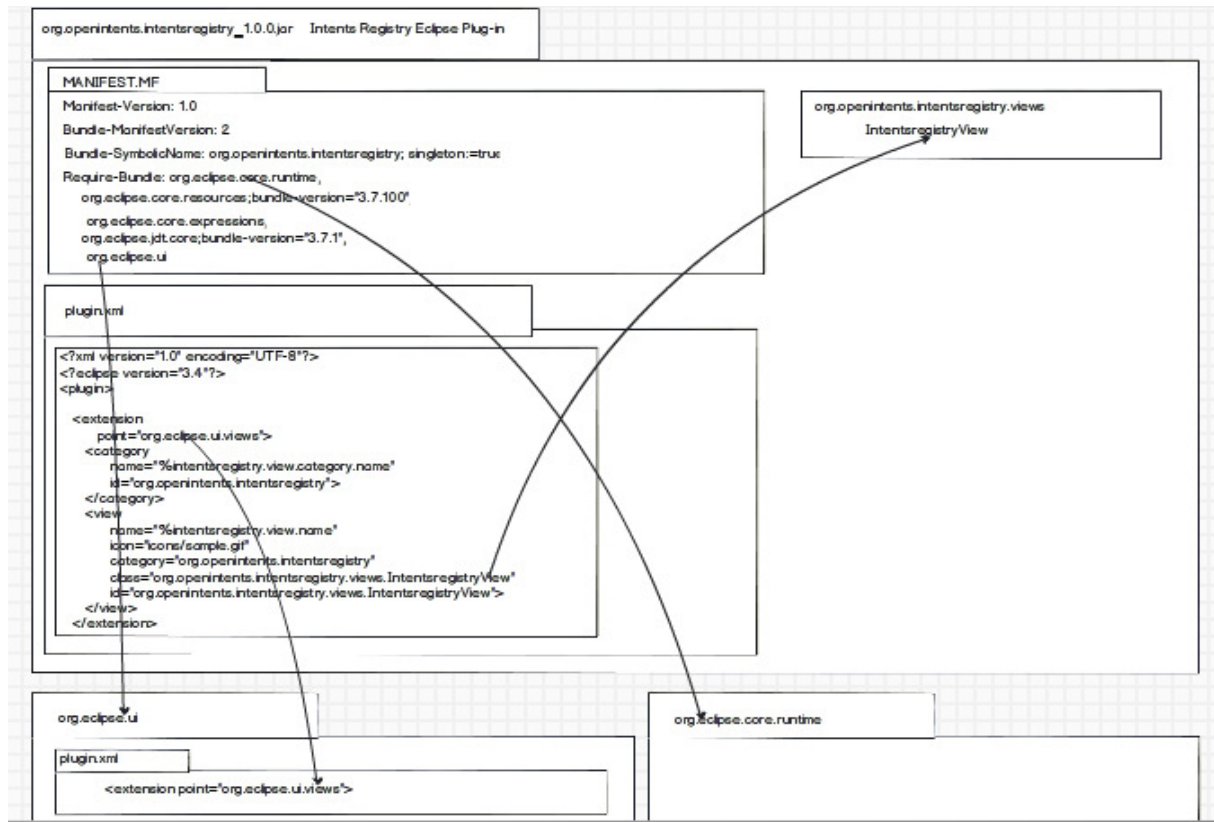


Figure 4. Declaring a new extension

The figure above shows an example of how a new extension is declared in the plug-in manifest with lines highlighting how the plug-in manifest references various plug-in artifacts.

2.1.3 Plug-in Directory or JAR file

The Intents Registry Eclipse plug-in JAR, `org.openintents.intentsregistry_1.0.0.jar`, contains files similar to a typical plug-in, including java class files, various images used by the plug-in and the plug-in manifest.

- Java classes – The classes that are contained within the plug-in can be found in a standard `java` directory within the plug-in JAR file.
- Icons – By default, image files are placed in the `icons` or `images` folder. This is referenced in the `plugin.xml` and by the various plug-in classes.

- MANIFEST.MF – This file contains the runtime information of the plug-in such as the identifier, version, name and other plug-in dependencies.
- Plugin.xml – The extension and extension points are described by this file in XML format.

The plug-in JAR must have a name and placed in a special directory for Eclipse to find and load. The JAR name can be a combination of the plug-in identifier declared in the MANIFEST, an underscore, version in dot separated form, such as:

Org.openintents.intentsregistry_1.0.0.jar

The plug-in JAR is usually placed in a special folder called plugins of the Eclipse installation directory.

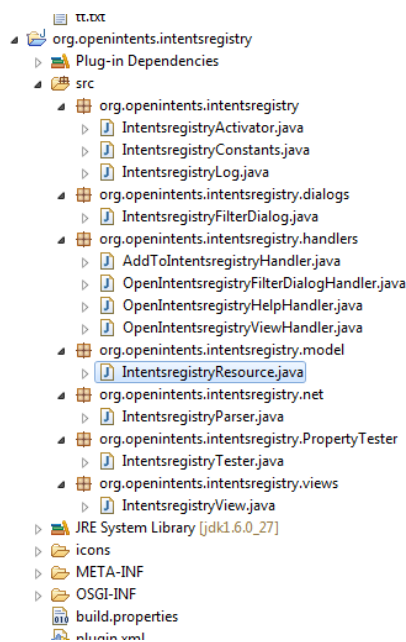


Figure 5: Intents Registry Eclipse plugin project tree

2.1.4 Installing a Plug-in

Plug-ins can be installed in several ways in the Eclipse installation directory:

1. A plug-in can be placed in the plugins directory as siblings to other Eclipse plug-ins.
2. A plug-in can be downloaded to Eclipse from an update site which is usually created by the provider of the plug-in.
3. Downloading Eclipse from the marketplace using the Eclipse marketplace client.
4. Placing the Eclipse plug-in in a product-specific directory and creating a link file for Eclipse to find and load the plug-in.

Plug-in Manifest

For every plug-in, usually there is a MANIFEST.MF and plugin.xml files. Within each manifest, there are usually entries for name, identifier, activator, version and provider.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: %Bundle-Name
Bundle-SymbolicName: org.openintents.intentsregistry; singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: org.openintents.intentsregistry.IntentsregistryActivator
Bundle-Vendor: %Bundle-Vendor
```

Snippet 1. Plugin Manifest

Strings in the manifest file such as the plug-in provider can be externalized by moving it into a build.properties or plugin.properties file. This process enables internalization for the localization of the software in different languages.

```
Bundle-Vendor = OpenIntents
Bundle-Name = Intentsregistry Plug-in
intentsregistry.view.category.name = OpenIntents
intentsregistry.view.name = Intentsregistry
```

Snippet 2. Properties files for org.openintents.intentsregistry

Plug-in identifier

This identifier (Bundle-SymbolicName) is designed to uniquely identify the plug-in within the Eclipse system. Usually it is based on the Java package naming convention, e.g. org.<organization or company name>.<product name>. In this particular case, the identifier is org.openintents.intentsregistry. It is also possible to have several plug-ins in one product. In such a case, then the different plugins can consist of extra parts in addition to the convention mentioned earlier as in org.openintents.intentsregistry.help, org.openintents.intentsregistry.tests, and org.openintents.intentsregistry.features.

Plug-in version, name, and provider

Every plugin must be specified by a version (Bundle-Version) using three digits separated by dots. The first number shows the major version number, the second for the minor version number and the third specify the service level. An example is 1.0.0. Version less than 1.0.0 such as 0.5.0 are typically used for the alpha version of the software and 1.0.0 and above for the beta version. The plug-in name is not unique and is used to represent the name of the plug-in. The provider is a non-unique name often used to describe the company or organization that makes the plug-in.

2.1.5 Extensions and extension points

A plug-in specifies an extension point for other plug-ins to extend its functionality.

```
<extension
  point="org.eclipse.ui.views">
  <category
    name="%intentsregistry.view.category.name"
    id="org.openintents.intentsregistry">
  </category>
  <view
    name="%intentsregistry.view.name"
    icon="icons/sample.gif"
    category="org.openintents.intentsregistry"
    class="org.openintents.intentsregistry.views.IntentsregistryView"
    id="org.openintents.intentsregistry.views.IntentsregistryView">
  </view>
</extension>
```

Snippet 3. Extension points

The attribute defines the extension point in the extension element.

Activator

This is the plug-in class that consists of functions to initialize and access static resources, preferences and state information of the plug-in.

Creating feature project

In order to create an update site project, a feature project is created with settings defined for version, name, provider, update site url and the plug-in id. The id is used to reference the plug-in. Information about feature description; copyright notice and license agreement are defined as well in the feature project.

Creating Eclipse Update site and Project updates site hosting

The update site is created after the feature project. A web server is used to host the contents of the project update site. The update site is located as `gh_pages` in Github. `/39/`. The project page is shown in Figure 9.

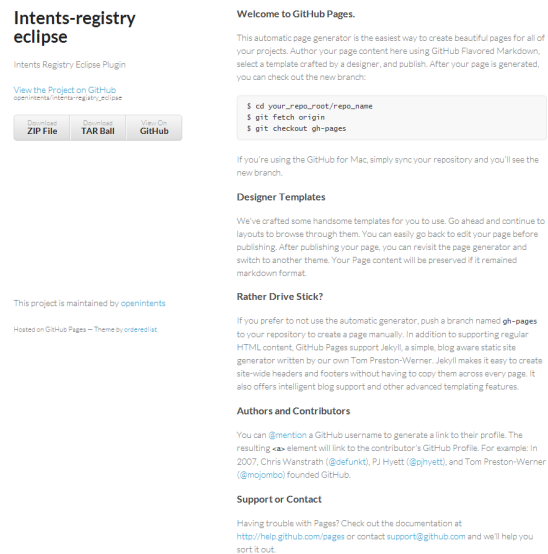


Figure 6. Project page

Code Repository

The code repository for the project can be found on this site shown in Figure 8. /39/

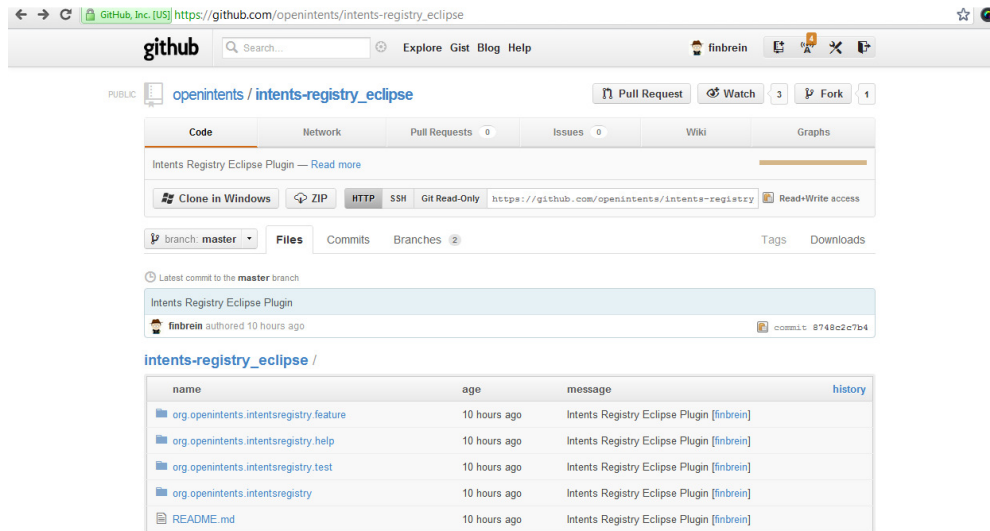


Figure 7. Code repository

2.1.6 Building and Maintaining the Github Update Site

The update site support consists of one or more feature projects and one update site project. The update site build is done by an ant script in the org.openintents.intentsregistry project, called build.xml.

There are three types of information that are kept for a version:

1. The site.xml holds references to the version
2. The jars built for the features
3. The jars for the associated plugin

Steps to create a plugin repository for a new Intents Registry Eclipse Plugin

1. Update the plug-in project in Eclipse
2. Add other plug-in sub-projects to the Github repository
3. Publish the update project to the gh-pages of the master branch of the repository

2.2 Updating the repository

There are several ways to work with git and commit code to OpenIntents Github repository. One way is the local branch to commit process for contributing to the project. The following shows the typical flow. Initialize the local branch for the first time:

```
git init
```

Add the updated source files:

```
git add org.openintents.intentsregistry  
git add org.openintents.intentsregistry.feature  
git add org.openintents.intentsregistry.help  
git add org.openintents.intentsregistry.test
```

Once you are satisfied with the changes, and everything is working, commit those changes to the local branch: The principle here is to “commit early and commit often”

