Chair of
Software Engineering

Origo IDE integration

Rafael Bischof

02-911-758
Master Thesis
March 2007 - September 2007

Chair of Software Engineering
Department of Computer Science
ETH Zürich

Till Bay
Prof. Bertrand Meyer

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Abstract

The goal of this master thesis is to design and implement extensions for several Integrated Development Environments that allow a developer to interact with the Origo development platform directly.

Acknowledgments

I want to thank my supervisor Till Bay. Further thanks go to Peter Wyss, my friend and co-worker on Origo. A special thank to Patrick Ruckstuhl, who did not only write almost the whole Origo system but also helped me with dozens of problems with EiffelStudio and Eiffel in general. His advices and knowledge saved me many days of work. And I thank Beat Strasser who helped me to solve several Eclipse and Java related problems.

Contents

Chapter 1

Introduction

1.1 Motivation

The distributed software development platform Origo allows to interact with the platform by using an XML-RPC API. Using this protocol and the extension mechanisms of various integrated development environments (IDEs) it is possible to integrate features of Origo into the IDEs. This eases and speeds up the development process because a developer does not have to use Origo's website for everything and can perform some tasks and access information directly in his IDE.

The goal of this work is to write extensions for various IDEs. We decided in favor of EiffelStudio (cf. [Eif]) because the Origo platform is mostly written in the Eiffel programming language and EiffelStudio is the most popular IDE for Eiffel. Origo supports projects independent of the programming languages they use, so we also implemented integrations for Visual Studio (cf. [Visa]), which is a very popular IDE for C++ and .NET on Windows, and for Eclipse (cf. [Ecla]), which is an open source, platform independent IDE used for Java and many other programming languages.

1.2 Overview

In Chapter 2 we describe the general design of the integrations by using a command line tool and give a short overview over the extension mechanisms of the three chosen IDEs. Chapter 3 explains everything about the command

line tool and can also be used as a manual. What the IDE integrations can do and how you do it is described in Chapter 4. In Chapters 5 to 7 we describe for each IDE how the integration is installed and give a short overview about the source code structure and what can be found in which class. Finally Chapter 8 is a tutorial that shows how the basic features of Origo can be used.

Chapter 2

Design

2.1 General Design

Because we have to implement same functianlity for three different IDEs of which each one uses another programming language for extensions we decided to implement a separate command line tool to perform the actual API calls. This way we do not have to search for and use XML-RPC and FTP libraries for each of these programming languages, in particularly this would have been a problem in EiffelStudio (see Section 2.2.1). Instead we have to start a process (the command line tool) and parse it's standard output. Methods to do this are part of the base framework of every used programming language.

2.2 Extension Mechanisms

Each of the three chosen IDEs has another extension mechanism although they all provide the functionality we needed for our integrations. In this section we will shortly describe these mechanisms.

2.2.1 EiffelStudio

The extension mechanism of EiffelStudio is clearly the hardest to use of the three IDEs: it does not have any extension support at all. But because EiffelStudio is an open source application there was still a way to extend it with

Origo functionality. Unfortunately this approach made things more complicated because we had to deal with a large amount of almost undocumented source code. At this point we once again want to thank Patrick Ruckstuhl who knew the code structure and whose help saved us days of work.

Because it's our goal that the Origo integration will be part of the normal EiffelStudio distribution some time, we were confronted with some restrictions. So we wanted to avoid to use any library that was not already used in the EiffelStudio source code. Because we needed the Goanna library (cf. [Goa]) for XML-RPC in Eiffel and it is not used in EiffelStudio, this was one of the major reasons to use an external command line tool to perform the Origo API calls.

Another restriction was, that EiffelStudio is a single threaded application. This made the initial idea of a self refreshing workitem list impossible because we could not start a separated thread which waits for a certain time span until it gets the data from Origo. Because it can take several seconds until a workitem list is fully retrieved it would have been unusable even if we found a way to call a feature periodically.

2.2.2 Visual Studio

Visual Studio provides three levels of extensiblilty (cf. [Visb], [CGE]). The first level are macros. These are very limited as they can't create new windows and many other things. Also they are not compiled but interpreted which is a performance drawback and you can't protect your code if you distribute them (although this is not a problem here because our IDE integrations are open source).

The second level are add-ins. They are easy to develop as Visual Studio 2005 provides good wizards to create add-in projects. Add-ins provide you full control over the Visual Studio automation object model (cf. [Visc]) and can be written with any programming language that supports COM.

The third and final level are packages. They can do almost anything with Visual Studio like providing new project types, support of new programming languages and integrating new debug tools. There is also the Managed Package Framework (MPF) which enables the developer to write managed code using C# or VB.NET. Another big advantage of MPF is that it provides default implementations to use many COM interfaces. But you can mix up the usage of MPF and the automation model.

We decided to use packages as they are the most powerful way to provide extensions and we did not really knew what exactly we will need. It is also the best choice if we look to the future if the integration should be extended we are not limited to certain capabilities. I do not know if the current integration would have been possible with an add-in, but i do not think so.

2.2.3 Eclipse

Eclipse has the best extension mechanism of the three IDEs as it is built as a framework and a set of services for building applications from plug-in components (cf. [Eclb]). Eclipse comes with several standard plug-ins including the Java Development Tools (JDT) and is mostly used for as a Java IDE. Although it is written in Java there are plug-ins that include support for other programming languages such as C/C++. As Eclipse is not an IDE but a plug-in framework the plug-ins are not limited to software development related applications.

One of Eclipse's standard plug-ins is the Plug-in Development Environment (PDE) which is tailored to the needs of an Eclipse plug-ins developer. It also contains templates that hook in to the various extension points Eclipse provides and which build it's extension mechanism. These templates can be selected from a menu and are added to the current plug-in project as default implementations of the corresponding extension point. The PDE also provides a very simple-to-use way of creating a deployment package.

Chapter 3

Command Line Tool

3.1 Purpose

The command line tool is an application that is used by the IDE integrations to perform the communication with Origo (see Section 2.1). This communication is not limited to the XML-RPC API provided by Origo but also contains some methods to handle the user's Origo FTP account.

There were several reasons to create a separate application that communicates with Origo. This way we only had to use one library for XML-RPC and FTP in one language instead of three different libraries in three different programming languages. We also did not have to add a new library to the EiffelStudio source code which would hinder our ambitions to make our integration an official part of EiffelStudio (see Section 2.2.1). Furthermore other programmers can use this command line tool for their own applications to get rid of the need of finding and using an XML-RPC library.

3.2 Usage

The arguments passed to the command line tool have following format:

```
<method_name> <arguments>
```

where `<method_name>` is the name of the XML-RPC API call or the FTP method and `<arguments>` are the necessary arguments. Each argument is

preceded by a `-` as argument sign and the arguments can be given in an arbitrary order. The command line tool performs the given call and writes the return values to the standard output stream. If anything goes wrong an error is printed to the standard error stream. If the command line tool is not used correctly a usage help is written to the standard error stream.

3.3 Methods

This section describes the various methods the command line tool provides. The title of each subsection is the `<method_name>` mentioned in Section 3.2.

3.3.1 login

description Performs a login into Origo.

arguments `uk [ak]`

- `uk` The userkey consists of 32 capital letters which replaces username and password and can be requested on the Origo website[1].
- `ak` An optional application key. An application key consists of 32 capital letters and can be also be requested on the Origo website. If this parameter is omitted the application key of the command line tool is used.

output A session ID which consists of 32 capital letters. It is valid for 1 hour and the timer is reset every time you make an API call using this session ID.

3.3.2 project_list_of_user

description Returns all projects of which a user is a developer or an owner.

arguments `s u`

- `s` A session ID that consists of 32 capital letters and can be received with the login method.

