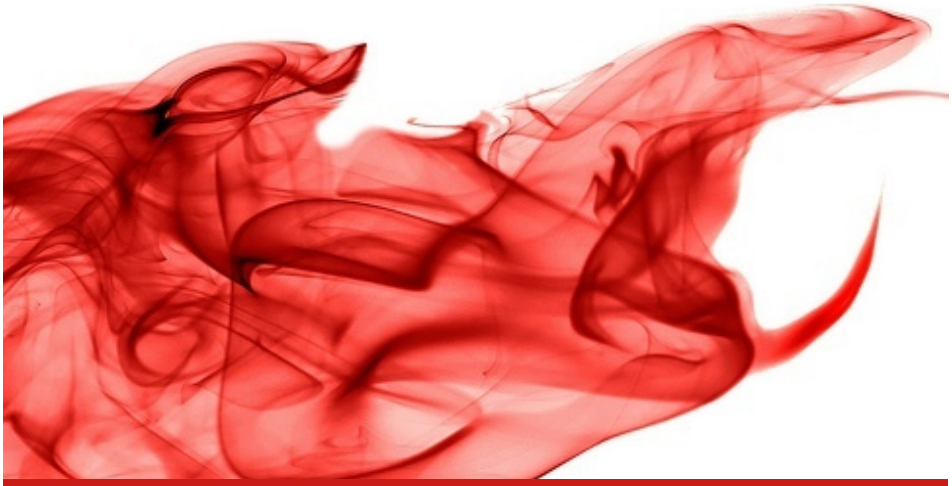
Berner Fachhochschule

Hochschule für Technik und Informatik
Software-Schule Schweiz

FLOW3

FLOW3 Development Environment for Eclipse



Master Thesis
MAS-06-01.01

Abstract

Design and develop an Eclipse-based development environment for FLOW3 (an open-source PHP application framework).

Author

David Brühlmeier
Meisenweg 7
3186 Düdingen
Switzerland
david@bruehlmeier.com

Supervisor

Robert Lemke
Rathenaustrasse 23
23568 Lübeck
Germany
robert@typo3.org

Expert	Philippe Seewer
Class	MAS-IT 2006/02
Date	2008-07-25

Revision History		
Revision 001	2008-03-18	David Brühlmeier
Initial Revision		
Revision 002	2008-07-22	David Brühlmeier
First Draft		
Revision 003	2008-07-23	David Brühlmeier
Added Chapters "Tests" and "Tracking"		
Revision 004	2008-07-24	David Brühlmeier
Added Chapters "Management Summary" and "Conclusion"		
Revision 005	2008-07-25	David Brühlmeier
Final Version		

Table of Contents

Chapter 1. Management Summary

The goal of this master thesis was to design and develop an Eclipse-based development environment for FLOW3 (an open-source PHP application framework). During the past six months, I have worked hard to achieve this goal and I am proud to deliver the results in the form of this report and a fully operational Eclipse plug-in, called DEV3.

DEV3 offers some useful features to develop FLOW3 packages, such as

- A wizard to create new FLOW3 packages
- A wizard to create new aspects
- An editor for the `Package.xml` file
- An editor for TypoScript 2.0 with syntax highlighting and folding as well as a protype implementation of syntax checking, code completion and an outline view

During the development of this master thesis, I was able to apply many of the skills I have acquired during my studies at the UASB, including the following:

- Project Management
- Requirements Engineering
- Object-oriented Software Design with UML
- Advanced Programming in Java
- XML Technologies, such as XSLT, XSL-FO, XSD, RelaxNG and JAXB
- User Interface Programming in SWT
- Grammar Definition in EBNF

Additionally, I learned a lot about the Eclipse platform and its architecture.

The Eclipse plug-in which resulted from this master thesis is published under the Eclipse Public License and can therefore be used and developed by anybody interested. In the spirit of open source, I have made contact with other interested parties early on and shared results often. This has already resulted in a small community and I am confident that DEV3 will be developed further in the future.

Chapter 2. Introduction

2.1. Purpose

This document is the final documentation of the master thesis of David Brühlmeier. It describes the design and the user interface of DEV3, covers the test results and assesses the fulfillment of the requirements with the actual implementation. The document concludes with a number of lessons learned during the whole project.

The requirements specification, the user documentation and the generated JavaDoc files are not part of this document. They are referenced in the appendices and available as separate documents.