```
git commit a m 'Updated Version Numbers'
```

Update the Github repo

```
git push origin
```

2.3 Internationalization and Help

Eclipse and the JRE expose APIs for GUI and languages from the code. And as the software is being used in different parts of the world with different languages, there is a need to internationalize the plug-in by externalizing the strings and translating them. Eclipse supports multiple languages.

The Eclipse framework was used to include an online help which provides details about the features in the plug-in.

2.4 ANT

Apache ANT serves as a tool to build the software. An ANT build script is written in Eclipse where various attributes are defined within the project element in the XML structure.

2.5 Android and Intents

Intent is an asynchronous message that enables component or code in the same or different applications to interact and request services from each other during runtime. It describes an operation to be performed. An activity component can send one or more intents to the Android system to start another activity. A broadcastreceiver component receives broadcast intents and a service component can be communicated with in the background through the startservice or bindservice intent.

Intent is an object itself and all intents are instances of the android.content.Intent class

Tasks are carried out based on the interaction of several or many intents within or outside an application.

2.5.1 Structure of Intent

The primary pieces of information in intent are:

- action – The general action to be performed, such as ACTION_VIEW
- data – The data to operate on such as person record in the contacts database, expressed as Uri

In addition to this primary attributes, there are a number of secondary attributes that you can also include in intent:

- Category – Gives additional information about the action to execute. For example, CATEGORY_LAUNCHER
- Type – Specifies an explicit type (a MIME type) of the intent data. Normally the type is inferred from the data itself

- Component – specifies an explicit name of a component class to use for the intent.
- Extras – This is a bundle of any additional information. This can be used to provide extended information to the component. For example, if we have an action to send an e-mail message, we could also include extra pieces of data here to supply a subject, body, etc. /34/

Android support explicit and implicit intents

2.5.2 Explicit and Implicit intents

In explicit intents, a component is named which is run by the Android system. It provides the exact class to be called. An example explicit intent in a code snippet to add a new account is shown below:

```
Intent k = new Intent(exampleAccount.this, Account.class);
k.putExtra("example", false);
startActivityForResult(k, 1);
```

Snippet 4. Explicit intent

Implicit intents do not describe the class or component to be called.

```
Intent z = new Intent(Intent.EDIT_VIEW);
z.setAddress(example.getUrl());
startActivity(z);
```

Snippet 5. Implicit intent

2.5.3 Open Intents

They facilitate interactions within an application as well as interactions with other applications. This intent can be used by other applications, or use the intents of other applications. OpenIntents provides several reusable components including OI About which displays a general about box for author information, OI File Manager which provides “Open” and “Save as” dialogs; OI Safe which encrypt strings with a single master password and OI Update which set up version checking other than the Google play. /40/

A developer can use these components in their application through intents. For example, the About dialog can be used in a developer application by calling the `SHOW_ABOUT_DIALOG` intent:

```
Intent intent = new Intent("org.openintents.action.SHOW_ABOUT_DIALOG");
startActivityForResult(intent, 0); //44/
```

Snippet 6. An Open Intent

2.5.4 Open Intents registry

The registry provides the following elements for extracting information about the various intents which can be implemented or called by other activities.

- Action
- Additional_extra_keys
- Basic_URI
- Category
- Extra_key
- Input
- Intent_Extra
- MIME_types
- Output
- Provides_intents_URIs
- URI_variants

The open intents registry is an xml file as shown below in Figure 9. /41/

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼ <xml>
  ▼ <node>
    <Action>android.intent.action.VIEW</Action>
    <Additional_extra_keys/>
    <Basic_URI/>
    <Category/>
    <Extra_key/>
    <Input>Uri of the data to be displayed</Input>
    <Intent_Extra/>
    <MIME_types/>
    <Output>None</Output>
    <Provides_intent_URIs/>
    <URI_variants/>
    ▼ <Body>
      <p>Display the data to the user.</p> <p>An activity implementing this action will display to the user the<br /> given data. It is not intended for the user to edit it, though an<br /> implementation can allow that to be done as a secondary part of its UI<br /> if desired (as it can provide any other optional features in its UI<br /> for the user).</p> <p>This is the most common action performed on data -- it is the generic action you can use on a piece of data to get the most reasonable thing to occur. For example, when used on a contacts entry it will view the entry; when used on a mailto: URI it will bring up a compose window filled with the information supplied by the URI; when used with a tel: URI it will invoke the dialer. </p> <p>See <a href="http://code.google.com/android/reference/android/content/Intent.html#ACTION_VIEW">Android documentation</a></p>
    </Body>
    <title>View data</title>
  </node>
  ▼ <node>
    <Action>android.intent.action.EDIT</Action>
    <Additional_extra_keys/>
    <Basic_URI/>
    <Category/>
    <Extra_key/>
    <Input>URI of the data to be edited.</Input>
    <Intent_Extra/>
    <MIME_types/>
    <Output>None</Output>
    <Provides_intent_URIs/>
    <URI_variants/>
    ▼ <Body>
      <p>Provide explicit editable access to the given data.</p> <p>Activity implementing this intent provides explicit editable access to the given data.
    </Body>
  </node>
</xml>
```

Figure 8. Open Intents online registry

2.5.5 Intent Filters

An intent filter wraps around one or more pieces of information in an intent. It specifies the intent type that an activity, service or broadcast receiver can interact with. These filters are defined within the AndroidManifest.xml file.

2.5.6 Android Manifest File

Android applications typically have this file which provides vital information about the application. The package name for example serves as the unique identifier for the application. It also contains component information for components like activities, services, broadcast receivers and content providers. For example, in snippet 7 of the AndroidManifest.xml for a Guestbook, there is an activity containing an intent-filter. This filter contains actions like “android.intent.action.MAIN” and “android.intent.category.LAUNCHER” classes. These are the kinds of actions being referred to in the open intents registry. /45/

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.finbrein.guestbook.android.client"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="10" />
    <uses-feature android:name="android.hardware.camera" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name="org.finbrein.guestbook.android.client.Guestbook"
            android:label="@string/app_name"
            android:configChanges="orientation\keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>

```

Snippet 7. Android Manifest file

2.5.7 Android Manifest Editor

Eclipse has a default text or structured text editor for files with .xml extension. During the installation of Android Development Tools (ADT) plugin into the Eclipse IDE, this setting is changed for AndroidManifest.xml file to use the visual Android manifest editor as the default.

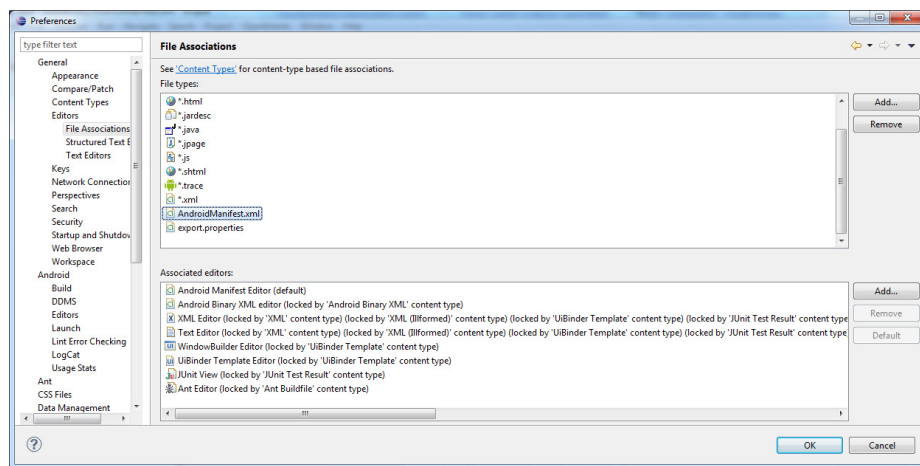
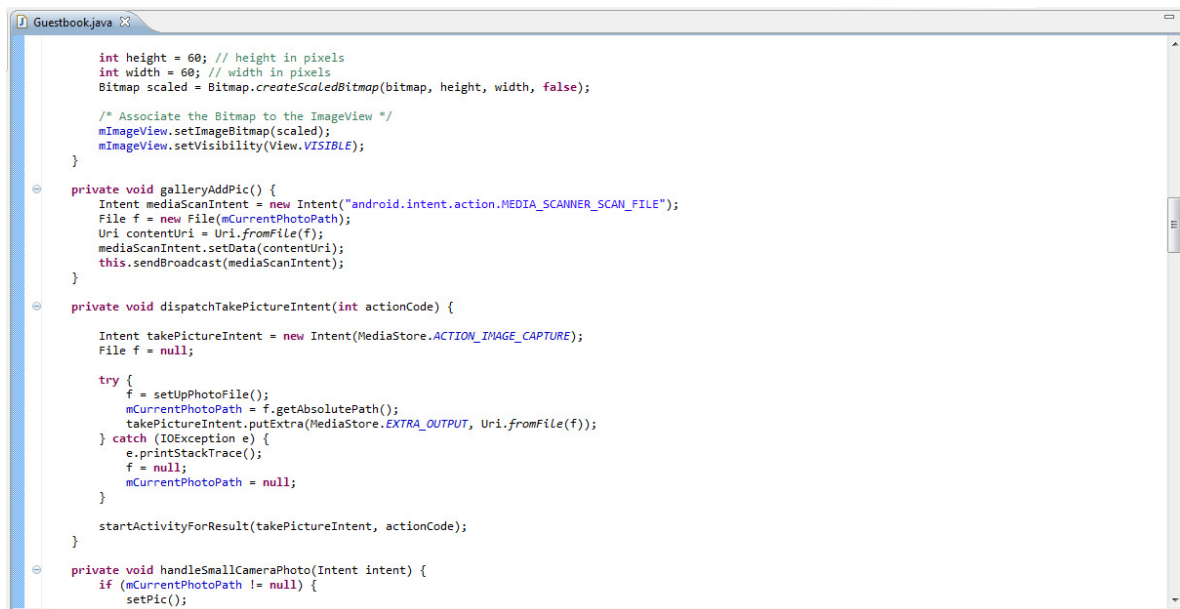


Figure 9. Preferences dialog showing file association between editor and file types

2.5.8 Default Java Editor

Unlike the Android manifest editor, the default text editor is text based. Actions, uris, extras and other pieces of information that describes intent can be added to the default editor while actions can be added to the Android manifest file. The figure below shows a default java editor with intent and an action with the name “android.intent.action.MEDIA_SCANNER_SCAN_FILE”.



```
Guestbook.java x
int height = 60; // height in pixels
int width = 60; // width in pixels
Bitmap scaled = Bitmap.createScaledBitmap(bitmap, height, width, false);

/* Associate the Bitmap to the ImageView */
mImageView.setImageBitmap(scaled);
mImageView.setVisibility(View.VISIBLE);
}

private void galleryAddPic() {
    Intent mediaScanIntent = new Intent("android.intent.action.MEDIA_SCANNER_SCAN_FILE");
    File f = new File(mCurrentPhotoPath);
    Uri contentUri = Uri.fromFile(f);
    mediaScanIntent.setData(contentUri);
    this.sendBroadcast(mediaScanIntent);
}

private void dispatchTakePictureIntent(int requestCode) {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    File f = null;

    try {
        f = setUpPhotoFile();
        mCurrentPhotoPath = f.getAbsolutePath();
        takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(f));
    } catch (IOException e) {
        e.printStackTrace();
        f = null;
        mCurrentPhotoPath = null;
    }