[1] http://origo.ethz.ch

`u` The nickname of the user whose project list you want to see.

output Each output of a line represents a project and has following format: `<project_id> <project_name> <user_group>` where `<user_group>` 3 denotes that the user is a member of this project and `<user_group>` 4 means the user is an owner of this project.

3.3.3 my_name

description Returns the username of the user associated with the given session.

arguments `s`

`s` A session ID that consists of 32 capital letters and can be received with the login method.

output Your username.

3.3.4 my_password

description Returns the password of the user associated with the given session.

arguments `s`

`s` A session ID that consists of 32 capital letters and can be received with the login method.

output Your password.

3.3.5 release

description Creates a new release for a project. The files you want to release must be in the main directory of your Origo FTP account. You can use the command line tool to upload your files using the `ftp_upload` method (see Section 3.3.8). The files that build the release are deleted form your Origo FTP account during this call.

arguments `s pid n [d] ver fl`

- `s` A session ID that consists of 32 capital letters and can be received with the login method.
- `pid` The ID of the project for which you want to build a release. This argument is a positive integer.
- `n` The release's name. You do not have to include the name of the project. Think of it as a kind of subtitle.
- `d` A description of the release. Most probably you will have to put it in `""`. do not pass an empty description because this will crash the command line tool due to a bug in the argument parser library in the Eiffel framework.
- `ver` The program version that is associated with this release. This argument is a string so you can write something like `1.4.7c`
- `fl` The list of file which will build this release. The list must be formatted this way: `<file>;<file>;...;<file>` where `<file>` equals `<filename>:<platform>`. `<filename>` is the name of a file in the main directory of your Origo FTP account and `<platform>` is the name of the platform the given file is compiled for.

output This method does not return anything if it is successful.

3.3.6 workitem_list

description Returns the last `num` workitems for which you are subscribed. You can change your workitem subscription on the Origo website.

arguments `s num [uro]`

- `s` A session ID that consists of 32 capital letters and can be received with the login method.

`num`	The maximum number of workitems the list contains. This argument must be a positive integer.
`uro`	Denotes that only the unread workitems should be returned. This argument does not have a value. If you do not add this argument you will receive all workitems.

output A list for workitems. One workitem consists of several lines and the workitems are separated by an empty line. The detail information for each workitem is not included in the workitem list. The exact format and content of workitems is described in Section 3.4.

3.3.7 workitem

description Returns all information about a workitem.

arguments `s w`

`s`	A session ID that consists of 32 capital letters and can be received with the login method.
`w`	The ID of the workitem of which you want details.

output The detailed information about one workitem. The exact format and content of workitems is described in Section 3.4.

3.3.8 ftp_upload

description Opens a FTP connection to upload.origo.ethz.ch and uploads the file to your Origo FTP account.

arguments `u p f`

`u`	Your Origo username.
`p`	Your Origo password.
`f`	Path of the file you want to upload.

output This method does not return anything if it is successful.

3.3.9 ftp_file_list

description	Returns a list of all files in the main directory of your Origo FTP account.
arguments	`u p`
`u`	Your Origo username.
`p`	Your Origo password.
output	A list of files. Each line contains one filename.

3.3.10 ftp_delete

description	Deletes a file in the main directory of your Origo FTP account.
arguments	`u p f`
`u`	Your Origo username.
`p`	Your Origo password.
`f`	Name of the file you want to delete.
output	This method does not return anything if it is successful.

3.4 Workitem Format

This section describes the format and content of the workitems returned by the command line tool. Some data is common to all workitems and is described in Section 3.4.1. Additionally there is specialized data for each workitem type and each workitem type can contain additional detailed information (see Sections 3.3.6 and 3.3.7). The type of the workitem and thus the way how you can see which information is passed is part of the general workitem data and thus also described in Section 3.4.1.

3.4.1 General Workitem Data

The data that is common to each workitem has following format

```
<type>
<workitem_id>
<creation_time>
```

```
<project_id>
<project_name>
<username>
```

and is followed by the specialized data of the corresponding workitem type.

`<type>`	An integer that specifies the workitem type and thus the data following to the general workitem data. Type `1` is an issue workitem (3.4.2), type `2` a release workitem (3.4.3), type `3` a commit workitem (3.4.4), type `4` a wiki workitem (3.4.5) and type `5` a blog workitem (3.4.6).
`<workitem_id>`	The ID of the workitem. This as a positive integer.
`<creation_time>`	The time when the workitem was created. It is a unix timestamp[2] and thus a positive integer.
`<project_id>`	ID of the project this workitem belongs to.
`<project_name>`	Name of the project this workitem belongs to.
`<username>`	Origo nickname of the user who performed the action that created this workitem.

3.4.2 Issue Workitems

Issue workitems are not implemented yet and thus do not hold any additional information.

3.4.3 Release Workitems

Format of specialized release workitem data:

```
<name>
<version>
```

[2] http://www.unixtimestamp.com

```
<description>
<file_count>
<files>
```

where

`<name>`	Release name, does not contain the project name. Look at it as a kind of subtitle.
`<version>`	Version of the program.
`<description>`	Description of this release. The format of `<description>` looks like that: `<length>:<text>` where `<length>` is a non-negative integer that is the number of characters of `<text>`. This number ignores all carriage returns (`\r` in C notation), so especially Windows users should be careful as Windows adds a carriage return in front of every line feed (`\n` in C notation) in the standard output.
`<file_count>`	This part of the detailed information. `<file_count>` shows how many files a release has.
`<files>`	This is part of the detailed information. `<files>` consists of `<file_count>` lines of which each looks like this: `<filename>:<platform>` where `<filename>` is the files name and `<platform>` the name of the platform the file belongs to.

3.4.4 Commit Workitems

Format of specialized commit workitem data:

```
<revision>
<log>
<diff>
```

where

`<revision>`	Revision of the commit. `<revision>` is a positive integer.
`<log>`	Commit log. This is formatted the same way as `<description>` in Section 3.4.3.
`<diff>`	This is part of the detailed information. `<diff>` contains the SVN commit diff and is formatted the same way as `<description>` in Section 3.4.3.

3.4.5 Wiki Workitems

Format of specialized wiki workitem data:

```
<title>
<URL>
<diffURL>
<revision>
<old_revision>
<diff>
```

where

`<title>`	Title of the wiki page.
`<URL>`	URL of the wiki page.
`<diffURL>`	URL of the changes to the last revision. If it is a new wiki page, this line is empty.
`<revision>`	This is part of the detailed information. It is the wiki page revision and thus a positive integer.
`<old_revision>`	This is part of the detailed information. The revision number of the wiki page before the change. If it's a new wiki page `<old_revision>` is 0. Together it's a non-negative integer.

`<diff>`	This is part of the detailed information. `<diff>` is the difference between the two revisions of the wiki page. If it's a new wiki page it's the difference between the new page and an empty page. It is formatted the same way as `<description>` in Section 3.4.3.

3.4.6 Blog Workitems

Format of specialized blog workitem data:

```
<title>
<URL>
<diffURL>
<revision>
<old_revision>
<diff>
```

where

`<title>`	Title of the blog entry.
`<URL>`	URL of the blog entry.
`<diffURL>`	URL of the changes to the last revision. If it is a new blog entry, this line is empty.
`<revision>`	This is part of the detailed information. It is the blog entry revision and thus a positive integer.
`<old_revision>`	This is part of the detailed information. The revision number of the blog entry before the change. If it's a new blog entry `<old_revision>` is 0. Together it's a non-negative integer.
`<diff>`	This is part of the detailed information. `<diff>` is the difference between the two revisions of the blog entry. If it's a new wiki page it's the difference between the new entry and an empty entry. It is formatted the same way as `<description>` in Section 3.4.3.