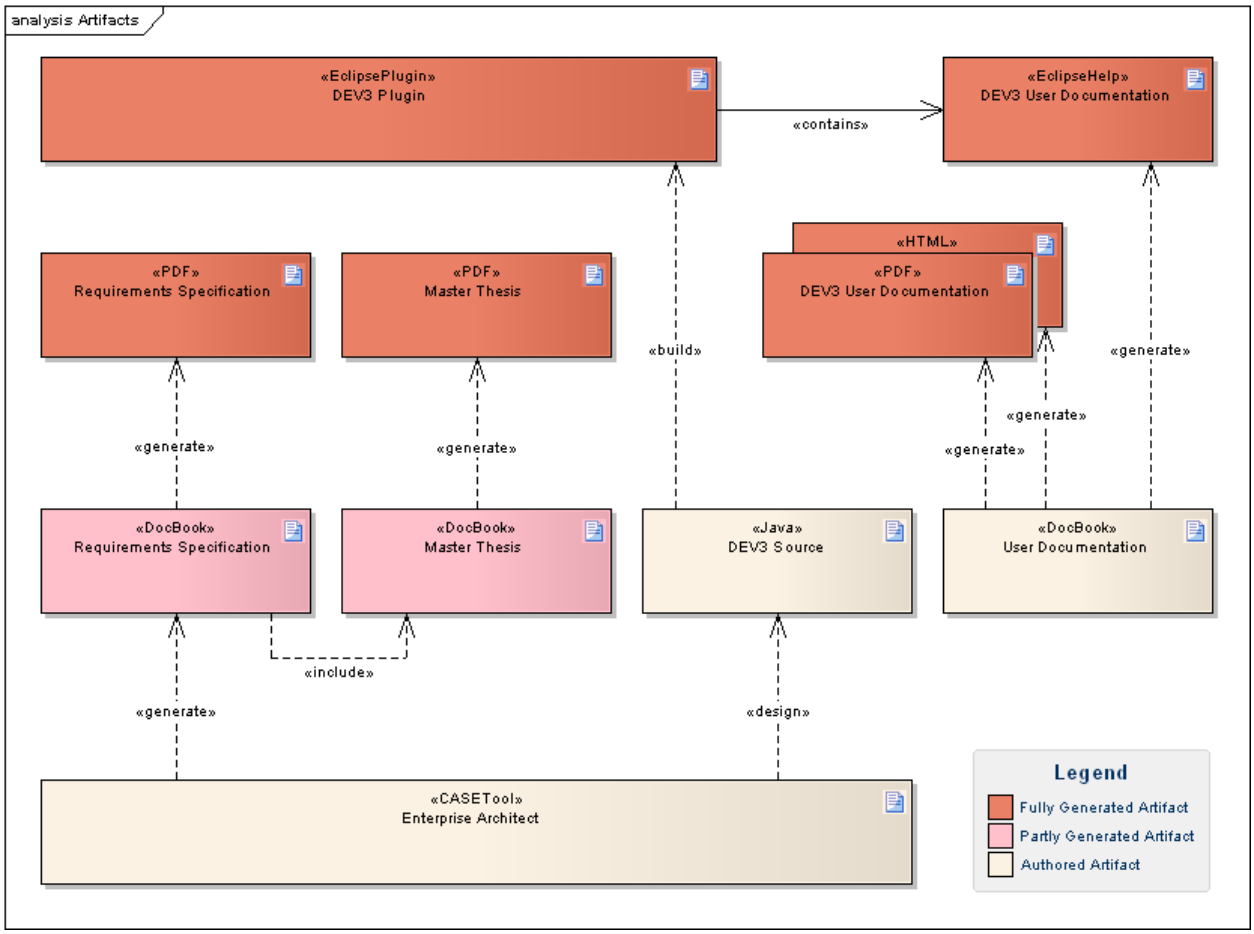
2.2. Intended Audience

This document is primarily intended to be read by the roles *Expert* (Philippe Seewer) and *Supervisor* (Robert Lemke). However, this document will also be published and is therefore also intended for any party interested in the development of DEV3.

2.3. Artifacts

This document has been written in DocBook, an open XML format for technical documents, and has been rendered using XSL technologies.

Below is an overview of the artifacts produced by this master thesis.



Artifacts Produced By This Master Thesis

2.4. Name of the product

When this project started, the product was intended to be published as open source from the very beginning. The intention of this master thesis was to deliver a product to the TYPO3/FLOW3 community which could then be taken over by a team of interested developers for further improvement. Shortly after the beginning of the master thesis, a community initiative started with a very similiar goal as this master thesis, but with a focus on the TYPO3 version 4.x string. Robert Lemke therefore proposed both parties to join efforts. This has lead to an even earlier joint-development with the community as initially planned.

The other initiative was named "TyClipse", the product of this master thesis was intended to be named "FLOW3DE" (FLOW3 Development Environment). Since both products were going to be combined, we decided to come up with a new name, which is "DEV3". "DEV" stands for Development Environment and "3" is the common denominator between TYPO3 and FLOW3.

For this reason, the product of this master thesis is called "DEV3". More precisely, the product of this master thesis are all *FLOW3-related features* of "DEV3".

2.5. Websites

There are two important websites with additional information about DEV3:

- General information can be found on the project site `www.dev3.org`
- Development takes place on TYPO3 Forge: `forge.typo3.org/projects/show/team-dev3`

These websites will be maintained even after the completion of this master thesis.

Chapter 3. Design

3.1. Overview

The whole system is completely implemented as an Eclipse Plug-In. Eclipse features (e.g. JFace, Logging, etc.) and the Eclipse API (e.g. to access the file system) are preferrably used over native Java to assure consistency with other Eclipse Plug-Ins.

DEV3 contains features for both TYPO3 and FLOW3. The TYPO3-related features all reside in a separate plugin with the namespace `com.crosscontent.typo34`. They are *not* part of this master thesis.

The DEV3 Plug-In is divided in the following packages.

- `org.typo3.forge.dev3.core`

  Core classes required by the Eclipse plug-in infrastructure as well as commonly used classes such as logging.

- `org.typo3.forge.dev3.model`

  Classes which belong to the model part of the MVC pattern.

- `org.typo3.forge.dev3.ui`

  Classes which belong to the view part of the MVC pattern.

- `org.typo3.forge.dev3.exceptions`

  Exceptions used throughout the plug-in.

- `org.typo3.forge.dev3.preferences`

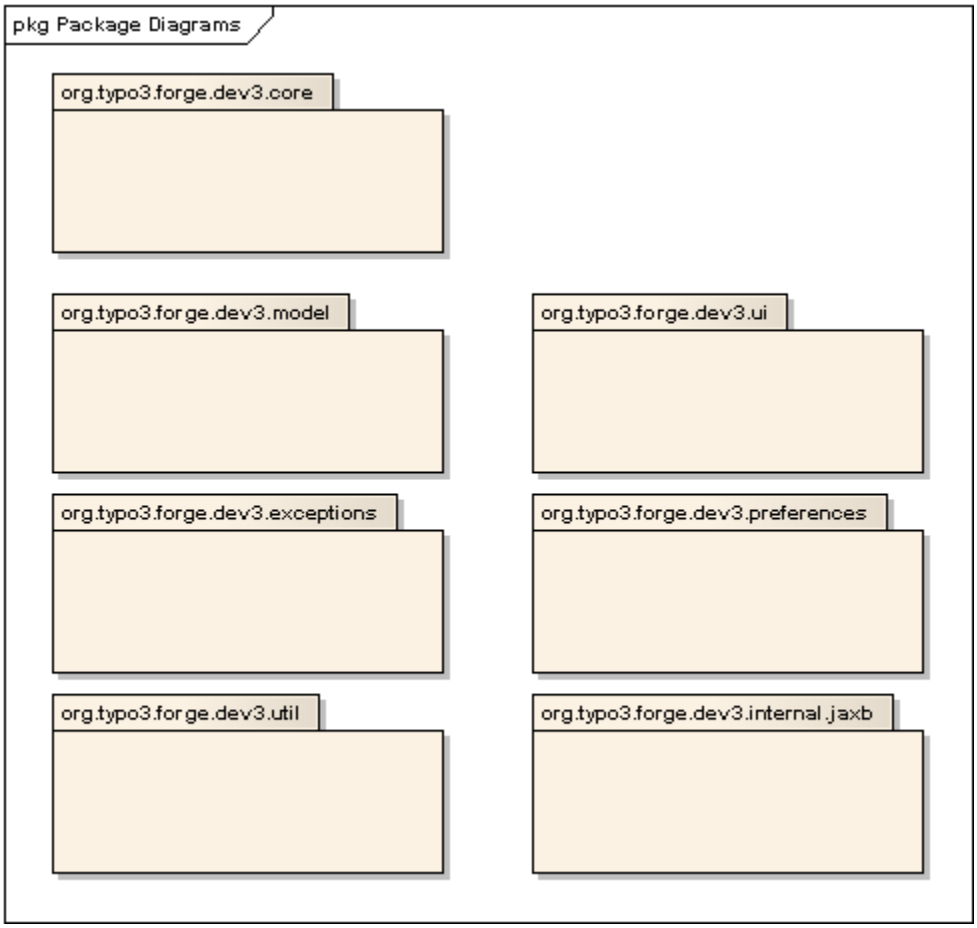
  Preferences for the plug-in, managed by the Eclipse preference framework.