    startActivityForResult(takePictureIntent, requestCode);
}

private void handleSmallCameraPhoto(Intent intent) {
    if (mCurrentPhotoPath != null) {
        setPic();
    }
}
```

Figure 10. Default editor

3 THE APPLICATION DESCRIPTION

The main functionality of this application is to integrate the online intents registry with the Eclipse IDE and provide a mechanism to search through the intents as well as add intents to the editors in the IDE. In order to achieve this, a plug-in written using Java and Eclipse API is required. This plug-in provides a dialog box, menus and views on top of the Eclipse platform. Libraries from the Java software development kit, `org.eclipse.jface.text`, `org.eclipse.ui.texteditor`, `org.eclipse.ui`, `org.eclipse.swt`, `org.eclipse.core.runtime`, `org.eclipse.core.resources`, etc. were referenced. The following requirements were gathered for the project.

3.1 Application Requirements

Requirement analysis consist of the applications must have, should have and nice to have.

The application must have

- A persistence layer (a text file) - the registry data should be persisted on the client machine and available to Eclipse even when there is no internet access to the online intents registry.
- A search function for intents - Create a dialog box for displaying titles or descriptive names of intents and for adding actions to the code in the programmer's editor.
- A browser within Eclipse
- Eclipse 3.x and 4.x compatible - the Intents Registry plugin for the Eclipse IDE has never existed for this popular IDE. Eclipse has a very robust API that allows developers to design and develop plugins.
- Multi-platform support - The plug-in is compatible with all operating systems that support Eclipse. Most important operating systems targeted are Linux, Mac OS X and Windows.
- Synchronization - Local storage of intents should be in sync with the online intents registry.

The application should have

- Insert a code block (intents, extras, interfaces, etc.) into the Eclipse editor
- Intents accessible via a menu

The application is nice to have

- Intents accessible via a context menu
- Intents accessible via a code completion feature
- A no-sql database (couch DB) for persistence
- Insert a code block in the Android manifest file
- A help function

3.2 Functional Specification

An OpenIntents or Android developer can access the online intents registry via browsing or searching the data within Eclipse. The intents, extras and action data stored in a text file within Eclipse can be updated automatically via internet connection and users can still have access to the data when there is no internet access. The following use case diagram documents the use cases performed by the developer of Android application using the intent registry plugin for open intents, libraries, uris, extras and other types of artifacts within Eclipse. Three main functions including the search function, browsing and inserting code blocks into the android developer editor.

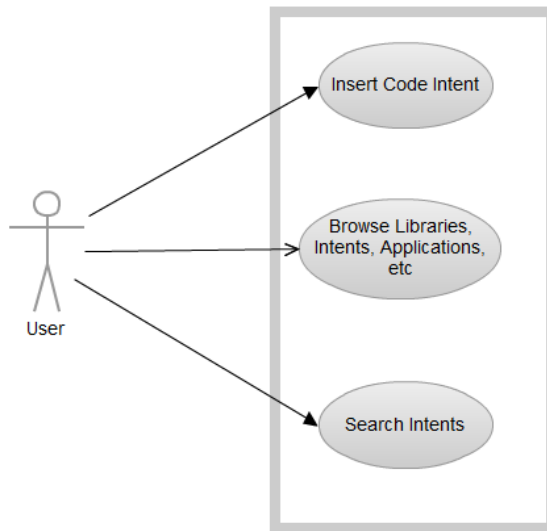


Figure 11. Use case diagram of the plugin

3.3 Class Hierarchy

The application consists of several packages such handlers, net, views, dialogs, model, intentsregistry and property tester. Each package consists of classes that define and implement behaviour that are common to the classes. Some classes derive from other classes forming a hierarchy of classes.

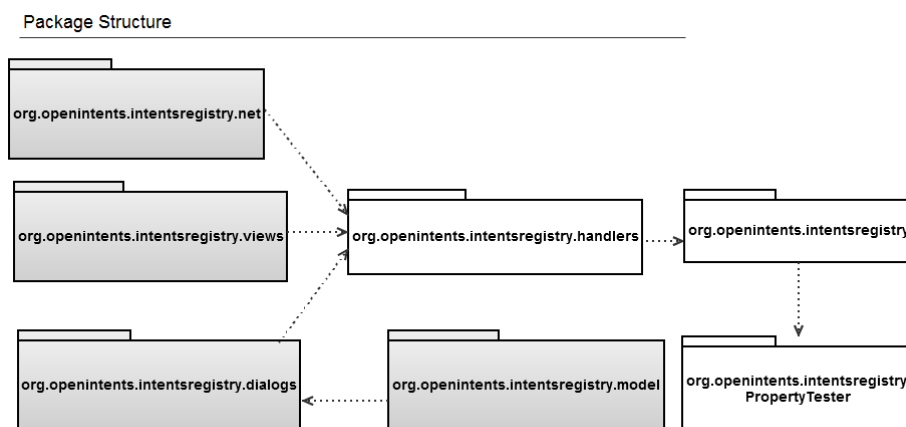


Figure 12. A package diagram of the plugin showing the relationships among the packages

3.3.1 Main UI Classes

The IntentsregistryView class defines the view for displaying the browser. It receives user interactions and implements appropriate changes. The most important functions within the class are highlighted below:

- createPartControl: This is a callback that will allow the creation of the viewer and initialize it
- setFocus: pass the focus request to the viewer's control
- IntentsregistryView: The constructor

The structure of these functions can be visualized in the figure below.

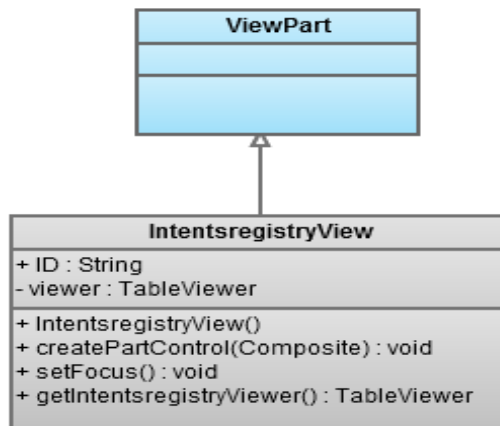


Figure 13. Structure of the main view

The IntentsregistryFilterDialog class implements a filtered selection dialog and is the main class for searching and filtering intents. The class is responsible for displaying a set of items to be selected by the user. The IntentsregistryResourceSelectionHistory, ResourceFilter and ShowOnlyLowerCaseStringsAction are enclosed within the IntentsregistryFilterDialog. Main functions of the class include:

- createExtendedContentArea: creates an extended content area located above the details
- createFilter: creates an instance of a filter

- fillContentProvider: fills content provider with matching items
- getDialogSettings: returns the dialog settings
- getItemsComparator: returns the comparator to sort items inside the content provider
- validateItem: validates the item
- fillViewMenu: Fills the menu of the dialog
- applyFilter: applies the filter created by createFilter() method to the items list
- getElementByName: returns the name of the given object

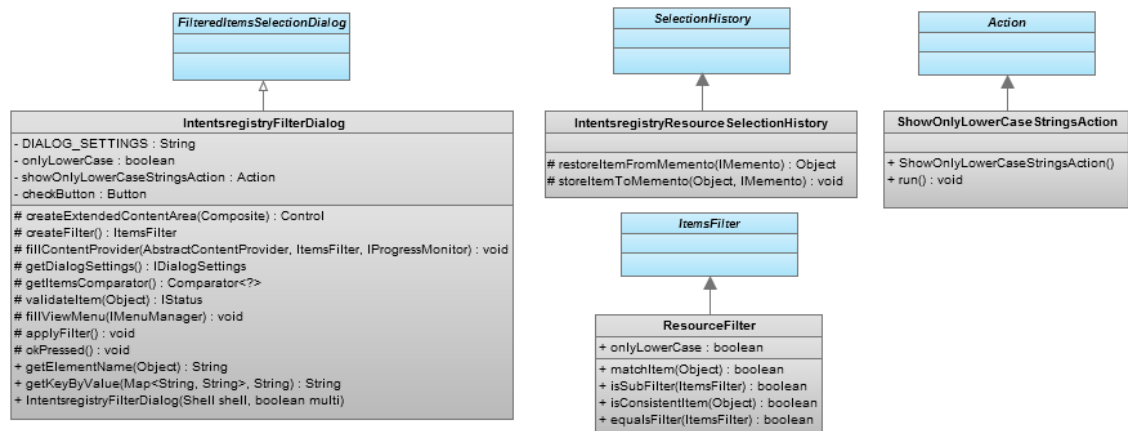


Figure 14. Filter dialog classes

3.3.2 Handler Classes

The handler classes receive, send and process messages. The `OpenIntentsregistryFilterDialogHandler` class open the intents registry filter dialog. The `OpenIntentsregistryViewHandler` open the Intentsregistry view while the `OpenIntentsregistryHelpHandler` show help for the intents registry. The most notable function common to these handler classes is the `execute` method stated below:

Execute: executes with the map of parameter values by name

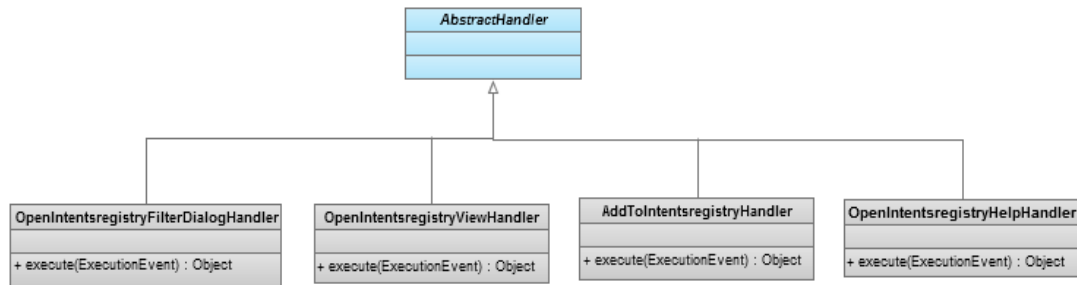


Figure 15. Handler classes

3.3.3 Plugin Activator and Logging Classes

The application lifecycle starts and stops within the `IntentsregistryActivator`. It is the main entry point into the program. It exploits interfaces and functionalities provided by the plug-ins and libraries. The `IntentsregistryLog` is responsible for collecting log information about the plug-in. This is necessary for debugging in case something goes wrong. The main attributes of the `IntentsregistryActivator` class are

- `PLUGIN_ID`: the plug-in id
- `Plugin`: the shared instance

The main functions include:

- `Start`: method is called when plug-in is started

- Stop: method is called when plugin is stopped
- getDefault: returns the shared instance
- getImageDescriptor: returns an image descriptor for the image file at the given plug-in relative path
- getConfigDir: answer the configuration location for this plug-in
- saveConfigPrefs: save the configuration preferences if they have been loaded

The most notable functions of the IntentsregistryLog class are:

- logInfo: log the specified information
- logError: log the specified error
- createStatus: create a status object representing the specified information
- log: log the given status

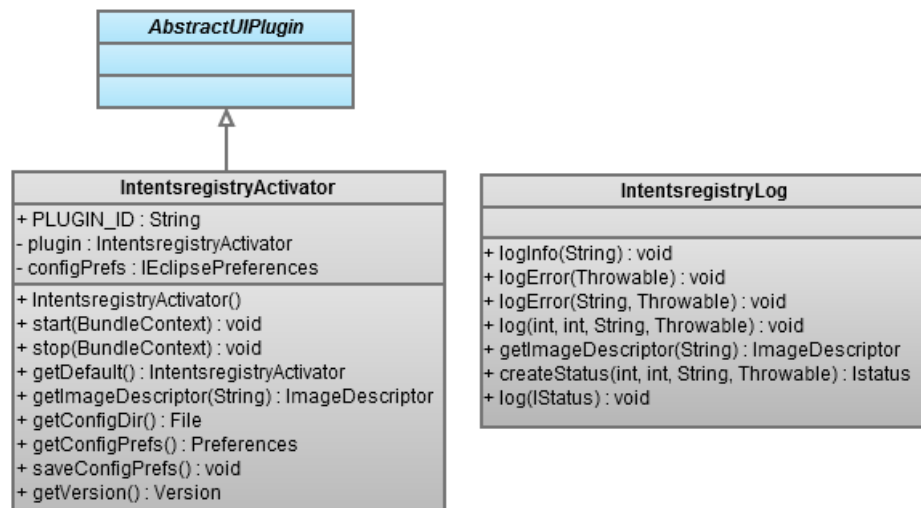


Figure 16. Plugin Activator and logging classes

3.3.4 Model and Parser Classes

The IntentsregistryParser parses the input source file and collects the information into a file while the IntentsregistryResource generates data for the IntentsregistryFilterDialog object. Notable functions within the classes include:

- List: creates a list of items received from the online intents registry xml file
- generateResources: read the list of items and store them in a text file as “=” delimited text

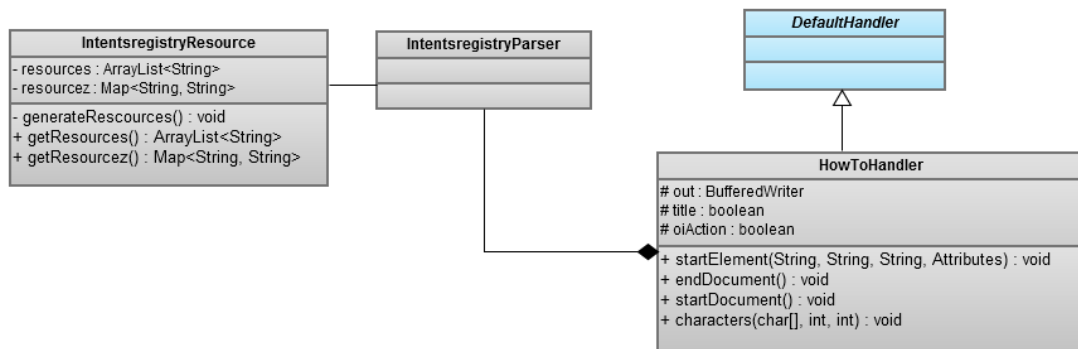


Figure 17. Parser and model classes

3.4 Description of Functions

This section provides a detailed description of the main functionality of the application. The use case diagram will be analysed and given a detailed description. The implemented classes for these functions are explained in subsequent chapters.

3.4.1 Insert Code Block

A user clicks the OpenIntents menu and chooses Add Intent... to open a dialog box. The user enters a search item to display a set of filtered items in the menu item box.

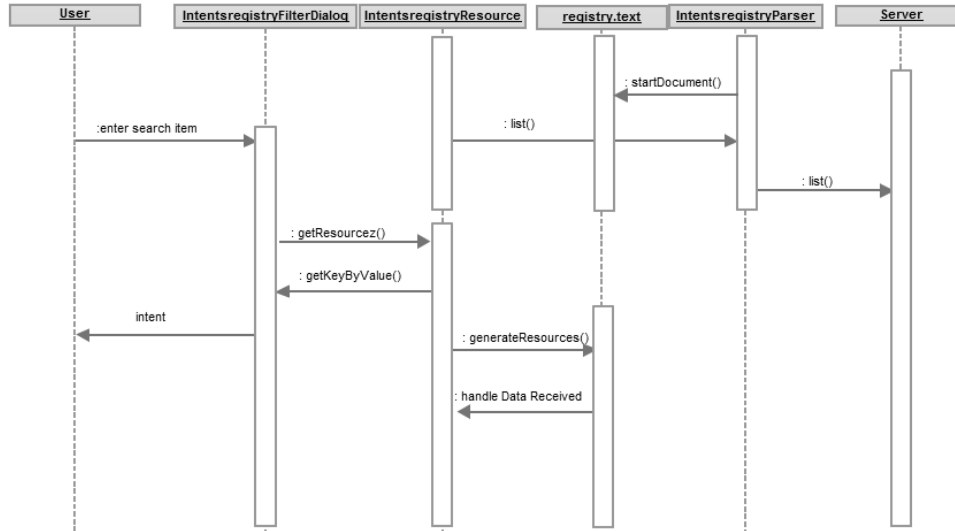


Figure 18. Sequence diagram – add intent

The sequence of actions is described as follows

- User enters a search item
- Item is analysed and filtered intents are sent to the dialog box
- User selects the desired intent and click the OK button
- Intent is added to the code editor of the IDE

3.4.2 Browse Intent

To view a page, a user has to open the OpenIntents menu and select Browse online to open the browser window. The user enters a url to fetch the page.

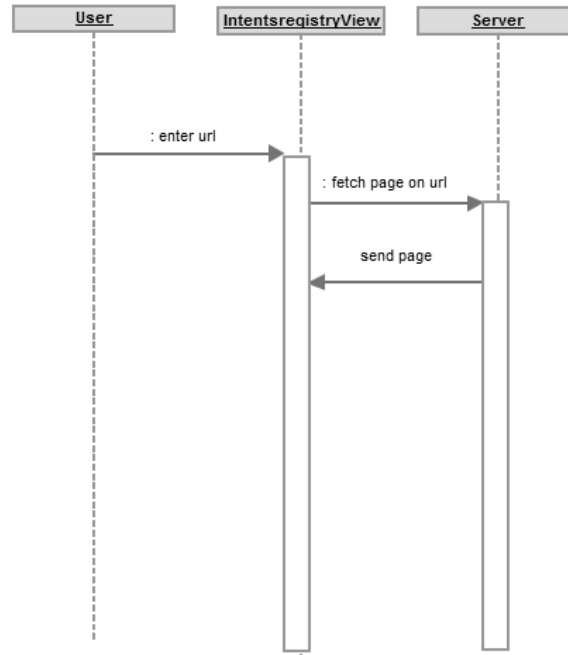


Figure 19. Sequence diagram – Browse intent

The sequence of actions is described as follows

- User enters a url in the address bar of the browser window
- If the page is available, then it is displayed to the user

3.5 Deployment Diagram

The architecture of the application can be described below. The intents are stored in an xml file on the OpenIntents web server while Eclipse is the client pulling data from the external source and storing it within a text file. The content of the text file is processed and provided as application data to Eclipse. The figure below shows the deployment diagram for the plug-in using HTTP client to communicate with the web server containing the online intents.

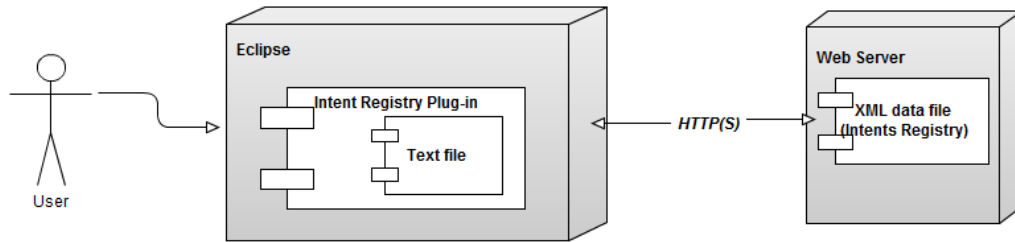


Figure 20. Deployment diagram for the plugin

4 DATA PERSISTENCE AND GUI DESIGN

This overview is intended to give a very high-level introduction of the design methodologies, interesting and important design decisions about the Intents Registry Eclipse plugin, key model and components of the plugin, how they work individually and how it all fits together.

Persisting and retrieving data within Eclipse can be done in various ways by storing it in a file, database or caching network data. To store the data, a flat file data source of type text was used. The intents are stored in a text file for easy retrieval rather than using an xml file. Using the text file is more suitable for the application. The plug-in uses a simple document model for data storage. To address the problem of efficient retrieval of data from the storage, a text file was used over XML. Reading data from an xml file takes longer than a text file since every path to a single data will be traversed from the root element. Our data model is simple enough to require a simple database for building this application.

The main GUIs are the filtered selection dialog box and the browser box which displays the intents and the window for browsing online respectively.

4.1 Main GUI

The user has to select the “Add Intent...” menu from the Eclipse menu bar to have access to the intents, extras, actions, interfaces as well as other application data. Below is a design of the filtered selection dialog created with the Balsamiq mockup tool. A user enters an item to open within the matching items box. The matching items box displays a list of items matching the text entered.

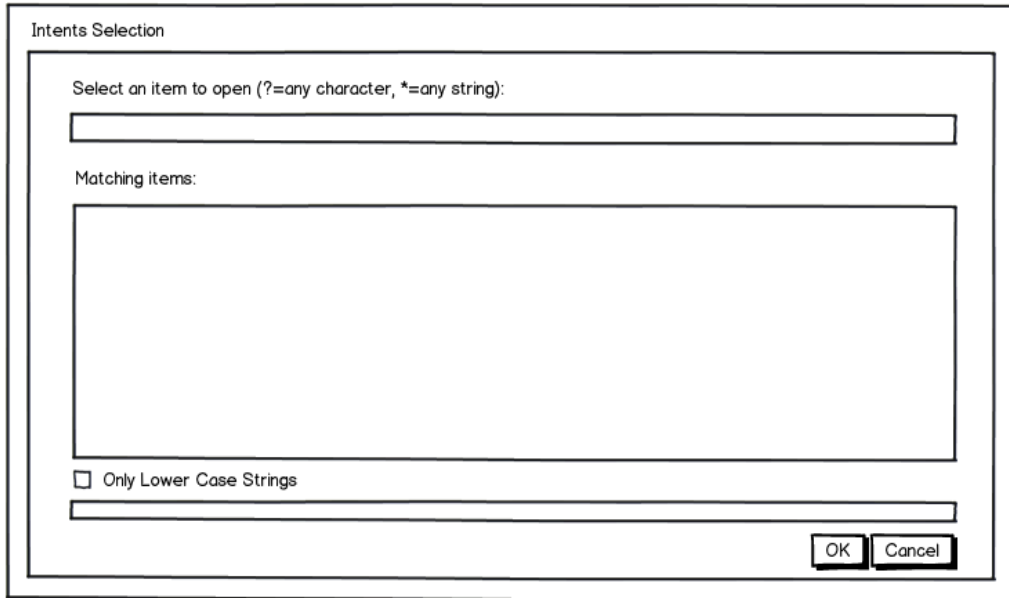


Figure 21. Add Intent

4.2 Browser Window

The browser window is used to connect the openintents website and browse the site. Items can be downloaded or copied into Eclipse editor like as obtainable in a typical browser.

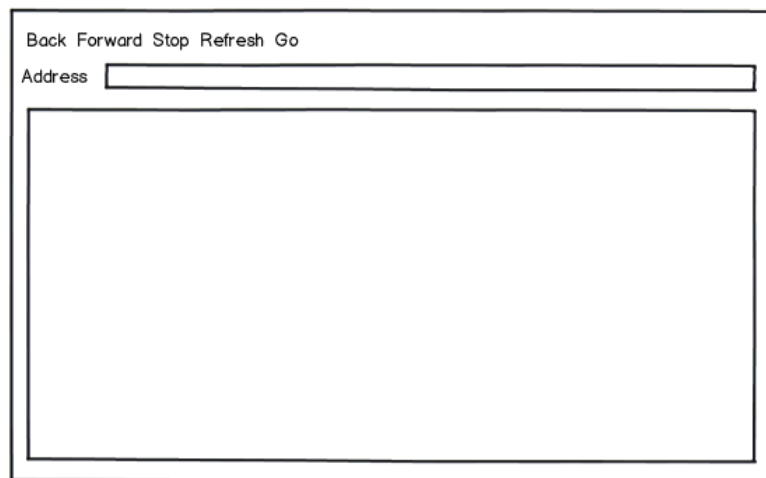


Figure 22. Browse Intent

5 IMPLEMENTATION

The plug-in is built on Eclipse platform by extending Eclipse classes via their interfaces using the Java language. The application was developed for real-time with an emphasis on reliability and availability. Details of the application can be found in the comments of the attached source codes for the application.

5.1 Software Methodology

Extreme Programming was applied for developing the software. This is necessary in order to deliver working prototypes of the software quickly and make corrections to bugs and bad design ideas before moving on. Since the application is quality driven based on test-driven development (TDD), it is necessary that written codes are tested before or during the development. Several spike solutions were created to figure out answers to the technical and design problems of the project. This actually involved two people working in pair.

5.2 Data Retrieval and Persistence

To get data into the file from the XML data source received by the HTTP client, a decision was made between SAX and DOM. DOM (Document Object Model) describes the XML file as hierarchy and which is stored in memory. It is resource intensive. Therefore, SAX, a simple API for XML was chosen and applied. Below is one of the several nodes or tags in the registry.xml file.