3.5 Code Design

3.5.1 Overview

The command line tool is written in the Eiffel programming language. It uses the Goanna library (cf. [Goa]) to perform the XML-RPC calls and Eposix (cf. [dB]) to handle the FTP methods. Additionally the *arg_parser* of the EiffelStudio framework is used.

The program consists of only three classes which we will shortly describe: *ROOT_CLASS* (Section 3.5.2), *USAGE_PRINTER* (Section 3.5.3) and *ORIGO_CLIENT_CONSTANTS*. The last one only contains several constants.

3.5.2 ROOT_CLASS

The *ROOT_CLASS* class performs everything except printing the usage of the command line tool if it's not used correctly (see Section 3.5.3 for that). The creation feature *make* has an exception handler that catches all exceptions and writes it's content to the standard error stream. The first important feature is *select_call* which reads the first passed argument and calls the corresponding feature to handle the method and the remaining arguments.

The features mentioned above are named *call_<method_name>* and handle the complete API call. First they parse the remaining arguments, perform the API call and finally write the results to the standard output. If anything goes wrong they write an error message to the standard error stream.

3.5.3 ORIGO_USAGE_PRINTER

The *ORIGO_USAGE_PRINTER* class contains several features to write the usage of the command line tool to the standard error stream. The feature *usage* prints the whole usage and is called when an unrecognized method name or no method name at all is passed as an argument to the command line tool.

For each method the program provides, there are two different features: *usage_<method_name>* and *usage_<method_name>_arguments*. The first one writes the method's name, the arguments it expects and a short description

what it does. The second feature lists all arguments together with a description of each argument. Both these features are called if a correct method name is passed to the command line tool but the arguments do not fit.

There are also are several helper features such as *usage_calls* that prints the usage of all calls or the *usage_argument_<argument_name>* features that write a description for the corresponding argument to the standard error stream.

Chapter 4

IDE Integration

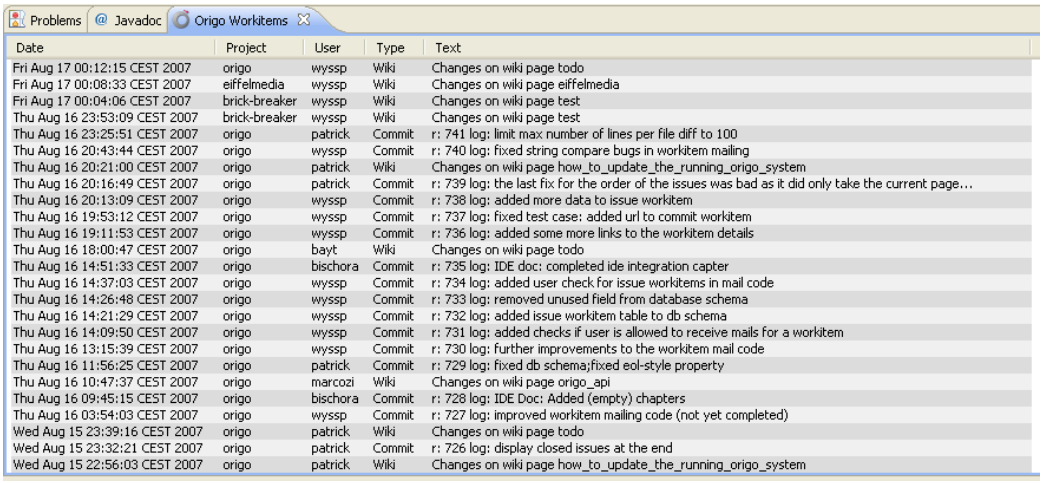
4.1 Overview

The integration of all three IDEs provide the the same functionality in the same way in principle. This chapter describes the general features the integrations provide and shows how to use them. If there are differences to the standard integration described here we will go into detail in the Chapter of the corresponding IDE.

4.2 Preferences

The integrations contain some parameters that must or can be configured. They can be found in the preferences of their respective IDE. Before you can use any features of the integration you have to set two of the parameters. The first one is the complete path of the command line tool (see Chapter 3). Without this the integration can't find it and thus it's impossible to perform the API calls. Because all Origo API calls require a login it is also necessary that you set your Origo user key, which is a substitute for your username and password. You can get one at http://origo.ethz.ch/origo_home/settings/userkey.

The remaining three preferences, number of workitems, refresh interval and unread only, are used for the workitem display and will be explained further in Section 4.3).



Date	Project	User	Type	Text
Fri Aug 17 00:12:15 CEST 2007	origo	wyssp	Wiki	Changes on wiki page todo
Fri Aug 17 00:08:33 CEST 2007	eiffelmedia	wyssp	Wiki	Changes on wiki page eiffelmedia
Fri Aug 17 00:04:06 CEST 2007	brick-breaker	wyssp	Wiki	Changes on wiki page test
Thu Aug 16 23:53:09 CEST 2007	brick-breaker	wyssp	Wiki	Changes on wiki page test
Thu Aug 16 23:25:51 CEST 2007	origo	patrick	Commit	r: 741 log: limit max number of lines per file diff to 100
Thu Aug 16 20:43:44 CEST 2007	origo	wyssp	Commit	r: 740 log: fixed string compare bugs in workitem mailing
Thu Aug 16 20:21:00 CEST 2007	origo	patrick	Wiki	Changes on wiki page how_to_update_the_running_origo_system
Thu Aug 16 20:16:49 CEST 2007	origo	patrick	Commit	r: 739 log: the last fix for the order of the issues was bad as it did only take the current page...
Thu Aug 16 20:13:09 CEST 2007	origo	wyssp	Commit	r: 738 log: added more data to issue workitem
Thu Aug 16 19:53:12 CEST 2007	origo	wyssp	Commit	r: 737 log: fixed test case: added url to commit workitem
Thu Aug 16 19:11:53 CEST 2007	origo	wyssp	Commit	r: 736 log: added some more links to the workitem details
Thu Aug 16 18:00:47 CEST 2007	origo	bayt	Wiki	Changes on wiki page todo
Thu Aug 16 14:51:33 CEST 2007	origo	bischora	Commit	r: 735 log: IDE doc: completed ide integration capter
Thu Aug 16 14:37:03 CEST 2007	origo	wyssp	Commit	r: 734 log: added user check for issue workitems in mail code
Thu Aug 16 14:26:48 CEST 2007	origo	wyssp	Commit	r: 733 log: removed unused field from database schema
Thu Aug 16 14:21:29 CEST 2007	origo	wyssp	Commit	r: 732 log: added issue workitem table to db schema
Thu Aug 16 14:09:50 CEST 2007	origo	wyssp	Commit	r: 731 log: added checks if user is allowed to receive mails for a workitem
Thu Aug 16 13:15:39 CEST 2007	origo	wyssp	Commit	r: 730 log: further improvements to the workitem mail code
Thu Aug 16 11:56:25 CEST 2007	origo	patrick	Commit	r: 729 log: fixed db schema;fixed eol-style property
Thu Aug 16 10:47:37 CEST 2007	origo	marcozi	Wiki	Changes on wiki page origo_api
Thu Aug 16 09:45:15 CEST 2007	origo	bischora	Commit	r: 728 log: IDE Doc: Added (empty) chapters
Thu Aug 16 03:54:03 CEST 2007	origo	wyssp	Commit	r: 727 log: improved workitem mailing code (not yet completed)
Wed Aug 15 23:39:16 CEST 2007	origo	patrick	Wiki	Changes on wiki page todo
Wed Aug 15 23:32:21 CEST 2007	origo	patrick	Commit	r: 726 log: display closed issues at the end
Wed Aug 15 22:56:03 CEST 2007	origo	patrick	Wiki	Changes on wiki page how_to_update_the_running_origo_system

Figure 4.1: Eclipse workitem display

4.3 Workitem Display

The workitem display is a tool window (or a view in Eclipse, but that's the same thing with a different name) which can be opened by the IDE's menu and contains a list of workitems. The list shows the date when the workitem was created, the project it belongs to, the user that created it, it's type and a short description (see Figure 4.1). You can doubleclick on a an list entry and a dialog opens that displays more detailed information about the workitem (see Figure 4.2). If it's a wiki- or a blogworkitem there are links that you can click on to open your standard browser and show the corresponding website.