- `org.typo3.forge.dev3.util`

  Utility classes used throughout the plug-in.

- `org.typo3.forge.dev3.internal.jaxb`

  Classes generated by the JAXB compiler for XML access.



Package Diagram of DEV3

Most of the functionality lies in the packages `org.typo3.forge.dev3.model.*` and `org.typo3.forge.dev3.ui.*`. For this reason, these packages are described in detail in the following sections.

3.2. Design Principles

The design principles for DEV3 are a number of rules I followed during the development of DEV3.

3.2.1. No PHP Editor

DEV3 does not provide its own PHP editor. It simply makes no sense and would be a waste of resources. There are a number of PHP editors available for Eclipse, e.g. `PDT`, `PHPeclipse`, `Aptana Studio` or `Zend Studio`.

3.2.2. Independant of PHP Editor

Because of the number of PHP editors available, DEV3 wants to stay independant of a PHP editor. DEV3 can even be used without a PHP editor installed.

3.2.3. Follow MVC Pattern

DEV3 follows the Model-View-Controller pattern because of its benefits in clearly defining the responsabilities of the different domains.

3.2.4. Model independant from Eclipse

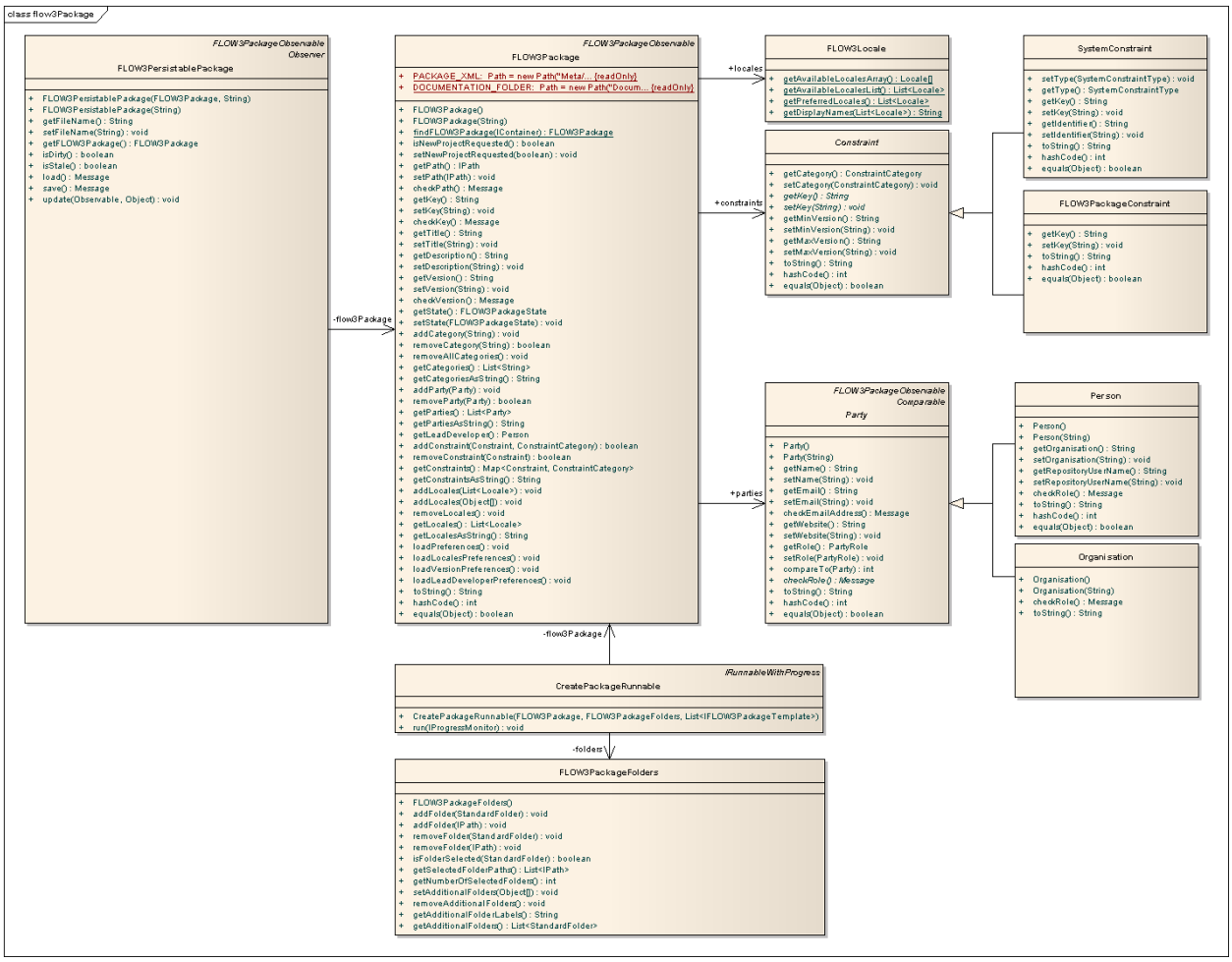
The DEV3 model classes try to be as independant from Eclipse as possible. This results in better testability because of fewer interdependencies and better reusability in other projects.

3.3. Model Packages

In this section, the most important classes from the `org.typo3.forge.dev3.model.*` packages are described in detail.

3.3.1. FLOW3 Package Model

The package `org.typo3.forge.dev3.model.flow3Package` contains the model for the FLOW3 Package. The class `FLOW3Package` is the central class of this package. It serves as "model" in the MVC pattern. In the "New Package Wizard", a new instance of `FLOW3Package` is created and the data entered by the user is saved in this object. When editing an existing package, an instance of `FLOW3Package` is created by the package editor.



Class Diagram of the package `org.typo3.forge.dev3.model.flow3Package`

Persistency is not handled by `FLOW3Package` itself, but rather by the class `FLOW3PersistablePackage`. I have decided to separate persistancy to respect the "Separation of Concerns". The `FLOW3PersistablePackage` offers methods to save an instance of `FLOW3Package` as an XML file and to

load contents from such an XML file. For the saving to / loading from XML, I have decided to use JAXB for the following reasons:

- JAXB offers an object-oriented way to access XML data.
- It is relatively easy to use.
- The classes to access the XML data can be generated from the published XML schema for FLOW3 Packages, which ensures compliance with the norm.