```
<node>
  <Action>com.mgeek.android.musicbox.PLAYBACK_VIEWER</Action>
  <Additional_extra_keys/>
  <Basic_URI/>
  <Category/>
  <Extra_key/>
  <Input/>
  <Intent_Extra/>
  <MIME_types/>
  <Output/>
  <Provides_intent_URIs/>
  <URI_variants/>
  <Body>
    <p>Shows a player, this is not really a protocol, maybe the author of the app can tell us more.</p>
  </Body>
</node>
```

```
</Body>
<title>Playback Music</title>
</node>
```

Snippet 8. Intent tag

The parser interface, org.xml.sax.XMLReader interface is the key for parsing the xml based registry. It hides the complexity of different input formats and parsing libraries while providing a simple and powerful mechanism for client applications to extract structured text content and metadata. All this is achieved with a single method. Below is the method extracted from the code snippet below to pull data from the xml data source on the web server.

```
public abstract void parse(InputSource paramInputSource)
    throws IOException, SAXException;
```

Snippet 9. Parse method

The parse method takes the source to be parsed as inputs and output the results as SAX events which are sent as callbacks to the methods of a ContentHandler instance given to the parse method:

Once an instance of the class, XMLReader in the list() of the code snippet below is obtained, XML can be parsed from a variety of input sources. These input sources are InputStreams, Files, URLs, and SAX InputSources. /3/

The list() method will iterate through the <node> tags and read the content of each of them. Only the <title> and the <action> tags of each <node> tag will be read as defined in the code and stored in the file using the startDocument() function.