The workitem displays a certain number of workitems. First, you can only see the workitems you have submitted on the Origo website. Furthermore you may only see the workitems that are not marked as read. This option can be set in the IDE preferences. There you can also set the maximum number of workitems you want to see in the workitems display. The list is periodically refreshed in an interval that can also be set in the IDE's preferences.

4.4 Release Dialog

The release dialog (see Figure 4.3) can be opened using the IDE's menu. Before the dialog is usable it receives some necessary information from Origo, so this can take a few seconds. On the top you have a combo box where you can select the project for which you want to build a release. The content of

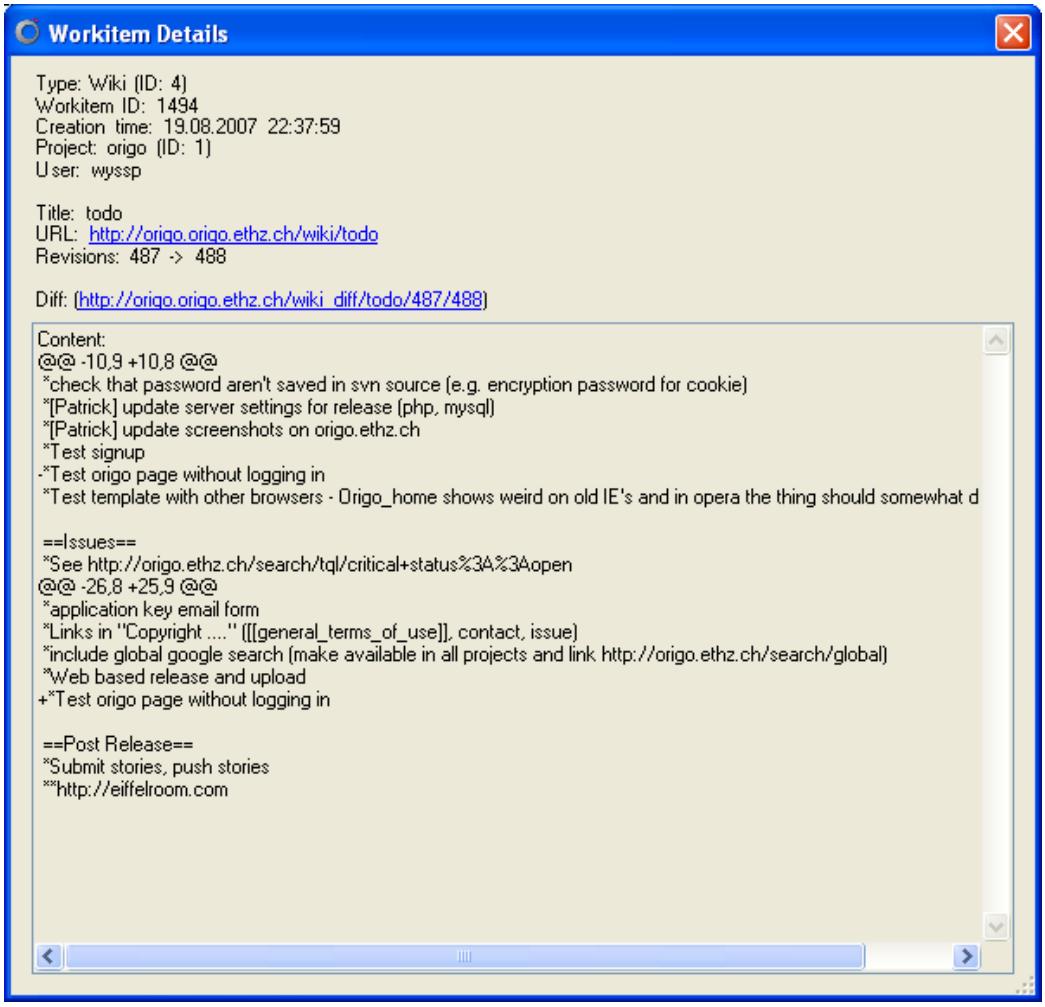
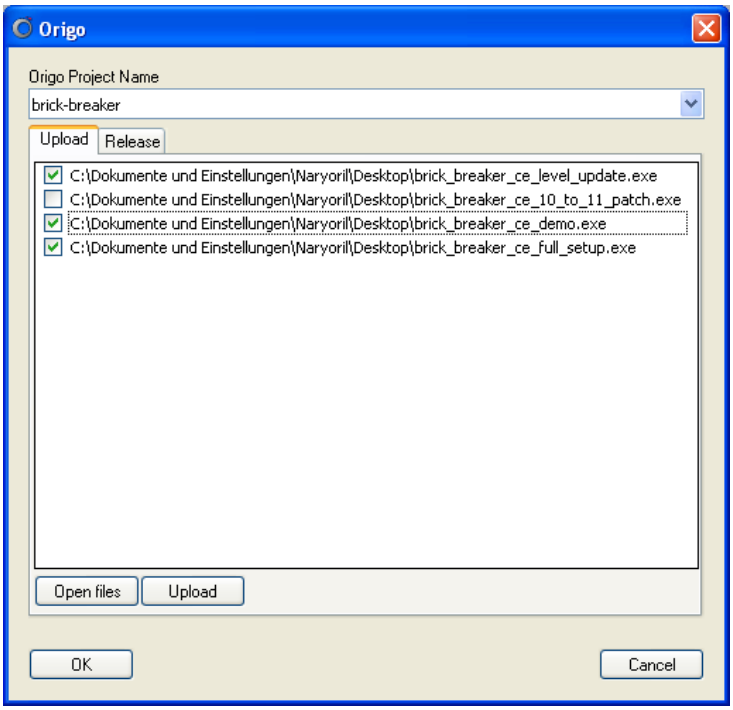
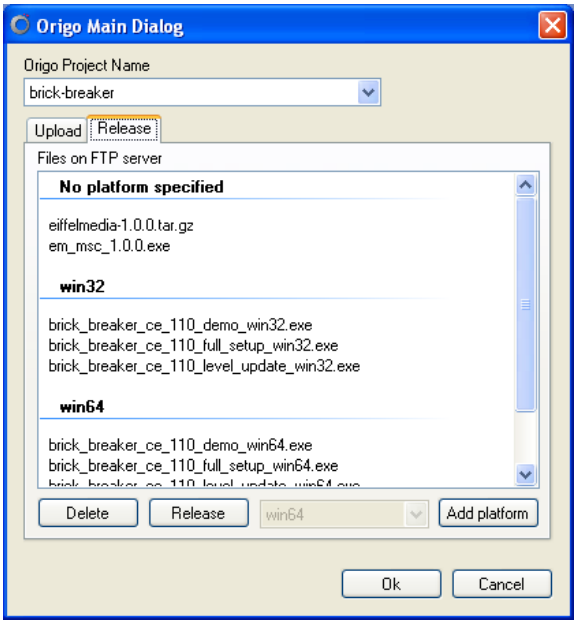


Figure 4.2: Visual Studio wiki workitem detail dialog



(a) EiffelStudio upload tab



(b) Visual Studio 2005 release tab

Figure 4.3: Release dialog

this project list is received from Origo and contains all projects of which you are owner and hence are allowed to publish releases for. It also contains two tabs which we will describe now.