I have decided against directly using the classes generated by JAXB as "model" for the following reasons:

- The model shall offer additional functionaliy, such as validation methods which go beyond the validation based on the XML schema.
- It shall be possible to re-generate the classes in case the schema changes without having to adjust the model.
- The generated classes are rather complex and I did not want to use a bindings file which, in my opinion, adds yet another level of complexity and an additional source for errors.

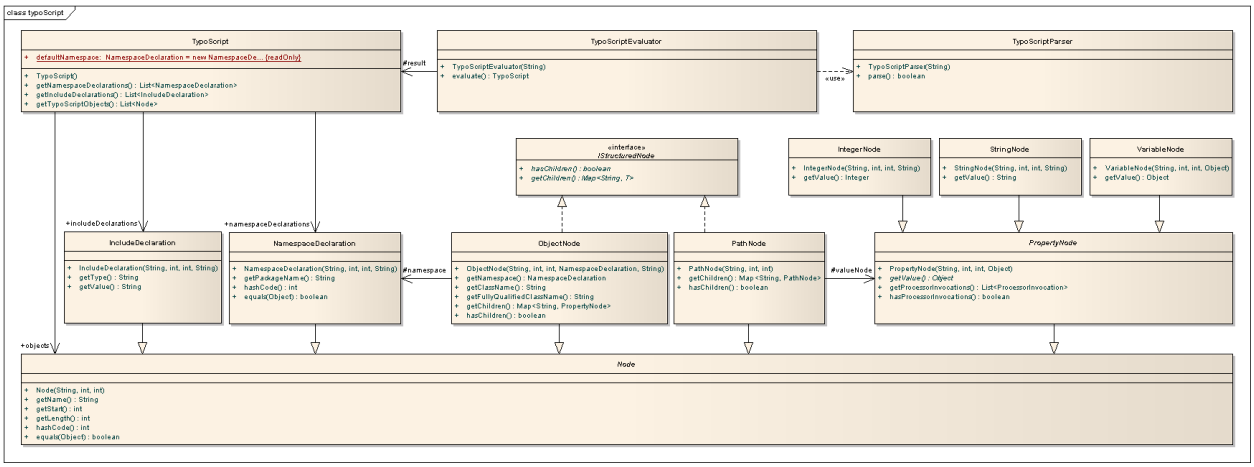
`FLOW3Package` and all associated classes are acting as "Observables" in the "Observer" pattern. This is necessary because of the modular implementation of the view, which results in multiple views displaying the data from the same `FLOW3Package` at the same time. In order to keep all views up-to-date, the view register themselves as "Observer" and are updated on each change. I have decided to add an abstract base class `FLOW3PackageObservable` to take over the Observable concern. I have decided to base this class on `java.util.Observable.Observable` for the following reasons:

- The methods for registering, unregistering and notifying Observers are already implemented.
- As part of the JDK, this class is well-established and tested.

All Obsverables in this package fire events of the type `FLOW3PackageChangeEvent`. This decision was meant as a simplification in order to avoid a plethora of classes, but with a hindsight, I would not do so again. It results in an akward mixture of attributes whenever a specific information is needed in the event which should only be contained in an event for a specific class.

3.3.2. TypoScript 2.0 Model

The package `org.typo3.forge.dev3.model.typoScript` contains the model for TypoScript 2.0. The classes `TypoScript` and `TypoScriptParser` and `TypoScriptEvaluator` are the central classes in this package.



Class Diagram of the package `org.typo3.forge.dev3.model.typoScript`

`TypoScriptParser` is responsible for parsing a string according to the grammar of TypoScript 2.0. If it detects an error, it throws a `ParseException`.

`TypoScriptEvaluator` uses `TypoScriptParser` to parse a String of TypoScript 2.0. If no `ParseException` is thrown, it evaluates the TypoScript 2.0 and returns an instance of `TypoScript`, which serves as model.

TypoScript consists of three lists:

1. A list of include declarations which describe the files to be regarded as part of this TypoScript 2.0 definition.
2. A list of namespace declarations which define the namespaces used in this TypoScript 2.0 definition (similar to Java `import` statements).
3. A list nodes which represent the actual `"payload"`, i.e. the parsed TypoScript 2.0 tree.

The TypoScript 2.0 tree is a recursive list of nodes. All elements in TypoScript 2.0 are subclasses of `Node`.

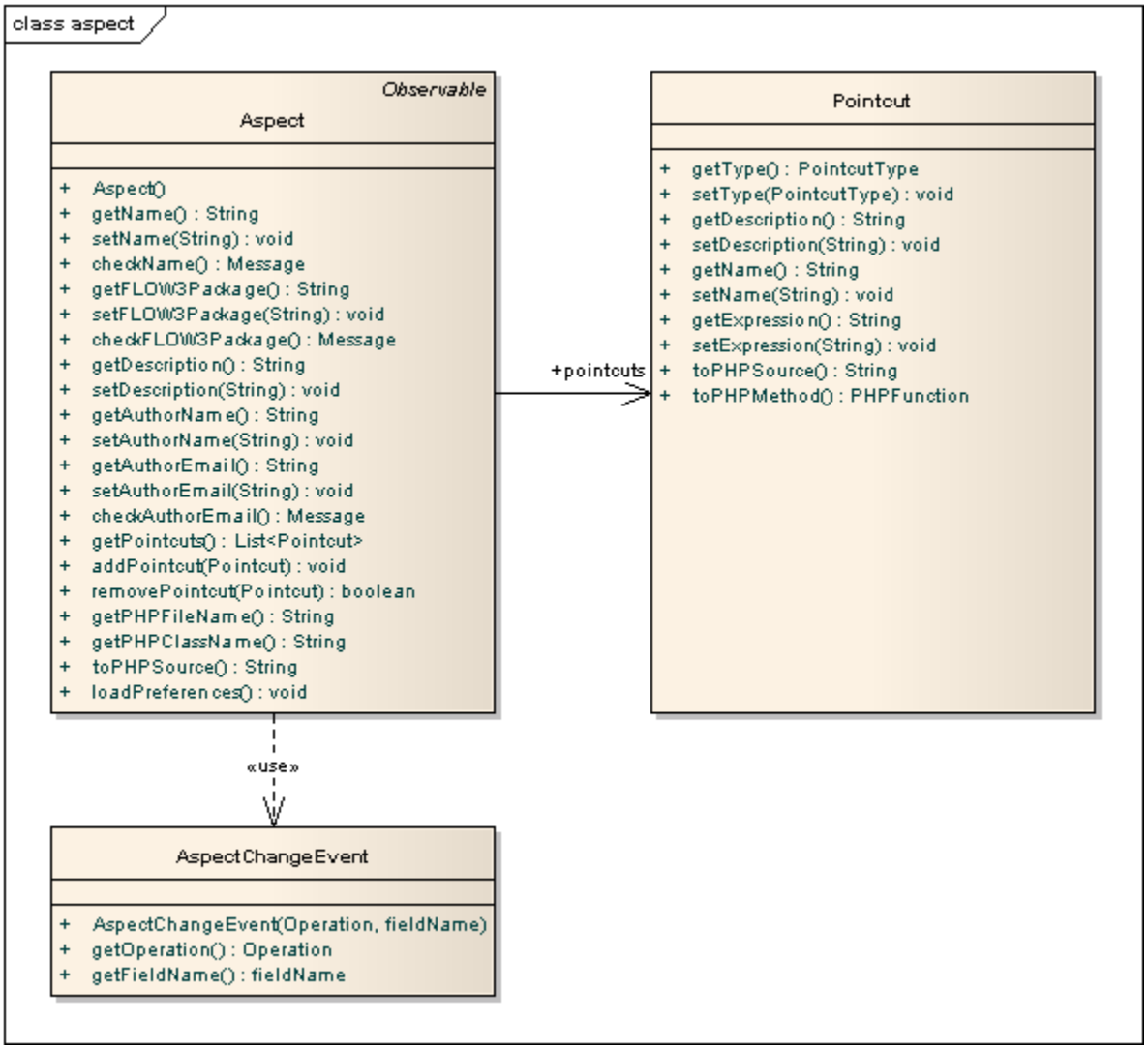
- `IncludeDeclaration`: Describes a file to be a part of the TypoScript definition (before parsing).
- `NamespaceDeclaration`: Describes a namespace to be used in this TypoScript definition.
- `ObjectNode`: Describes the instance of a TypoScript class.
- `PathNode`: Describes the path to a TypoScript object (has no type).
- `PropertyNode`: Abstract base class for all nodes carrying a value, i.e. `IntegerNode`, `StringNode` and `VariableNode`.