```
public class IntentsregistryParser {
    class HowToHandler extends DefaultHandler {
        BufferedWriter out = null;
        boolean title = false;
        boolean oiAction = false;
        public void startElement(String nsURI, String strippedName, String tagName,
Attributes attributes) throws SAXException {
            if (tagName.equalsIgnoreCase("title"))
                title = true;
            if (tagName.equalsIgnoreCase("Action"))
                oiAction = true;
        }
    }
}
```

```

    @Override
    public void endDocument() throws SAXException {
        try {
            out.close();
        } catch (IOException e) {
            IntentsregistryLog.logError(e);
        }
    }

    @Override
    public void startDocument() throws SAXException {
        try {
            out = new BufferedWriter(new
FileWriter(IntentsregistryConstants.PROPERTY_DEFAULT_REGISTRY_FILE));
        } catch (IOException e) {
            IntentsregistryLog.logError(e);
        }
    }

    public void characters(char[] ch, int start, int length) {
        try {
            if (oiAction) {
                out.write(new String(ch, start, length));
                oiAction = false;
            }
            else if (title) {
                out.write("=" + new String(ch, start, length) + "\n");
                title = false;
            }
        } catch (IOException e) {
            IntentsregistryLog.logError(e);
        }
    }
}

public void list( ) throws Exception {
    XMLReader parser = XMLReaderFactory.createXMLReader();
    parser.setContentHandler(new HowToHandler( ));
    parser.parse(new InputSource(new
URL(IntentsregistryConstants.DEFAULT_BASE_URL).openStream()));
}
}

```

Snippet 10. IntentsregistryParser

5.3 Generating Resources

A simple text file is used to store the xml data retrieved from the intents registry xml data source. This file is regularly updated from the web server. To get data out the file from the file system into Eclipse, you go through BufferedReader object.

```
public class IntentsregistryResource {

    private static ArrayList<String> resources = new ArrayList<String>();
    static Map<String, String> resourcez = null;
    static {
        generateResources();
        IntentsregistryParser intentList = new IntentsregistryParser();
        try {
            intentList.list( );
        } catch (Exception e) {
            IntentsregistryLog.logError(e);
        }
    }
    private static void generateResources() {
        try {
            BufferedReader in = new BufferedReader(new
            FileReader(IntentsregistryConstants.PROPERTY_DEFAULT_REGISTRY_FILE));
            String str = "";
            resourcez = new HashMap<String, String>();
            while ((str = in.readLine()) != null) {

                resources.add(str.substring(str.indexOf(IntentsregistryConstants.EQUAL_DELIMITER) + 1,
                str.length()));

                resourcez.put(str.substring(0,
                str.indexOf(IntentsregistryConstants.EQUAL_DELIMITER)),
                str.substring(str.indexOf(IntentsregistryConstants.EQUAL_DELIMITER)+1, str.length()));
            }
            in.close();
        } catch (IOException e) {
            IntentsregistryLog.logError(e);
        }
    }

    // @return the resources
    public static ArrayList<String> getResources() {
        return resources;
    }

    /**
     * @return the resourcez
     */
    public static Map<String, String> getResourcez() {
        return resourcez;
    }
}
```

Snippet 11. IntentsregistryResource

The content of the registry file in the plugin is shown below in Figure 11.

```
android.intent.action.VIEW=View data
android.intent.action.EDIT=Edit data
android.intent.action.ALL_APPS=List all applications
android.intent.action.DELETE=Delete data
android.intent.action.DIAL=Dial a number
org.openintents.action.TAG=Tag data
com.google.android.radar.SHOW_RADAR=Show radar
android.intent.action.PICK=Pick data
android.intent.action.INSERT=Insert data
android.intent.action.MAIN=Main
android.intent.action.GET_CONTENT=Get content
android.intent.action.SEARCH=Search
com.android.notepad.action.EDIT_TITLE=Edit title
android.intent.action.CREATE_SHORTCUT=Create shortcut
android.intent.action.SET_WALLPAPER=Set wallpaper
android.intent.action.RINGTONE_PICKER=Ringtone picker
com.google.zxing.client.android.SCAN=Scan
com.google.zxing.client.android.ENCODE=Encode
org.openintents.intents.UNRESOLVED_INTENT=Resolve unresolved intents
org.theb.ssh.action.CONNECT_HOST=Connect host
com.google.android.photostream.FLICKR_STREAM=Flickr stream
com.google.android.photostream.FLICKR_PHOTO=Flickr photo
com.borntotinker.intent.action.MEASURE=Environmental measurement
android.provider.MediaStore.RECORD_SOUND=Record sound
android.intent.action.SEND=Send data to someone
android.intent.action.SENDTO=Send a message to the given uri
org.openintents.intents.CHECK_VERSION=Check for Update
android.media.action.IMAGE_CAPTURE=Capture an image
org.openintents.action.PICK_FILE=Pick file
org.openintents.action.PICK_DIRECTORY=Pick directory
com.google.zxing.client.android.SEARCH_BOOK_CONTENTS=Search book contents
com.google.zxing.client.android.SHARE=Share (through QR code)
android.intent.action.WEB_SEARCH=Web search
android.intent.action.CALL=Call
com.twidroid.SendTweet=Send Twitter Message
org.openintents.action.SHOW_ABOUT_DIALOG=Show about dialog
org.openintents.action.CALCULATOR=Calculator
com.eclipsim.gpsstatus.VIEW=Show GPS Status
com.roozen.intent.VOLUME_CONTROL=Volume Control
com.roozen.intent.SOUND_CONTROL=Sound Control
```

Figure 23. Local registry of Intent actions and titles

5.4 View and Dialog Handler

There are two main user interfaces in this application with behaviour defined in the `IntentsregistryView` and `IntentsregistryFilterDialog` classes. The `IntentsregistryView` is responsible for displaying the browser while the `IntentsregistryFilterDialog` is responsible for displaying the dialog box containing filtered intents. In order to have the filter dialog appear, an handler method to open the `IntentsregistryFilterDialog` is defined below:

```
public Object execute(ExecutionEvent event) throws ExecutionException {
    new IntentsregistryFilterDialog(HandlerUtil.getActiveShell(event),
        isEnabled()).open();
    return null;
}
```

Snippet 12. Execute method of Filter Dialog

The method to open the IntentsregistryView is as follows:

```
/**
 * Open the intentsregistry view.
 */
public Object execute(ExecutionEvent event) throws ExecutionException {

    // Get the active window
    IWorkbenchWindow window =
HandlerUtil.getActiveWorkbenchWindowChecked(event);
    if (window == null)
        return null;

    // Get the active page
    IWorkbenchPage page = window.getActivePage();
    if (page == null)
        return null;

    // Open and activate the Intentsregistry view

    try {
        page.showView(IntentsregistryView.ID);
    } catch (PartInitException e) {
        IntentsregistryLog.logError("Failed to open the Intentsregistry view", e);
    }
    return null;
}
```

Snippet 13. Execute method of IntentsregistryView

5.5 Logging

Exceptions and service related information are appended to a log file. The activator has a `getLog()` method for accessing the plug-in logging mechanism.

Several methods are used to log the error messages and exceptions to the log for the Intents Registry Eclipse plugin.

```
public static void logInfo(String message) {  
    log(IStatus.INFO, IStatus.OK, message, null);  
}  
  
public static void logError(Throwable exception) {  
    logError("Unexpected Exception", exception);  
}  
  
public static void logError(String message, Throwable exception) {  
    log(IStatus.ERROR, IStatus.OK, message, exception);  
}
```

Snippet 14. Logging

6 TESTING

In order to have a reliable and robust plug-in, there is a need to for testing the code. New features will always be added to the project; therefore, there is a need to test and add tests as the project evolve and to ensure that the product keeps functioning properly over many releases. This section contains all testing information including unit tests and user testing.

The JUnit library was added to the test project as a dependency for testing and writing the unit tests. These are small units of codes that are designed to be efficient. Each method validate one logical piece of code. Methods within the test classes have an Assert.* call in them.

6.1 JUnit Testing

JUnit is a simple, open source framework to write and run repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. JUnit features include:

- Assertions for testing expected results
- Test fixtures for sharing common test data
- Test runners for running tests. /42/

Snippet shows the test written to assert that the IntentsregistryView resource object will contain a list of intents received from the server.

```
public class IntentsregistryResourceTest {  
  
    @Test  
    public void testGetResources() {  
        assertEquals( 84, IntentsregistryResource.getResources().size() );  
    }  
  
    @Test  
    public void testGetResourcez() {  
        Map<String, String> resourceMap = IntentsregistryResource.getResourcez();  
        assertNotNull( resourceMap.size() );  
    }  
}
```

Snippet 15. RegistryResource JUnit Test Case

6.2 SWTBOT

SWTBot is an open-source java based UI/functional testing tool for testing SWT and Eclipse based applications.

SWTBot provides APIs that are simple to read and write. The APIs also hide the complexities involved with SWT and Eclipse. This makes it suitable for UI/functional testing by everyone, not just developers. SWTBot also provides its own set of assertions that are useful for SWT. You can also use your own assertion framework with SWTBot.

SWTBot can record and playback tests and integrates with Eclipse, and also provides for ant tasks so that you can run your builds within CruiseControl or any other CI tool that you use.

SWTBot can run on all platforms that SWT runs on. Very few other testing tools provide such a wide variety of platforms. /43/

Snippet shows the test written to test an SWT based GUI.

```
public class IntentsregistryViewTest {  
  
    private static SWTWorkbenchBot bot;  
  
    @BeforeClass  
    public static void beforeClass() throws Exception {  
        bot = new SWTWorkbenchBot();  
        bot.viewByTitle("Welcome").close();  
    }  
  
    @Test  
    public void testCreatePartControlComposite() {  
        // Get text after the label "Address"  
        SWTBotText textWithLabel = bot.textWithLabel("Address");  
  
        // Set the focus and write a text into the text field  
        textWithLabel.setFocus();  
  
        textWithLabel.setText("my text");  
        assert(textWithLabel.getText().equals("my text"));  
  