4.4.1 Upload Tab

In the upload tab of the release dialog (see Figure 4.3(a)) you can upload files to your Origo FTP account instead of using a fully-fledged FTP client. With the open files button you can select the files you want to upload. If you decide that you do not want to upload some of them you can simply uncheck the checkbox in front of the filename. The upload button uploads all checked files in the file list.

4.4.2 Release Tab

The release tab (see Figure 4.3(b)) is a bit more complicated. The list contains the files on your Origo FTP account. If you select some of them and press the delete button you delete them from the FTP server, so be careful with it.

With the add platform button you can add a platform to your release. Each file that should be released must be assigned to a platform. To do this, you have to select the files in the list and then select the platform you want to assign them to. If you have assigned a platform to all files that should build the release you can click the release button and are asked to enter some additional information about your release. Before the release is really published you are asked for a confirmation. The files that made up the release will be deleted from your Origo FTP account and moved to the Origo download area.

Chapter 5

EiffelStudio 6 Integration

5.1 Differences to the Standard Integration

As mentioned in Section 2.2.1 EiffelStudio does not have an extension mechanism and is not multi threaded. This leads to mainly one big difference to the "standard integration" described in Chapter 4 which causes to many small changes: It is not possible to periodically refresh the workitem display list. This means of course, that you do not have a preference that specifies the refresh interval. And you have to force a refresh by yourself by pressing the refresh button in the bottom right corner of the workitem display tool window.

Another difference is, that Vision2, the Eiffel widget library, does not have a link enabled label. Even if it had one, there is no easy way to open the system's standard browser, according to the Eiffel developer mailing list. So the feature that you can click on a link in the workitem detail dialog of wiki- and blogworkitems is missing in the EiffelStudio integration. As a replacement for that the two URLs are part of the textbox that contains the diff. This way you can select the URLs, copy and past them into your browser.

5.2 Installation Instructions

To install EiffelStudio with Origo integration download the command line tool from *http://origo.ethz.ch/download* if you do not already have it some-

where on your hard disk. Copy it into any directory you like. Also download the EiffelStudio with Origo integration and copy the downloaded archive into the directory where you installed EiffelStudio and unzip it there. The path structure inside the archive should be identical with the one in your EiffelStudio directory.

Remember to set the preferences (*Tools–>Preferences–>Tools–>Origo*) as described in Section 4.2 after you started EiffelStudio for the first time upon installing the Origo version.

To open the release dialog select *Project–>Origo Release Dialog* respectively *View–>Tools–>Origo workitems* to open the workitem display tool window.

5.3 Code Documentation

5.3.1 Menu Commands

The release dialog can be opened by selecting *Project–>Origo Release Dialog*. The class that handles this is *EB_ORIGO_COMMAND* and is used in feature *build_project_menu* of *EB_DEVELOPMENT_WINDOW_MENU_BUILDER*. where the real menu entry is added. In the same class in the feature *build_tools* the feature *build_workitem_tool* is called that creates and adds the Origo workitem display tool window to the *View–>Tools* menu.

5.3.2 Preferences

EB_ORIGO_DATA holds the Origo preferences. If you want to add a new setting simply add it there similarly to the ones already existing, but be careful that you do not forget anything. It is added to the *EB_GUI_PREFERENCES* class so the Origo preferences can be found in *Tools–>Preferences–>Tools –>Origo*.

5.3.3 API Calls

All API and FTP methods, namely everything concerning the command line tool, are handled in the class *EB_ORIGO_API_CALLS*. The standard output and standard error of the command line tool that is called with the correct

arguments are appended to respective strings. After each call *last_error* is assigned to the output of the standard error stream, so if it is empty no error occurred. Several helper features help to parse the standard output to construct a correct output. Because workitems are pretty complex there are several helper features of which each parses the string that was attached to the standard output stream and creates a corresponding workitem object out of it.

5.3.4 Tool Window

Class *EB_ORIGO_WORKITEM_TOOL* implements the workitem display tool window by inheriting from *EB_TOOL* so there is not much that has to be done to make it a working tool window. The class is pretty straight forward. If the refresh button is pressed *refresh_workitem_list* is called that performs the API calls to login and receive the workitem list from Origo and then calls *fill_workitem_grid* which loops through the workitem list and fills an *EV_GRID*. A double click on a grid row calls *display_workitem_details* which opens an *EB_ORIGO_WORKITEM_DETAILS_DIALOG*.

show_information_label hides the *EV_GRID* that contains the workitem list and shows a label to display some information instead. On the other hand *hide_information_label* hides the label and shows the workitem list.

5.3.5 Release Dialog

The release dialog is implemented in the class *EB_ORIGO_DIALOG* which contains an *EV_NOTEBOOK* which itself contains two tabs represented by *EB_ORIGO_UPLOAD_TAB* and *EB_ORIGO_RELEASE_TAB*. The classes itself are not very complex. One important thing to not is that in the FTP file list on the release tab the currently selected platform of a file is stored as a string in the list item's *data*. If *data* is *Void* then the list item is part of a group header.

Chapter 6

Visual Studio 2005 Integration

6.1 Installation Instructions

To install the Visual Studio package download the command line tool from *http://origo.ethz.ch/download* if you do not already have it somewhere on your hard disk. Copy it into any directory you like. Also download the Visual Studio package installer. Close all open instances of Visual Studio, start the installer and select the directory where you want to install it to. If you do not already have installed the ProjectAggergator2 or the Visual Studio SDK leave the checkbox at the end of the installation and the ProjectAggregator2 will also be installed. Remember to set the options (*Tools–>Options–>Origo*) as described in Section 4.2 after you started Visual Studio for the first time upon installing the package.

You will have two new menu commands in your *Tools* menu, one opens the release dialog and the other the workitem display tool window.

6.2 Code Documentation

6.2.1 Package

The main class of the Visual Studio package is *OrigoVSIntegration* found in *VsPkg.cs*. It contains several attributes that configure things like new preference entries or tool windows. Some additional information like the image that is shown on the splash screen are also defined in this class by overriding meth-

ods inherited from *Microsoft.VisualStudio.Shell.Interop.IVsInstalledProduct*.

6.2.2 Menu Commands

The menu commands are added in *OrigoVSIntegration.Initialize()* which is inherited from *Microsoft.VisualStudio.ShellPackage*. Both of the menu commands simply call a method that open the release dialog or the tool window respectively. Some additional information like the picture that is shown is configured in *CtcComponents/OrigoVSIntegration.ctc*. The image itself is defined in *VSPackage.resx*.

6.2.3 Options

The options pages are registered by the *ProvideOptionPage* attribute on the *OrigoVSIntegration* class. They are represented by the two classes *OrigoSettings* and *OrigoWorkitemSettings* which are very simple and self explaining and inherit from *Microsoft.VisualStudio.Shell.DialogPage*.

6.2.4 API Calls

The class *XmlRpcCalls* handles all calls to the Origo API or Origo FTP server by using the command line tool (see Chapter 3). It is almost the same as the Eiffel implementation for EiffelStudio which is described in Section 5.3.3. But instead of setting a member variable of the class for the last error an *ApplicationException* is thrown if an error occurs.

Although it is possible to directly read the standard output and standard error streams from the command line tool process we had to read them into a string and then parse the resulting strings. The reason for that is, that we have to wait until the process has finished until we try to parse the stream. But as soon as there are 4096 characters (at least on Windows XP SP2) in the stream's buffer the process can not write more into the stream and waits until some characters are read. Because we have to wait until the process has finished this lead to a deadlock.

6.2.5 Tool Window

The workitem display tool window is implemented in class *OrigoWorkitemToolWindow*. Its base class is *Microsoft.VisualStudio.Shell.ToolWindowPane*. It almost does not contain anything but defines that its content is a control of type *OrigoWorkitemControl* which is the real implementation of the tool window's functionality and inherits from *System.Windows.Forms.UserControl*.