ObjectNode and PathNode are the only nodes which can have children. This additional concern is described in the Interface `IStructuredNode` which is implemented by both classes.

Even though all TypoScript elements are subclasses of Node, I have decided against using just one list in TypoScript because the declarations (`IncludeDeclaration` and `NamespaceDeclaration`) rather have the nature of meta-data, in contrast to the other classes which represent the actual payload.

3.3.3. Aspect Model

The package `org.typo3.forge.dev3.model.aspect` contains the model for Aspects. The central classes in this package are `Aspect` and `Pointcut`.



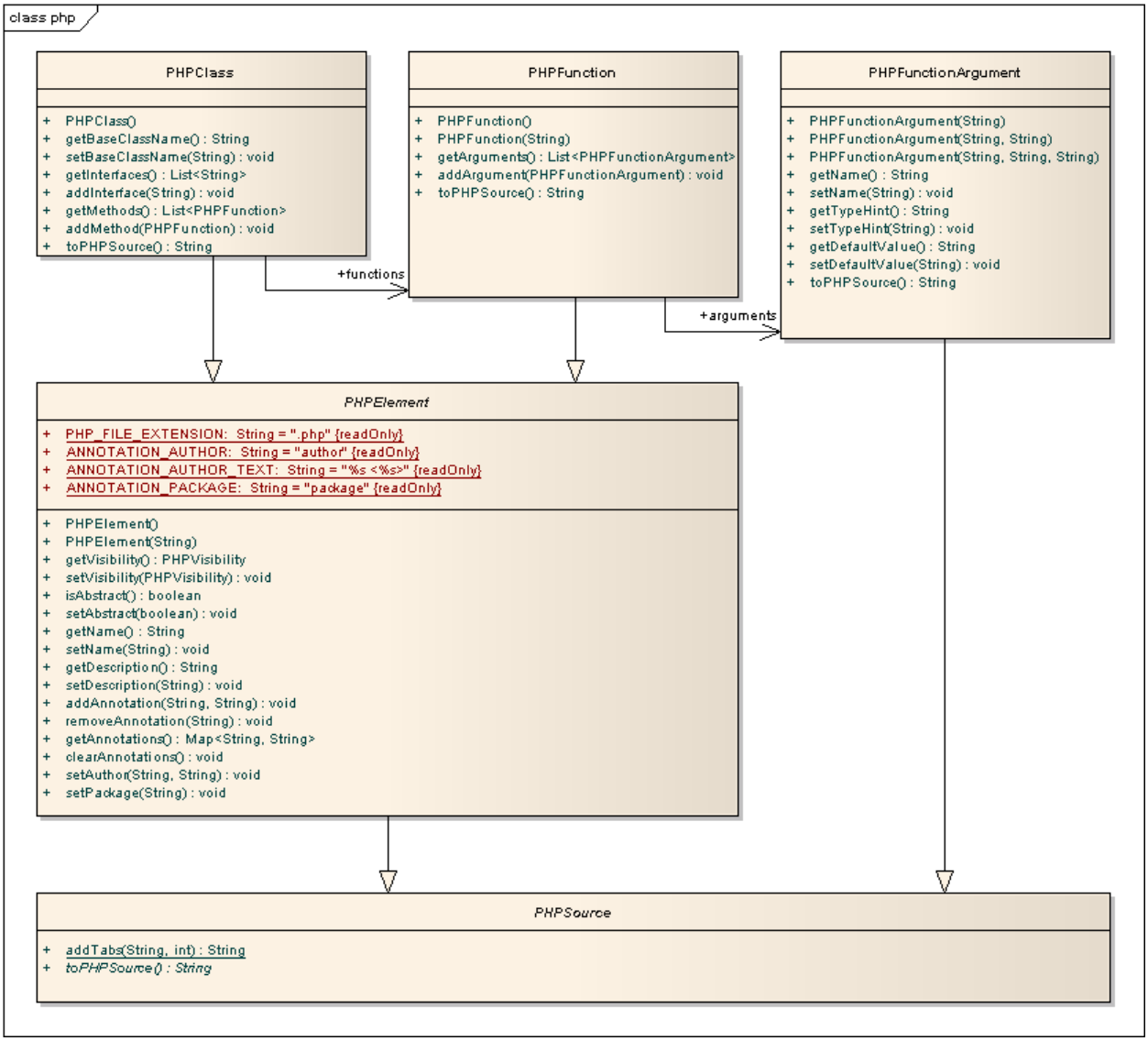
Class Diagram of the package `org.typo3.forge.dev3.model.aspect`

`Aspect` serves as model for the "New Aspect Wizard". Similar to the concept used in `FLOW3Package`, this class does not offer persistency. It does however offer a method to convert itself to a string representing PHP source code (using classes from the package `org.typo3.forge.dev3.model.php`) . This is a slight break of the MVC pattern because rendering belongs to the view, not to the model. I chose this solution because of its simplicity.