        assertTrue(true);  
    }  
}
```

```

    @Test
    public void canOpenABrowser() throws Exception {
        bot.menu("OpenIntents").menu("Browse online").click();
    }
}

```

Snippet 16. IntentsregistryView Test Case

6.3 Inserting Code Blocks

The plug-in defines an Intents Registry which allows users to insert code blocks within the Eclipse IDE. Appendix 1 contains an installation and user manual for the tool. In this section, we will describe how the tools work without mentioning specific details. The picture in Figure shows the dialog box for searching, filtering and adding intents to the editor in Eclipse.

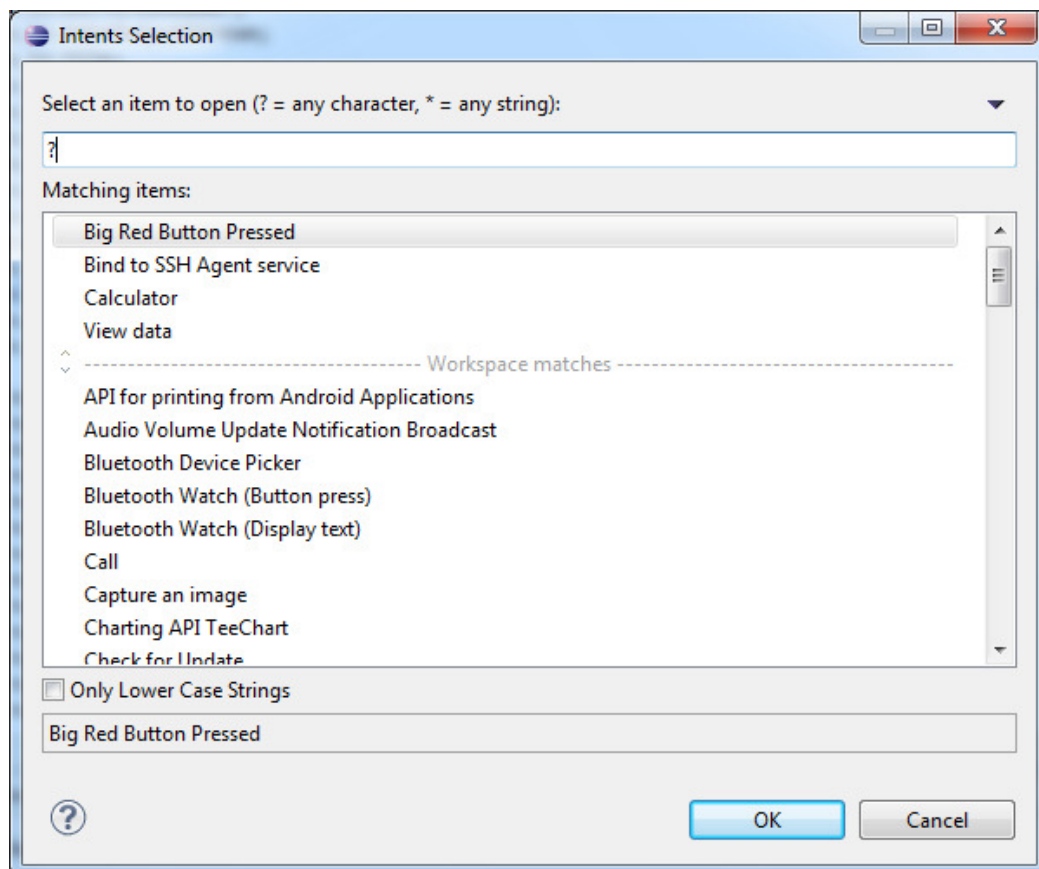


Figure 24. Intent selection page

The process starts with clicking on the OpenIntents menu and selecting “Add Intent...”. This will fire up the handler which calls the dialog box pre-populated with data.

6.4 Browsing Libraries and Artifacts

The intents registry view below allows the developers to browse the libraries and other artifacts. Buttons are added to allow the go to the forward or back page. The use can also type urls to the address bar to browse different sites.

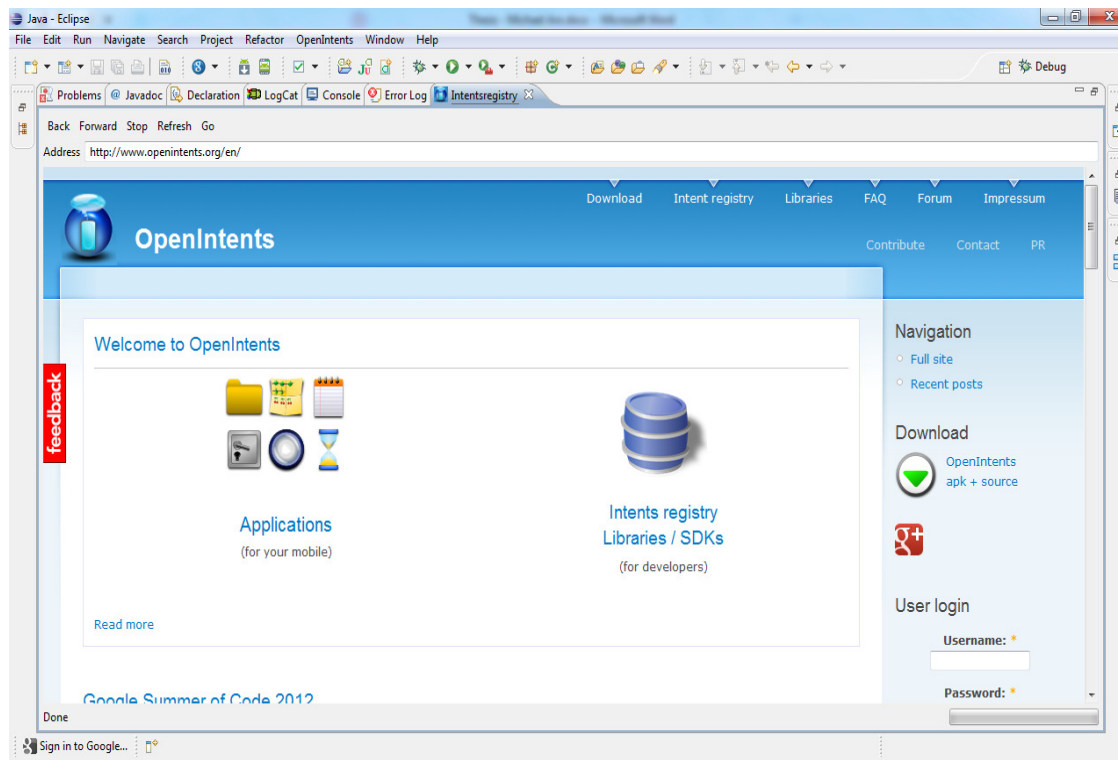


Figure 25. Intents registry browser view

7 SUMMARY

The previous chapters gave an overview of the module. The structure and functionalities were discussed. In the entire chapter, the Intents Registry Eclipse plugin project was analysed. As a proof of concept, a tool for browsing the registry within Eclipse has been created.

7.1 Publishing plugin to the Eclipse marketplace

Eclipse marketplace provides an easy to use means of finding, downloading and installing Eclipse based plugins directly within Eclipse using the marketplace client that comes pre-installed with the latest versions of Eclipse IDE. The Intents Registry Eclipse plugin contains a description of the plugin, current version number, the author, license used, supported platforms and IDEs to mention a few. By publishing the Intents Registry Eclipse plugin to the Eclipse marketplace, it will gain exposure to many users in different parts of the world.

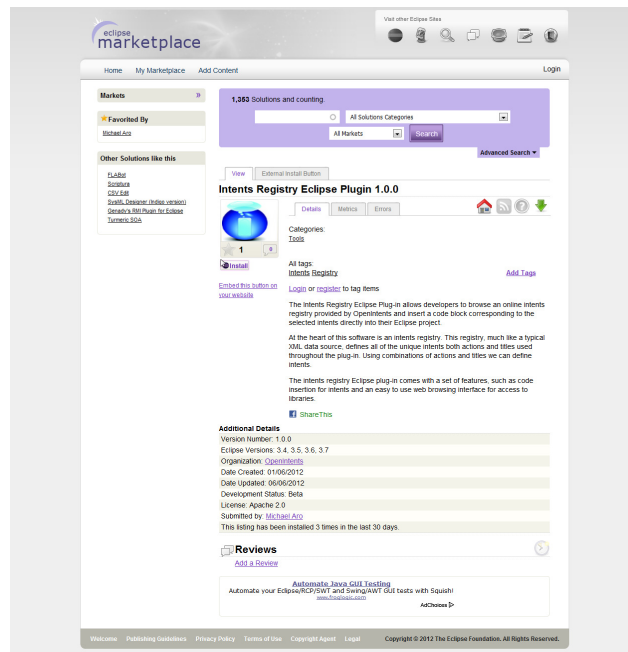


Figure 26. Plugin on the Eclipse marketplace

7.2 Becoming a contributor / committer

I have being consistent as a developer with OpenIntents working on projects and already I have a committer access. The benefit of this is that I can quickly check in code to our repository without waiting for approval patches to be approved. An individual cannot gain a committer status except nominated by other committers.

7.2.1 Contributing to OpenIntents

There are several ways to contribute to OpenIntents. One can help with documentation, translation of software to different languages, bug reporting, feature request, bug fixing and feature development. OpenIntents allows me to publish the project using the Apache 2.0 open source license. This allows other developers to contribute to the plug-in project. My aim is that the plug-in helps the developers to be more productive and further development of the plugin is ensured. The mailing list serves as a communication channel with other developers.

7.2.2 Project mailing list

The OpenIntents mailing list is open to anyone. Subscription to the mailing list is needed to get mails in the box. There is also a Google group site for discussions, notices and updates about the different OpenIntents projects. Interested developers can discuss about bugs and new features for example, directly with other developers.

8 CONCLUSION

In this thesis, we describe the implementation of an intents registry Eclipse plugin. This tool allows a developer to browse the intents registry made available online on the OpenIntent's web server. The requirement was to make this registry available to the developer within the Eclipse IDE. The goal was to create a copy of the online registry within Eclipse and provide an easy access for programmer to add intents to their code in their editors. In addition, the functionality of browsing online artifacts within Eclipse was added.

So far, the plug-in provides three views with the following contributions made by the thesis:

- Intents selection page
- Browsing view
- Help view
- An Intents Registry Eclipse plugin was built.