An object of the class *System.Threading.Timer* waits for a given interval and then gets the workitem list from Origo by calling *GetWorkitems*. But because this feature runs in another thread it is not allowed to edit the *System.Windows.Forms.ListView* that represents the workitem list. For this reason a delegate is invoked which calls *RefreshWorkitems* which causes that the method is run in the thread where the control and hence the *ListView* was created.

6.2.6 Release Dialog

The release dialog is implemented in class *OrigoMainDialog*. There is not much to explain as this class is pretty simple although it is relatively large. Because *System.Windows.Forms.ListView*s support grouping by themselves we did not have to take special care about that. One thing to mention may be that the necessary data is received from Origo in the *Shown* event, because the dialog is at least partially visible at that time.

Chapter 7

Eclipse 3.3 Integration

7.1 Installation Instructions

To install the Eclipse plug-in download the command line tool from the origo download page (*http://origo.ethz.ch/download*) if you do not already have it somewhere on your hard disk. Copy it into any directory you like. Also download the Eclipse plug-in and unzip it into the *plugins* folder in the folder you have installed Eclipse. Remember to set the preferences found in *Window–>Preferences–>Origo* as described in Section 4.2 after you started Eclipse for the first time after installing the plug-in.

To open the workitem display view select *Window–>Show View–>Other*. In the dialog that opens up select *Origo–>Origo workitems*. To open the release dialog select *Window–>Origo Releases*.

7.2 Code Documentation

7.2.1 Plug-in

If you open *plugin.xml* with the Plug-in Development Environment (PDE) you see a very nice overview about the plug-in. You do several things very easily, like changing the version number, adding new extensions or building a deliverable. This file is the heart of the plug-in.

7.2.2 Menu Commands

The menu command is implemented as an action set. Its menu path is set in the *Extensions* tab of the PDE view of *plugin.xml* at the extension point *org.eclipse.ui.actionSets*. This also allows to have the command not only as a menu but also as icon in the toolbar. If the command is launched the method *run* of class *DialogLauncher* is called.

Adding an action set for the workitem display view is not necessary as an appropriate menu command automatically added by using the extension point *org.eclipse.ui.views*. That's also the place where you can set up some things about this menu entry.

7.2.3 Preferences

The preference pages use the *org.eclipse.ui.preferencePages* extension point and can be set up there using *plugin.xml*. The real implementation is found in the package *ch.ethz.origo.ide.eclipse.preferences* and contains three files. The class *PrefrenceInitializer* simply sets the default values and *OrigoPreferencePage* which is the implementation of the preference page is also straight forward if you take a look at the source code.

7.2.4 API Calls

Class *OrigoApiCalls* handles all API calls and FTP methods by using the command line tool (see Chapter 3). The implementation is almost identical to the Visual Studio implementation described in Section 6.2.4. We also faced the same problem with the standard output stream buffer limitation which is also mentioned there.

7.2.5 View

Views are registered at the *org.eclipse.ui.views* extension point and our view is implemented class *WorkitemView*. The periodical refresh is implemented as inner class *RefreshTimer*. Because of the same problem as in Visual Studio described in Section 6.2.5 the timer has to call classes derived from *java.lang.Runnable*, namely the inner classes *TableFiller* and *ErrorShower*. The workitem list is implemented using a *org.eclipse.jface.viewer.TableViewer*

and the inner class *TableMouseListener* handles double clicks on a workitem and shows the workitem detail dialog. This dialog is implemented in class *OrigoWorkitemDetailDialog*. The rest of the class is self explaining.

7.2.6 Release Dialog

The release dialog is implemented in class *OrigoReleaseDialog*. The class is not very complicated although it is pretty large. The file lists are represented by *org.eclipse.swt.widgets.Table*s. Because they do not provide some functionality that we expected we had to write some helper functions like *moveTableItem* or *indexOfTableItem*. The platform of a file in the release file list can be retrieved by using *getData* on the table item. If it returns *null* it is a header separator item.

Chapter 8

Origo Tutorial

8.1 Introduction

This chapter will show you step by step how you can use some features of Origo. We will request a new project, set some basic project settings, use the workitem list and the issue tracker and create a release of the project.

8.2 Creating a User Account

Before we can do anything else except downloading or reading some stuff about a project we have to create a user account. To do so visit the Origo website on http://origo.ethz.ch and you will see a *Register* in the top right corner. Fill in all the information you are asked for (see Figure 8.1) and click on *Create new account*. Your are now signed in with your new user account.

8.3 Creating a Project

The next thing we have to do is to create a project. Click on *Home - <username>* in the menu at the left. In the case of this tutorial it's called *Home - Naryoril*, we will refer to it as *Origo home* in the rest of the tutorial. There isn't much to see there for now, but that will change later on. For now, click on *Create Project* which is displayed in the menu now.

Now you see the project request form (see Figure 8.2). Think carefully

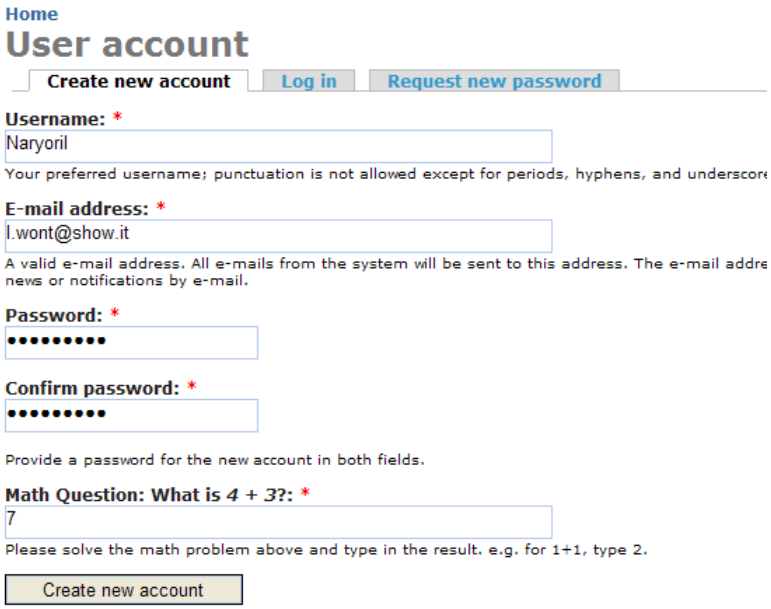


Figure 8.1: Origo user account creation page

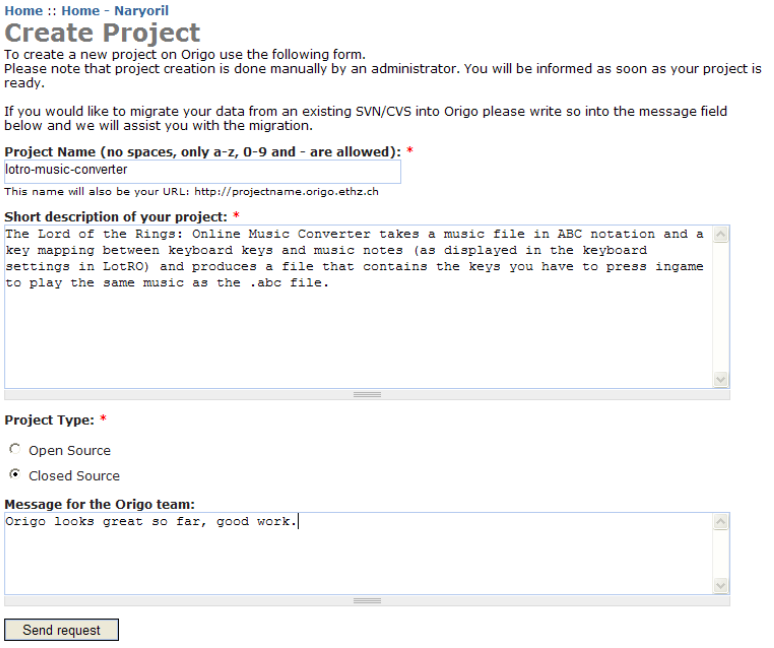


Figure 8.2: Origo project request form

about your project name, as it will also be your project's URL. You can also choose that your project should be closed source, this means that nobody who is not added as a project member can access the project's SVN repository or receive commit workitems. If everything is filled in, click on *Send request*.