`Pointcut` also serves as model for the "New Aspect Wizard". The same principles regarding persistency and rendering to PHP source code apply as for `Aspect`.

3.3.4. PHP Model

The package `org.typo3.forge.dev3.model.php` contains a very basic model for PHP source code. I chose to implement this myself rather than using an existing implementation (i.e. from the PHP Development Tools - PDT - by Zend) because I didn't want to create a dependency for DEV3 on a certain PHP editor. The central classes in this package are `PHPClass`, `PHPFunction` and `PHPFunctionArgument`.



Class Diagram of the package `org.typo3.forge.dev3.model.php`

The base class for all PHP source code is `PHPSource`. It is abstract and asks the implementors to provide a method which allows to convert itself to PHP source code.

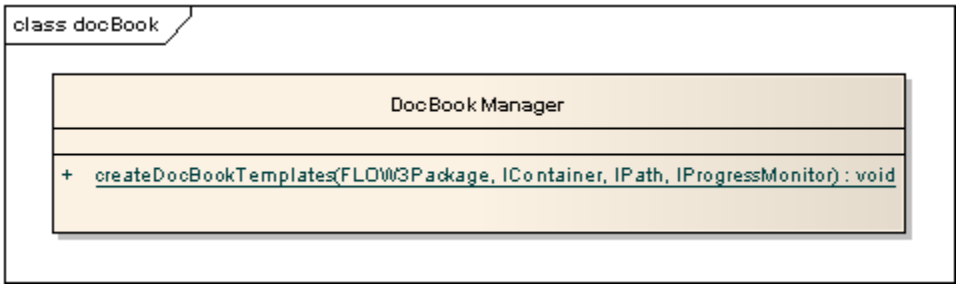
"PHP elements", i.e. classes and functions, share common features such as visibility, descriptions, annotations, etc. Therefore I decided to implement these common features in the abstract class `PHPElement`.

The whole package only features a partial representation of PHP source code. I only implemented the parts which are currently needed for DEV3. Therefore, `PHPClass` only knows its base class, the names of the interfaces it implements and the list of its functions (instances of `PHPFunction`).

`PHPFunction` knows the list of its arguments (instances of `PHPFunctionArgument`) and `PHPFunctionArgument` is fully modelled according to PHP5, including type hints.

3.3.5. DocBook Model

The package `org.typo3.forge.dev3.model.docBook` currently contains only a single class `DocBookManager`, which is responsible for handling DocBook files serving as documentation of a FLOW3 Package.



Class Diagram of the package `org.typo3.forge.dev3.model.docBook`

I originally intended to implement `DocBookManager` using JAXB, with the same design as `FLOW3Package`. However, JAXB does not seem to handle `xi:include` statements, but these are essential for DocBook files. Due to the time constraints, I have therefore decided to drop using JAXB. Since all that is currently needed is just a replacement of a few strings, I have implemented this as a simple line-based search-and-replace method.

3.4. UI packages

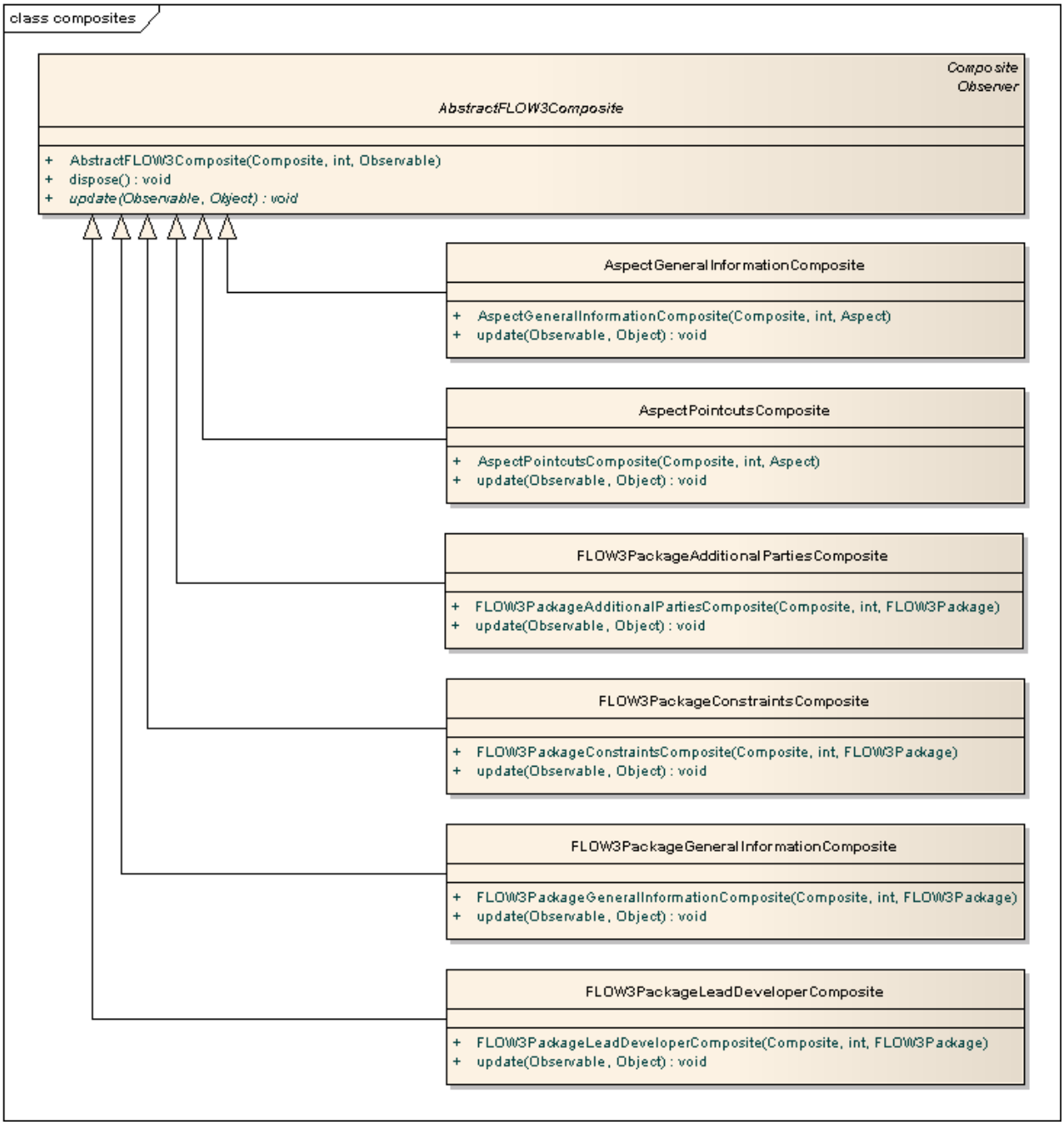
In this section, the most important classes from the `org.typo3.forge.dev3.ui.*` packages are described in detail.