It can be changed to another language. Intents can be added from the registry. These are stable prototypes with functionalities that need to be improved but good enough to be put to use. With the intents selection page, the user can add insert code to the editor. With the browser view, a user can navigate the OpenIntents' online resources without leaving Eclipse. The help provides guides on the use of the software.

There were obvious challenges in the beginning trying to understand the Eclipse codebase which is very large as well as how to extend it with a plugin. But it is very rewarding to improve my problem solving, java and Eclipse development skills. More importantly, learning how to become a good open source developer.

8.1 Future Work

In the future release of the system, more features will be added including context assist using the actions of intents as proposals. This will enhance programmer's productivity. New user stories will be collected from the people that use the tool to determine the useful parts, parts to be improved and the new ones to be added.

8.1.1 Feature

Content Assist – This will be made available via the code completion feature of Eclipse.

9 REFERENCES

- /1/ Eclipse Platform [online].
Available in URL form - <http://www.eclipse.org/platform/> <Checked 08.29.2012>
- /2/ Eclipse Platform API Specification [online].
Available in URL form:
<http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Findex.html> <Checked 08.29.2012>
- /3/ SAXParser [online].
Available in URL form:
<http://docs.oracle.com/javase/6/docs/api/javax/xml/parsers/SAXParser.html>
<Checked 08.29.2012>
- /4/ Grindstaff, Chris, "How to Use the JFace Tree Viewer," Applied Reasoning, May 5, 2002 (www.eclipse.org/articles/treeviewer-cg/TreeViewerArticle.htm)
- /5/ Gauthier, Laurent, "Building and Delivering a Table Editor with SWT/JFace," Mirasol Op'nWorks, July 3, 2003
(www.eclipse.org/articles/Article-Tableviewer/table_viewer.html).
- /6/ Springgay, Dave, "Creating an Eclipse View," OTI, November 2, 2001(www.eclipse.org/articles/viewArticle/ViewArticle2.html).
- /7/ Liotta, Matt, "Extending Eclipse with Helpful Views," July 20, 2004(www.devx.com/opensource/Article/21562).
- /8/ Deva, Prashant, "Folding in Eclipse Text Editors," March 11, 2005(www.eclipse.org/articles/Article-Folding-in-Eclipse-Text-Editors/folding.html).
- /9/ Ho, Elwin, "Creating a Text-Based Editor for Eclipse," HP, June 2003(devresource.hp.com/drc/technical_white_papers/eclipeditor/index.jsp).
- /10/ Klinger, Doina, "Creating JFace Wizards," IBM UK, December 16, 2002(www.eclipse.org/articles/Article-JFace%20Wizards/wizardArticle.html).

- /11/ Fatima, Azra, “Wizards in Eclipse: An Introduction to Working with Platform and Custom Wizards,” HP, August 2003
(devresource.hp.com/drc/technical_articles/wizards/index.jsp).
- /12/ Adams, Greg, and Dorian Birsan, “Help Part 1, Contributing a Little Help,”
- /13/ August 9, 2002 www.eclipse.org/articles/Article-Online%20Help%20for%202_0/help1.htm
- /14/ Ford, Neal, “Centralizing Help in Eclipse,” ThoughtWorks, June 21, 2005(www-128.ibm.com/developerworks/opensource/library/os-eclipsehelp)
- /15/ Zink, Lori, “Understanding Eclipse Online Help,” HP, February 2005(devresource.hp.com/drc/resources/eclipsedoc/index.jsp)
- /16/ Java internationalization tutorial
(java.sun.com/docs/books/tutorial/i18n/intro/index.html).
- /17/ Internationalization (I18n) (java.sun.com/j2se/1.4.2/docs/guide/intl/).
- /18/ Kehn, Dan, Scott Fairbrother, and Cam-Thu Le, “How to Internationalize Your Eclipse Plug-in,” IBM, August 23, 2002
(eclipse.org/articles/Article-Internationalization/how2I18n.html).
- /19/ Kehn, Dan, “How to Test Your Internationalized Eclipse Plug-in,” IBM,
- /20/ August 23, 2002 (eclipse.org/articles/Article-TVT/how2TestI18n.html).
- /21/ Kehn, Dan, Scott Fairbrother and Cam-Thu Le, “Internationalizing Your Eclipse Plug-in,” IBM, June 1, 2002 www-128.ibm.com/developerworks/opensource/library/os-i18n
- /22/ ISO 639 language codes www.unicode.org/onlinedat/languages.html
<Checked 08.29.2012>
- /23/ ISO 3166 country codes (www.unicode.org/onlinedat/countries.html).
- /24/ Eclipse Help: Java Development User Guide > Tasks > Externalizing Strings

- /25/ "Dynamic Content in Eclipse Help System Pages," Eclipse.org, December 2005(www.eclipse.org/eclipse/platform-ua/proposals/xhtml/HelpDynamicContent.html)