Please note that the projects are activated manually, so it can take a few hours. We hope that we can activate a project within 24 hours, although we can't guarantee it. So you will have to wait now until you receive an e-mail that tells you that your project was created.

8.4 Basic Project Settings

8.4.1 Project Home

So now we received an e-mail that our project was created and can be found hat *http://<projectname>.origo.ethz.ch* which is *http://lotro-music-converter.origo.ethz.ch* in our tutorial. When we visit that page, called *project home* from now on, we will see a text about minestrone. This is just a standard text that is added if a new project is created and that can be changed easily. Just click on *Edit* right above the text and write whatever you want in the large text field.

8.4.2 Logo

After writing a new text for our project home we want our own logo at the top left corner. To change click on *Project Settings* in the menu and select the tab *Logo*. Browse for the image you want to use as a logo and click on *Save*. Then wait a moment and you will see your own logo instead of the Origo logo. Please be careful that you don't take a logo that's too large or it will look really ugly. Take a size that isn't bigger than 200x110 pixels and you won't have any problems. You can also test a bit with other sizes, but don't expect too much.

8.4.3 Forums

Next we want to give our users the possibility to communicate with each other or with us. Origo provides a simple forum for that, but we must create

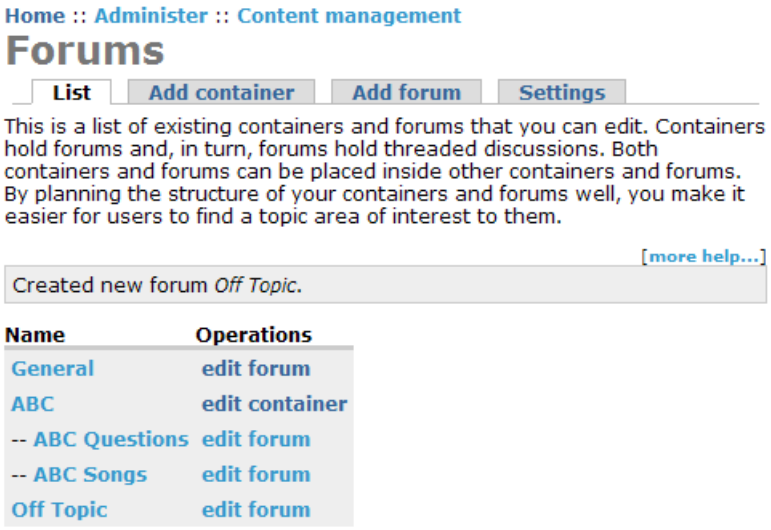


Figure 8.3: Admin view of the forums

them first. Select *Forums* in the menu (at the left). Let's say we want to add 4 forums: General, ABC Questions, ABC Songs and Off Topic.

Select the tab *Add forum* fill in "General" and a description, e.g. "All about the lotro-music-converter", and click on *Submit*. We will be back on the forum where the new forum was added. Because the next two Forums we want to create are similar, we pack them together into a container, so select the tab *Add container*. A container is created the same way as a forum and we name it "ABC". Back in the list we see that the ABC container is above the General forum, but we don't want that. So click on *edit container*, select a weight of 1 and click once again on *Submit*. Once again in the list we see the forums are ordered now as we wanted to.

The next step is to add the ABC Questions forum inside the ABC container. To do so to the same as when we added the General forum but select ABC in the *Parent* dropdown list. Repeat that with the ABC Songs forum. Finally add the Off Topic forum by leaving the *Parent* on <root> and set the weight to 2 such that it will be displayed at the bottom.

Now the list should look like in Figure 8.3. You can now click on *Forum* at the top (above the search) and you will see that the forum now contains the ones we added. You can see this view and the rest of he page including the logo in Figure 8.4.

Figure 8.4: User view of the forums with the rest of the page

8.4.4 Adding other Developers

We don't want want to do everything by ourselves and we have friends that want to help us with our project. Adding other developers is pretty simple, just click on *Project Settings* in the menu and you will see the member list. Now you have to know your friend's Origo username, type it into the textfield and click on *Add User*. Now your friend can create and edit wiki pages and blogs, read private wiki pages, he can commit on the SVN, or even read it if it's a closed source project and he has several additional rights in the issue tracker. All other special acitivity like creating releases is reserved for project owners. Please note that it's possible to have more than one project owner.

8.4.5 Subversion (SVN)

You don't really have to set up an SVN, but Origo provides you one. You can find it at *https://svn.origo.ethz.ch/<projectname>/* . If you want to know how to use SVN take a look at their homepage[1].

8.5 Using Workitems

Now we come to a special feature of Origo: the Workitems. You can get a short overview about the workitems in the workitems on the corresponding wiki page[2], so we won't go into that but describe how you can use them.

We only receive the workitems we have subscribed. To change the subscriptions go to Origo home[3] and click on *Settings* in the menu. There you will see a list that contains all projects you have bookmarked (we will describe that in just a little later) or of which you are a member. So you will most probably see only one project now. For each project you can select how you want to be notified about which workitem type. There are only two notification types now: *Show on Origo Home* (and the IDE plug-ins) and by *Mail Notification*. In our case we want to be notified about all types on Origo Home and about SVN commits and new Issues by mail notification. Simply check the corresponding checkboxes and click on *Save subscriptions*.