3.4.1. Composites

The package `org.typo3.forge.dev3.ui.composites` contains all SWT composites for DEV3. I have decided to create composites for all GUI elements which are used in more than one class. For instance, many of the elements in the "New Package Wizard" are reused in the "FLOW3 Package Editor". This approach has two major advantages:

- Re-use of code: "DRY "principle - Don't Repeat Yourself.
- Consistent user interface: Things which do the same, look the same.

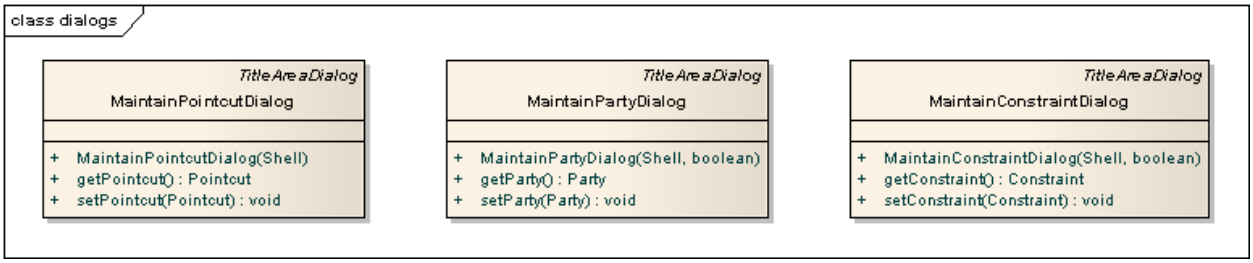
All composites are extending the abstract base class `AbstractFLOW3Composite`. This class is responsible to register the composite as listener for the respective model. The `update()` method is delegated to the concrete implementations.



Class Diagram of the package `org.typo3.forge.dev3.ui.composites`

3.4.2. Dialogs

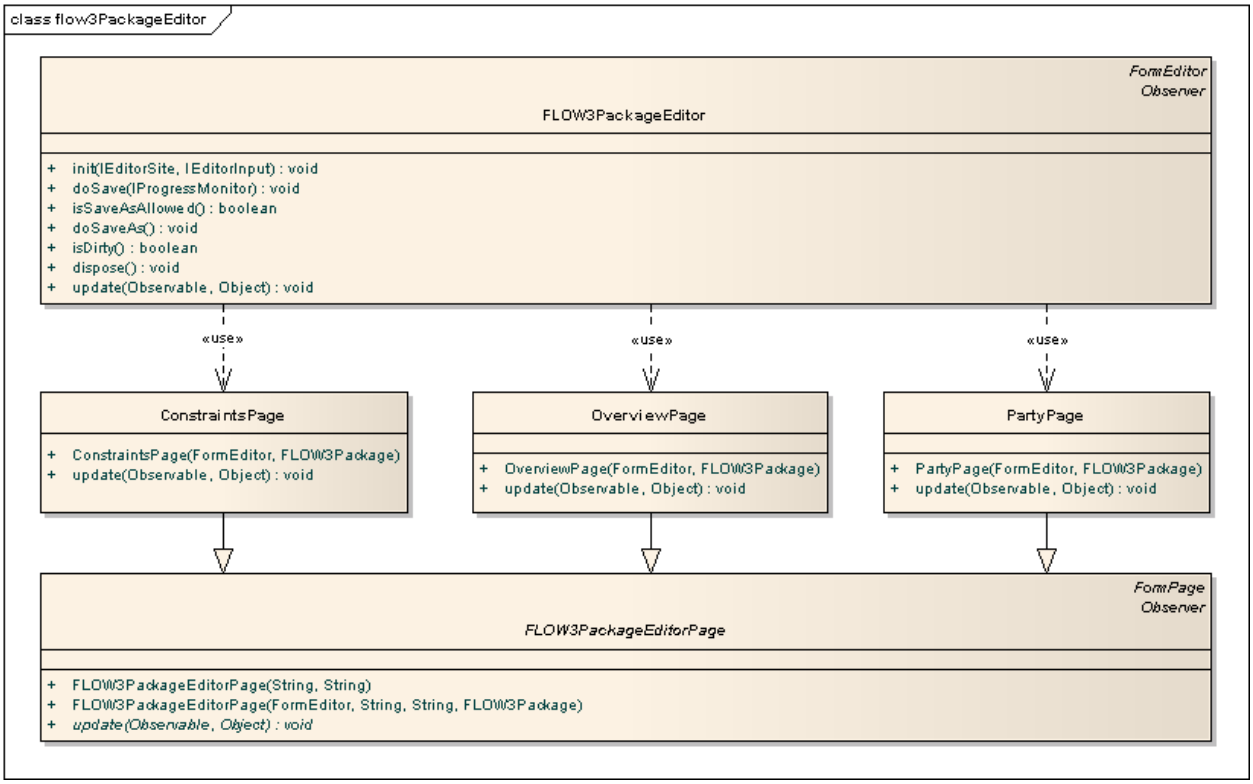
The package `org.typo3.forge.dev3.ui.dialogs` contains all SWT dialogs for DEV3. All dialogs are subclasses of `org.eclipse.jface.dialogs.TitleAreaDialog` to achieve a consistent look-and-feel with other Eclipse plug-ins. I have chosen `TitleAreaDialog` to be able to present messages (i.e. error messages) to the user in realtime.



Class Diagram of the package `org.typo3.forge.dev3.ui.dialogs`

3.4.3. FLOW3 Package Editor

The package `org.typo3.forge.dev3.ui.editors.flow3PackageEditor` contains the editor and the editor pages for the "FLOW3 Package Editor". The class `FLOW3PackageEditor` is the main class which is registered as editor in `plugin.xml`. It is a subclass of `org.eclipse.ui.forms.editor.FormEditor`, a JFace editor which is able to display multiple pages, a concept used in several Eclipse plug-ins (e.g. the PDE). Upon instantiation, this class registers the three pages (`OverviewPage`, `PartyPage` and `ConstraintsPage`), which are all subclasses of the abstract base class `FLOW3PackageEditorPage`. This class is responsible to register the pages as listener for the model. The `update()` method is delegated to the concrete implementations.



Class Diagram of the package `org.typo3.forge.dev3.ui.editors.flow3PackageEditor`

3.4.4. TypoScript 2.0 Editor

The package `org.typo3.forge.dev3.ui.editors.typoScriptEditor` contains the TypoScript 2.0 editor and all helper classes for this editor.