- /26/ Eclipse Help: Platform Plug-in Developer Guide > Programmer's Guide >Plugging in help
- /27/ Adams, Greg, "Creating Product Branding," OTI, November 27, 2001(www.eclipse.org/articles/product-guide/guide.html).
- /28/ Glozic, Dejan, and Dorian Birsan, "How to Keep Up to Date," IBM, August27, 2003 (www.eclipse.org/articles/Article-Update/keeping-up-to-date.html).
- /29/ McCarthy, Pat, "Put Eclipse Features to Work for You," IBM, October 14, 2003 (www-128.ibm.com/developerworks/opensource/library/os-ecfeat/).
- /30/ Markus Barchfeld, "Build and Test Automation for plug-ins and features" <http://www.eclipse.org/articles/Article-PDE-Automation/automation.html> <Checked 08.29.2012>
- /31/ Ant Web site ant.apache.org/ <Checked 08.29.2012>
- /32/ Eclipse Ant FAQ (eclipsewiki.editme.com/ANTFaq). <Checked 08.29.2012>
- /33/ Eclipse Help: PDE Guide > Exporting a plug-in
- /34/ Android Intents <http://developer.android.com/reference/android/content/Intent.html> <Checked 08.29.2012>
- /35/ <http://www.eclipse.org/org/#about> <Checked 08.29.2012>
- /36/ Java FAQ - http://java.com/en/download/faq/whatis_java.xml <Checked 08.29.2012>
- /37/ XML <http://www.w3.org/XML/> <Checked 08.29.2012>

- /38/ Project URL: https://github.com/openintents/intents-registry_eclipse <Checked 08.29.2012>
- /39/ Code repository : https://github.com/openintents/intents-registry_eclipse <Checked 08.29.2012>
- /40/ OI - <http://code.google.com/p/openintents/wiki/OpenIntents> <Checked 08.29.2012>
- /41/ Intents Registry site: <http://www.openintents.org/en/xml/registry> <Checked 08.29.2012>
- /42/ Junit - http://junit.sourceforge.net/doc/faq/faq.htm#overview_1 <Checked 08.29.2012>
- /43/ SWTBot - <http://www.eclipse.org/swtbot/> <Checked 08.29.2012>
- /44/ OI About - <http://www.openintents.org/en/node/202> <Checked 08.29.2012>
- /45/ Intent Filters - <http://developer.android.com/guide/components/intents-filters.html> <Checked 08.29.2012>

APPENDIX 1

INSTALLATION

System Requirements

The Intents Registry Eclipse plugin is compatible with the following Eclipse versions:

- 3.6.x (Helios)
- 3.7.x (Indigo)
- 4.2.x (Juno)

Update Site Location

The update site is located at http://openintents.github.com/intents-registry_eclipse/org.openintents.intentsregistry.update/

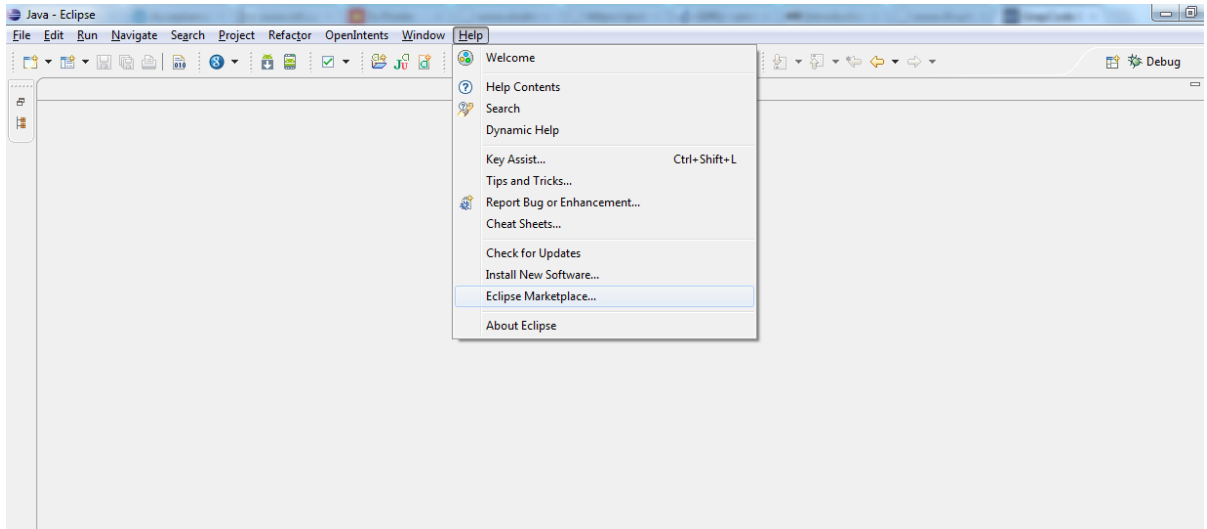
Installation

There are several ways to install the plug-in, like any other plug-in, by applying the following steps.

Using the Eclipse marketplace client

1. Install or update using the Eclipse Marketplace (recommended)

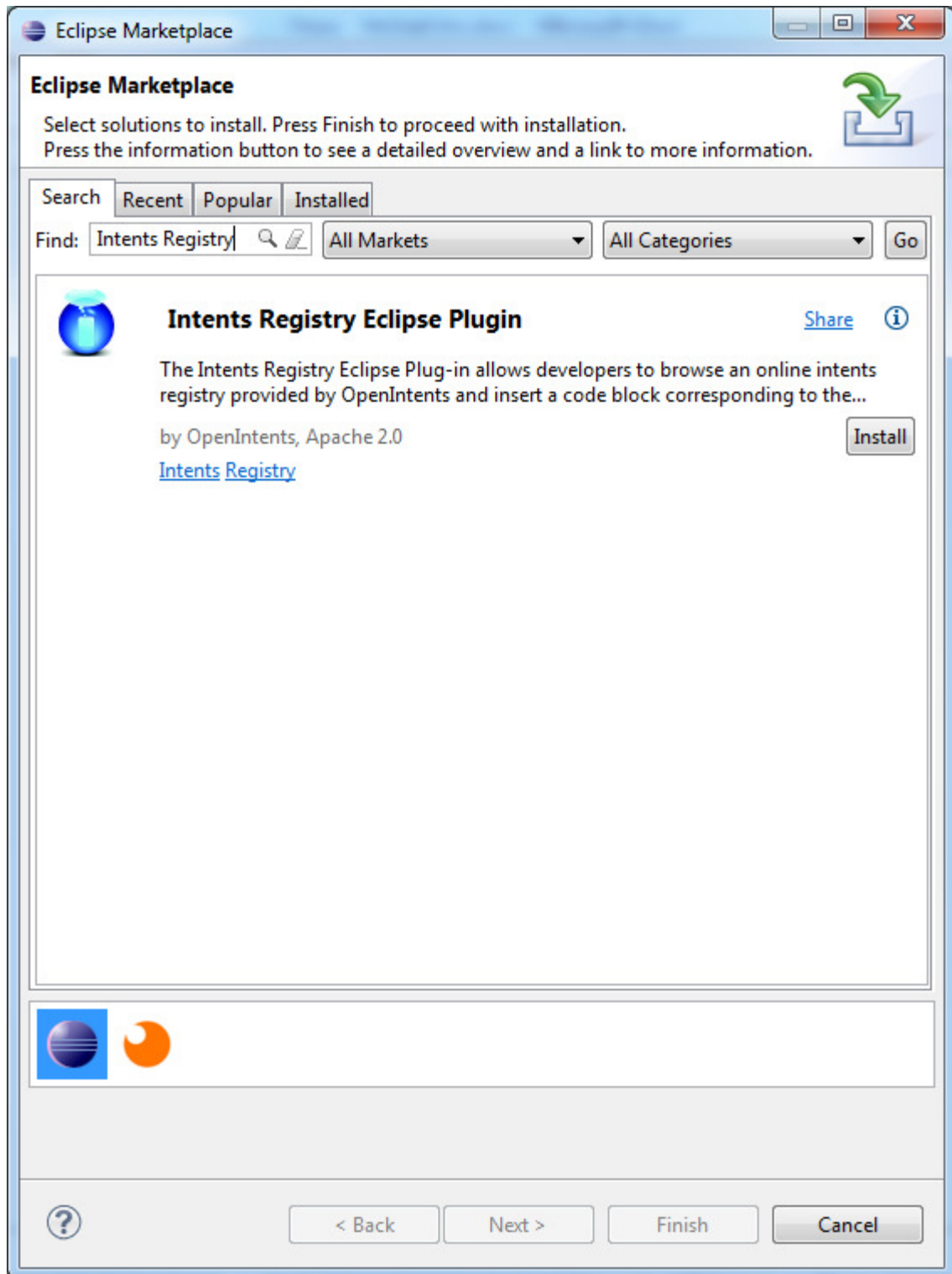
Help -> Eclipse Marketplace



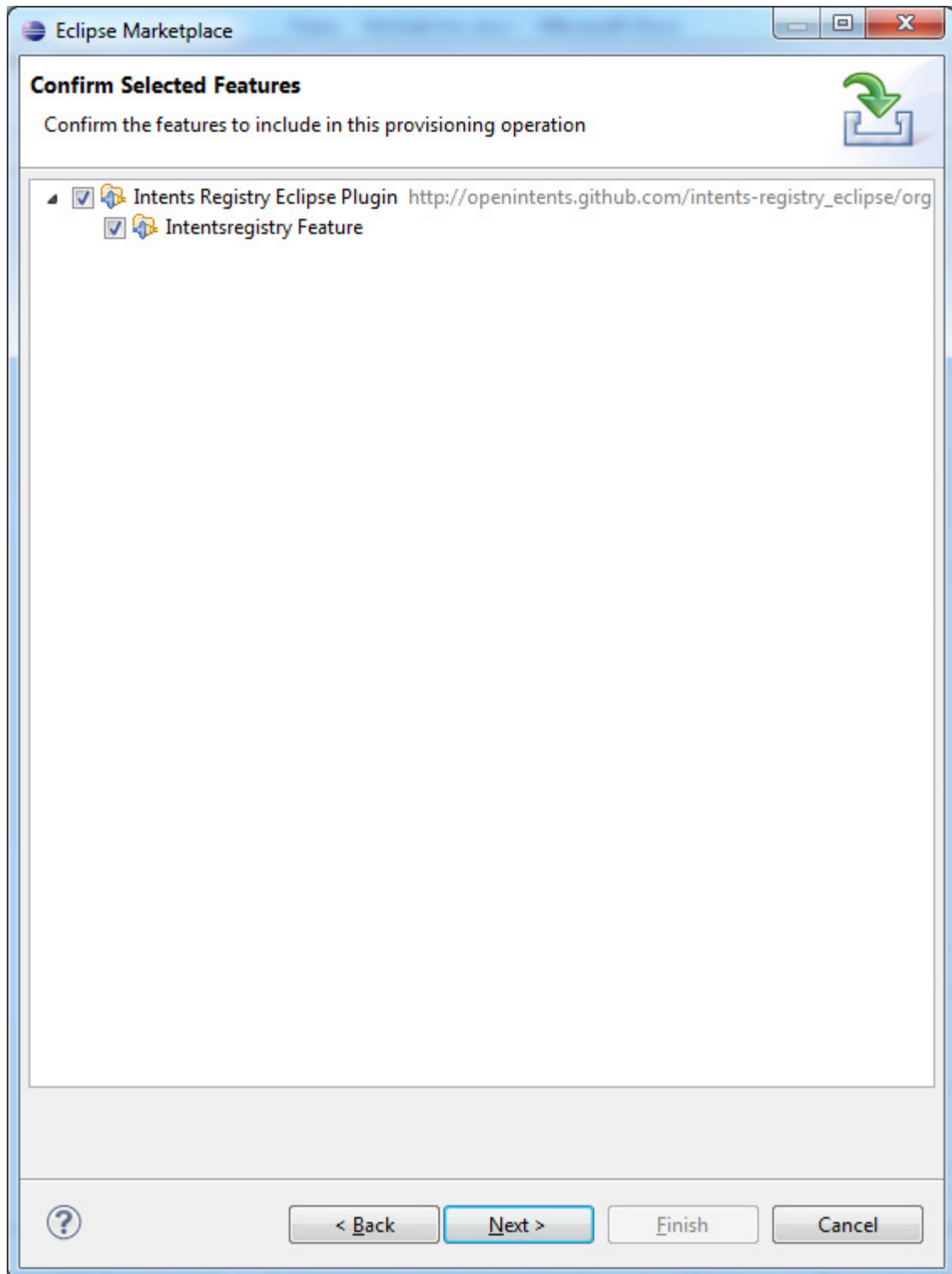
2. Find by typing “Intents Registry” in the search box and click the search button

Intents Registry Eclipse Plugin (by OpenIntents, Apache 2.0)

On the “Eclipse Marketplace”, click on the “Install” button.



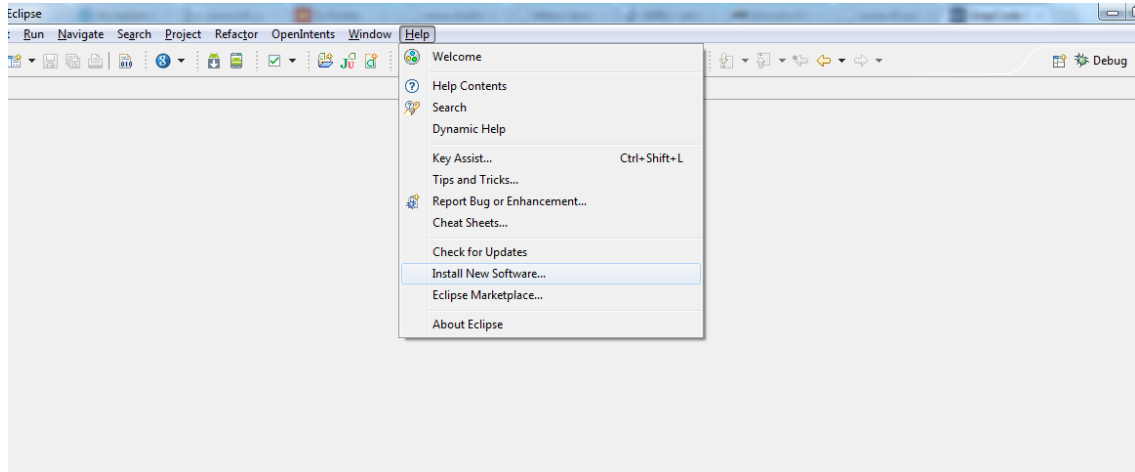
3. Confirm selected features and click on the “Next” button



4. Accept the license and click on the “Finish” button.
5. Accept the security warnings and restart Eclipse when prompted.

Installing via the update site

1. Install new software

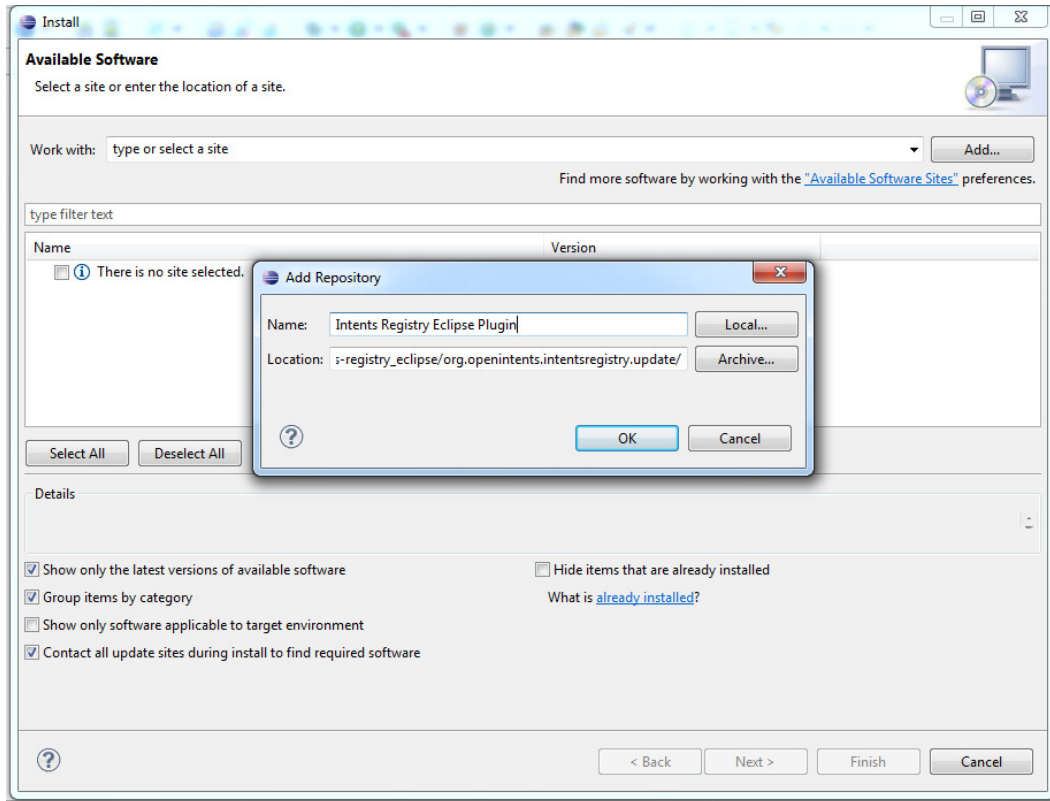


2. Press the “Add...” button on the ‘Available Software’ page

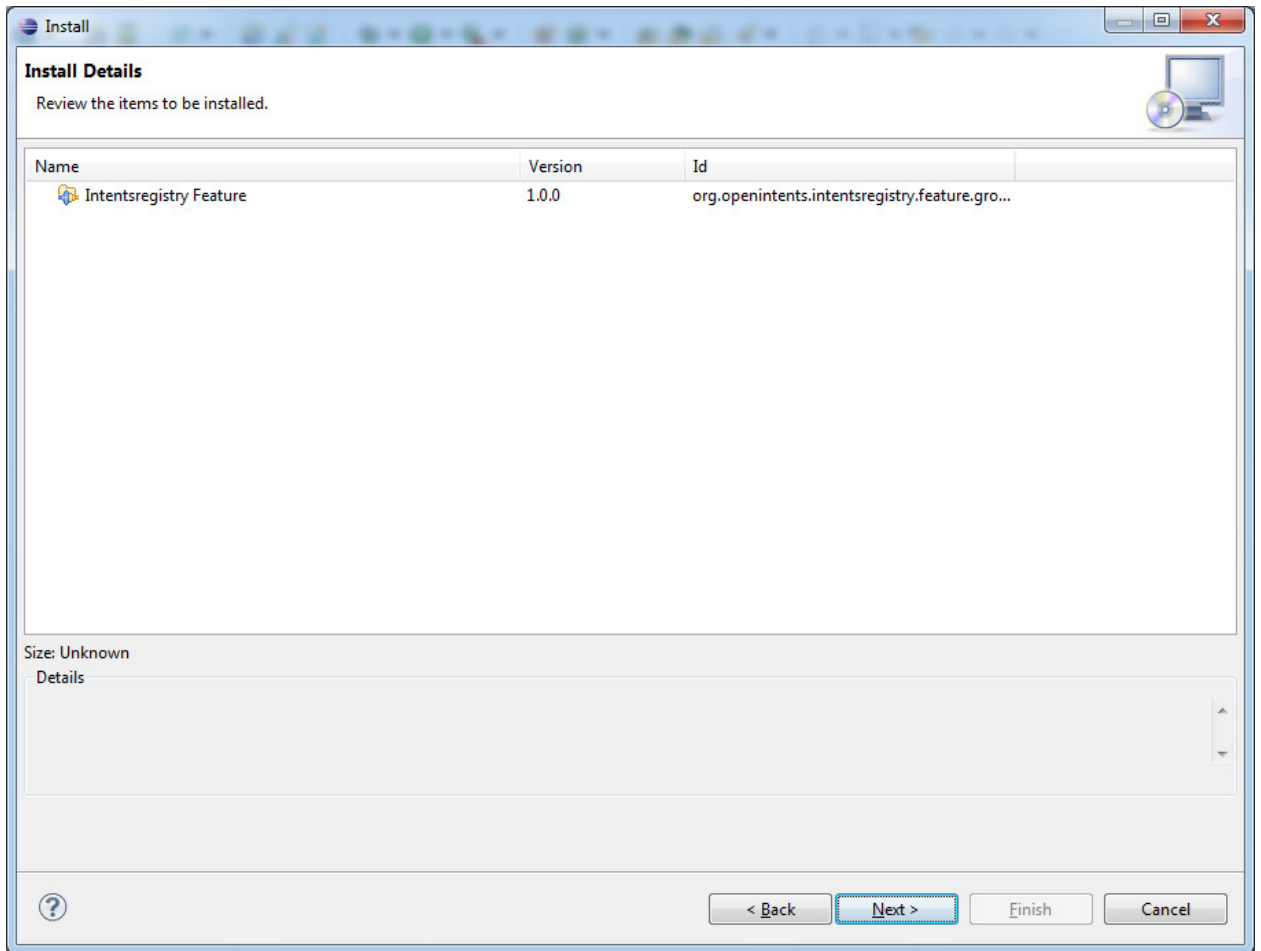
In the “Add Repository” dialog, enter the following information and press the “OK” button

Name: Intents Registry Eclipse Plugin

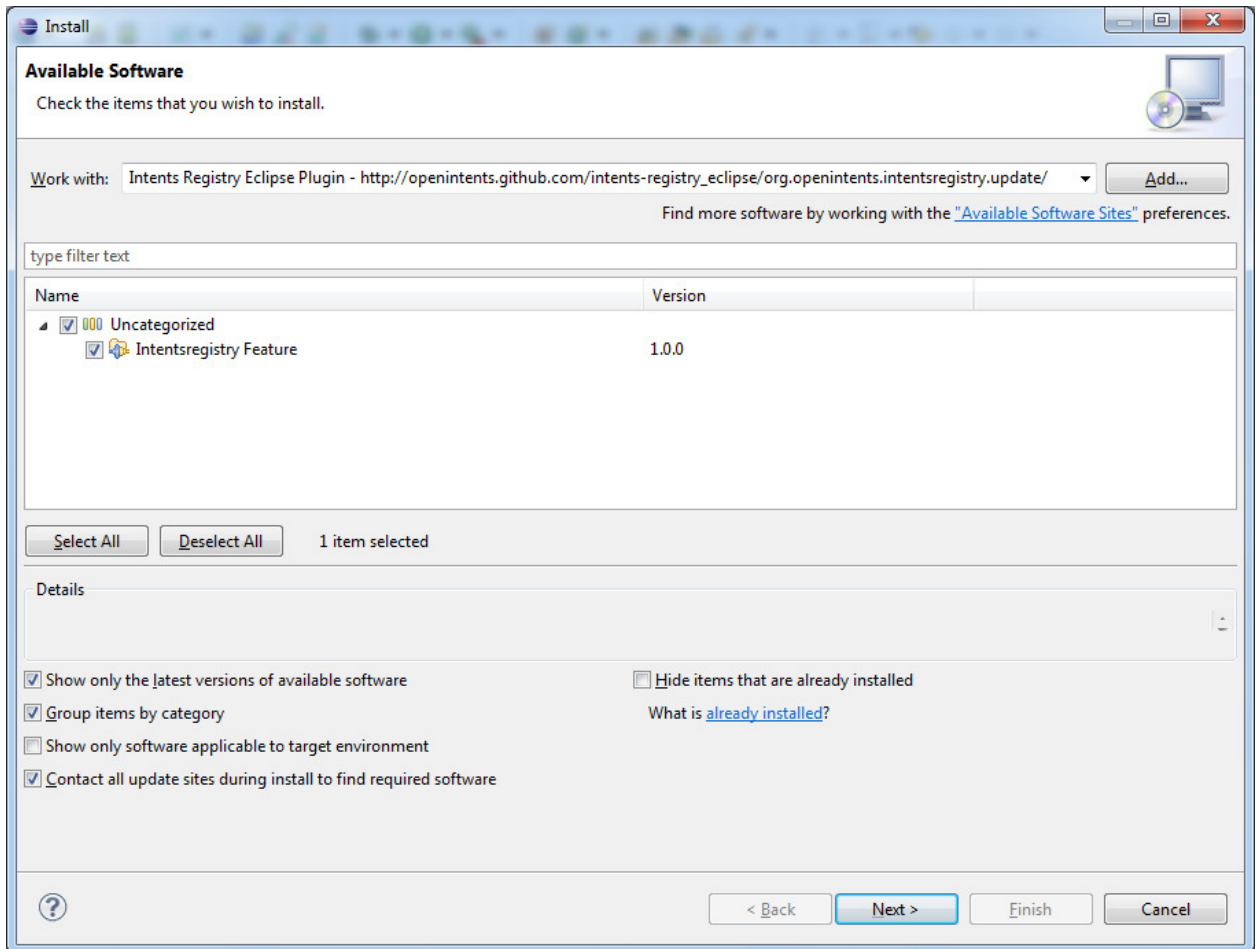
URL: http://openintents.github.com/intents-registry_eclipse/org.openintents.intentsregistry.update/



3. Click the Next button on the “Install Details” page



5. Accept the License Agreement and click on the “Finish” button.
6. Accept the security warnings and restart Eclipse when prompted



Usage

The installation will add “OpenIntents” to the menu bar of the Eclipse workbench window.

Once the plugin has been installed, you can use it to insert intent actions into your code by selecting the appropriate title from the dialog box. You can also use it to browse the source code of libraries that you use in your software development. Here is an example to illustrate this.

Assume that you are working on a project and you want to add an action to Intent for displaying data to the user, you will select “View data” on the dialog box which will add the corresponding action to your code.