[1] http://subversion.tigris.org/

[2] http://origo.ethz.ch/wiki/Workitems

[3] http://origo.ethz.ch/origo_home or http://<project>.origo.ethz.ch/origo_home

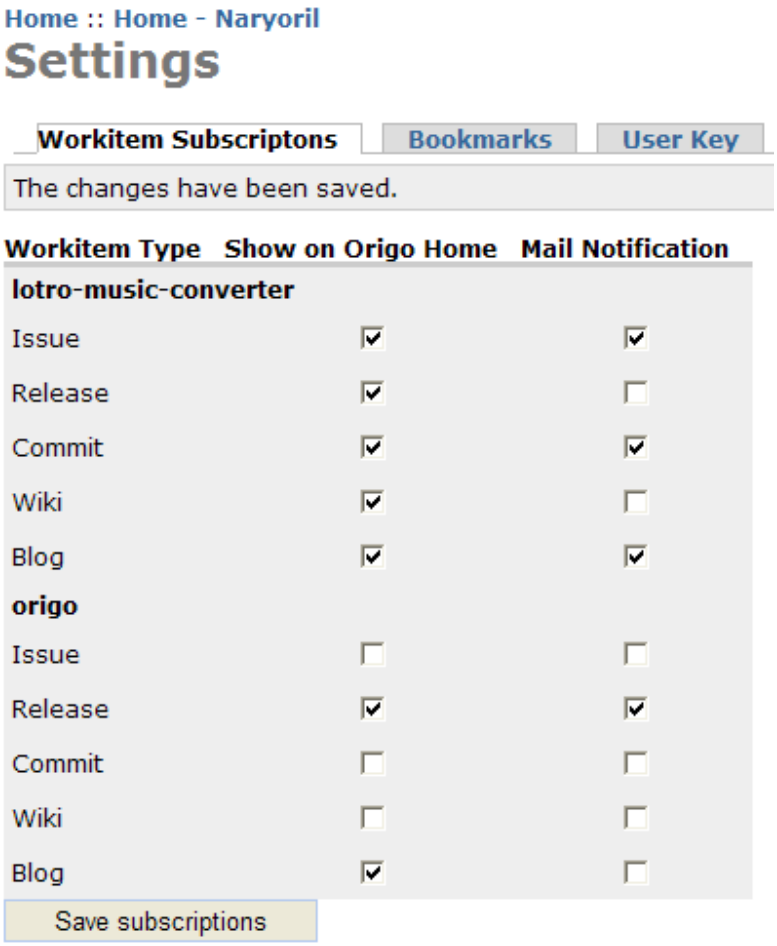


Figure 8.5: Workitem subscription page

As I mentioned above, you can also receive the workitems of a project you have bookmarked. Say, we want to be notified about new releases of the Origo IDE plug-ins, but we are not members of the Origo project we can do this. First visit the Origo project page (*http://origo.ethz.ch*) and go back to the user settings. There select the tab *Bookmarks* and then click on *[Bookmark this project (origo)]* and you will see that Origo was added to the bookmarked projects. Now select the *Workitem Subscriptions* tab once again and you will see that Origo is listed there too now. Here we want to be notified about releases and blogs on Origo home and about releases by e-mail. Again, simply check/uncheck the corresponding checkboxes and click on *Save subscriptions*. Figure 8.5 shows how it should look now.

Now let's visit Origo home again (remember, select *Home* - <*username*>

Figure 8.6: Origo Home - Workitem List

in the menu) and you will see the change on the wiki page we made earlier. Click on the *origo* tab and you will see a few releases and blogs (see Figure 8.6). You can mark a workitem as read by checking the checkbox or by clicking on the workitem, which also forwards you to the corresponding web page (e.g. the download page for a release or the blog entry).

You will see the same workitems in your IDE plug-ins as in Origo home. You can read how you install and use the IDE plug-ins in Chapters 4 - 7.

8.6 Issue Tracker

Let's talk about the issue tracker. Users can submit issues there, with issues we mean bugs and feature requests, but they are handled exactly the same. To see the issue go to the corresponding project and click on *Issues* at the top. Now you should see a list of them, if there are any (take a look at the issue tracker of the Origo project itself, there are a lot). First the ones with a, well... rose background. These are the open issues. Afterwards there are

the closed ones which have a light green background color.

To submit a new issue click on *Create new issue report* at the bottom. Then just fill in a meaningful (we really mean **meaningful**) title and a description that is as exact as possible. These are the only fields you see if you aren't a member of a project. But if you are a developer you have three additional fields. The first is *Status* which simply tells whether the issue is open or closed. Next there is *Assigned to* where you can choose who is responsible to solve that issue. And finally the *Issue Tags*. There you can type in whatever you want like "feature request" or "critical" or to which part of the program it belongs like "web" or "ide". The tags will be shown in the issue list (you can also see that in the issue tracker of the Origo project). By replying to an issue (first click on the issue and then on *Post Reply*) you can change the fields mentioned above.

8.7 Creating a Release

Let's say you have programmed for a week and your program is in a state where you want to release it such that all people can download it from Origo. The first step is to upload it to your Origo FTP account. One way to do this is to use a normal FTP client and connect to *upload.origo.ethz.ch* by using your normal Origo username and password. The other way is to visit your project's Origo page and to select *Create Release* in the menu. Browse for your file(s) and click on *Next*. This (or clicking *Upload more*) will also upload the files to your FTP account.

On the next page you have to enter a name for your release version, a version number and a description. On the bottom there is a list of all files that are on your FTP space. For each file that is part of a release select the corresponding platform from the dropdown list. Figure 8.7 shows an example of how it could look like. 8.6). Then click on *Create Release* and your release will be created and the selected files will be deleted from your FTP space. All that's necessary to create a release, including uploading, is also possible by using one of the IDE integrations as described in Chapter 4. Now click on *Download* at the top and you see that your release was created and can be downloaded by anyone.

That's it, you know now the basics of how Origo can be used. If there are any other questions don't hesitate to ask in the forum of the Origo project or by writing an e-mail at support@origo.inf.ethz.ch

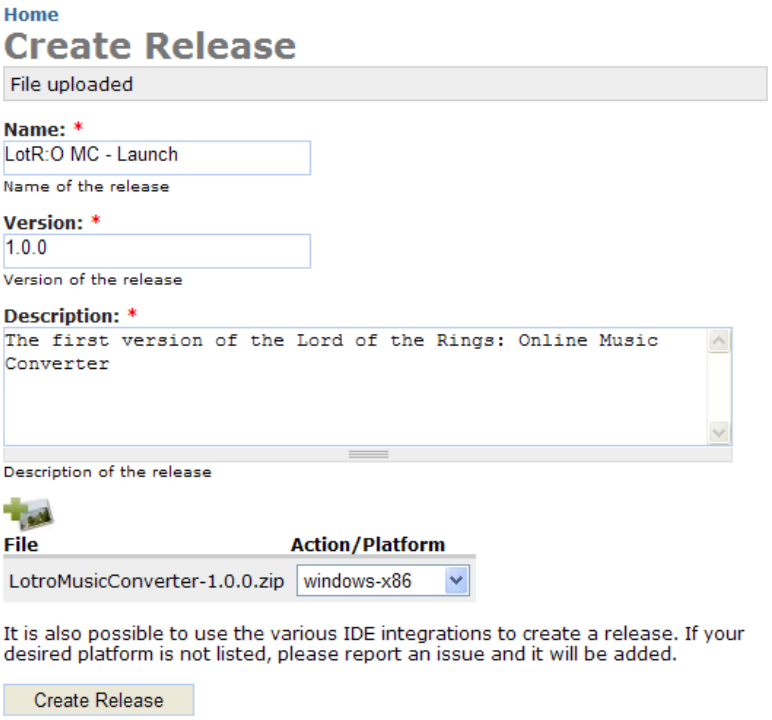


Figure 8.7: Create Release Form

Chapter 9

Future Work

There are several additional features that could be implemented and we will implement some of them. The most important would be that you can mark workitems as read or as unread so that you can really use the "show only unread workitems" feature and to fully support issue workitems which is not possible at the moment because Origo does not provide the needed information. Furthermore the possibility to change the workitem subscriptions directly from the IDE. We could provide a way to display workitems in the IDE similarly to the way they are shown on the Origo website by using the `workitem.list_projects` API call. Additionally in future we could extend the command line tool such that it implements all API calls that Origo provides and not only the ones we needed for our integrations. Also an integration to the Origo issue tracker could be done.

List of Figures

Bibliography

[CGE] *CodeGuru: Extending Visual Studio 2005.* http://www.codeguru.com/csharp/.net/net_vs_addins/visualstudioaddins/article.php/c11835/.

[dB] Berend de Boer. *e-POSIX, the complete Eiffel to POSIX binding.* http://www.berenddeboer.net/eposix/.

[Ecla] *Eclipse - an open development platform.* http://www.eclipse.org/.

[Eclb] *Get started with the Eclipse Platform.* http://www.ibm.com/developerworks/opensource/library/os-eclipse-platform.

[Eif] *EiffelStudio - A Complete Integrated Development Environment.* http://www.eiffel.com/products/studio/.

[Goa] *Project Goanna, The Eiffel Web Application Framework.* http://goanna.sourceforge.net/.

[Visa] *Microsoft Visual Studio 2005.* http://msdn2.microsoft.com/en-us/vstudio/.

[Visb] *Microsoft Visual Studio Extensiblilty Portal.* http://www.devx.com/vstudioextensibility.

[Visc] *Visual Studio Automation Object Model.* http://msdn2.microsoft.com/en-us/library/za2b25t3(VS.80).aspx.