The class `TypoScriptEditor` is the main class which is registered as editor in `plugin.xml`. It is a subclass of `org.eclipse.ui.editors.text.TextEditor` which is the standard text editor in Eclipse. It serves as the base for most source code editors, including JDT and PDT. `TextEditor` mainly takes care of updating the outline viewer, markers and folding. Most of the specific functionality is configured through `TypoScriptSourceViewerConfiguration`, which is extending `org.eclipse.jface.text.sourceSourceViewerConfiguration`. This is the standard way to configure `TextEditor`.

`TypoScriptSourceViewerConfiguration` uses a lot of helper classes, which are responsible for various aspects of the TypoScript editor.

- `TypoScriptAutoIndentStrategy`: Auto indent line strategy sensitive to brackets. This class has been copied from the example java editor provided by IBM Corporation. No specific changes have been made for the TypoScript editor, due to time constraints.

- `TypoScriptDoubleClickSelector`: Double click strategy aware of TypoScript identifier syntax rules.

- TypoScriptTextHover: Text Hover for the TypoScript Editor. Currently just shows the element over which the cursor hovers. This will need to be connected to the TypoScript model in the future so that e.g. hovering over a variable shows where the variable is defined and what its contents are.

- `TypoScriptCompletionProcessor`: Responsible for computing the completion proposals.

  Currently, this is only a prototype implementation. It reads the available TypoScript classes from a file called TypoScript.xml. It is not yet decided if such a file will be part of the FLOW3 package structure. I have made a proposal to the FLOW3 team to include such a file, because it would allow editors, such as this one, to more easily offer code completion, withouth having to parse PHP source code.

  Furthermore, the computed proposals are not yet context-aware, but this is an important feature for code completion.

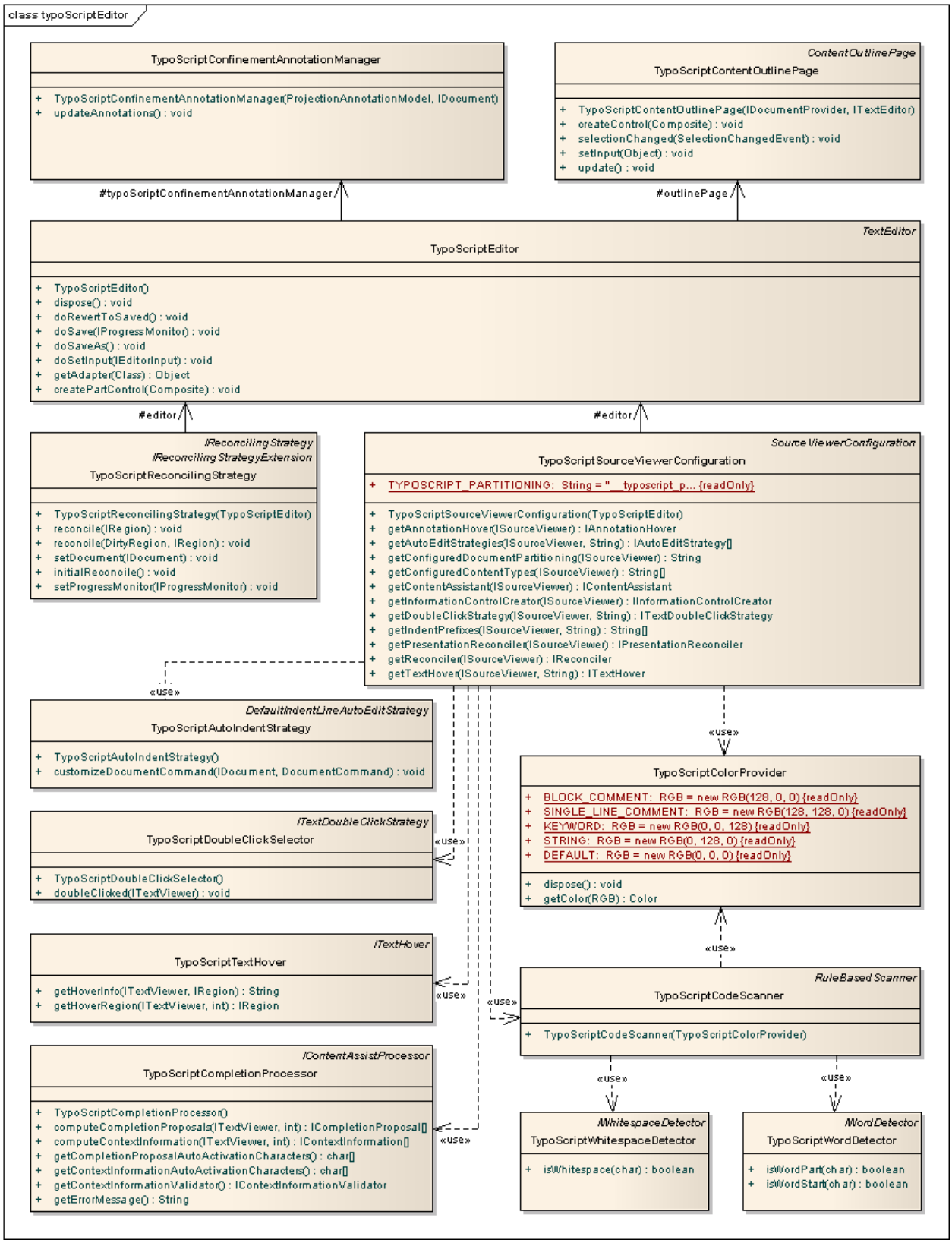
- `TypoScriptColorProvider`: Manager for colors used in the TypoScript editor.

- `TypoScriptCodeScanner`: Scans for syntactical elements like keywords, block comments, single line comments and whitespace. This is the base for syntax highlighting.

- `TypoScriptWhitespaceDetector`: Detects whitespace according to the TypoScript 2.0 Grammar.

- `TypoScriptWordDetector`: A TypoScript aware word detector. A word is a series of characters, for which `TypoScriptRules.isLetter()` returns true.

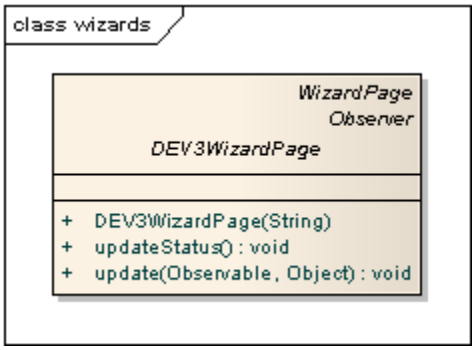
In order to provide custom partitioning, `TypoScriptDocumentSetupParticipant` also needs to be registered in `plugin.xml`. This class is responsible for the creation and connection of the `TypoScriptPartitionScanner` with the `IDocument` of `TextEditor`.



Class Diagram of the package `org.typo3.forge.dev3.ui.editors.flow3PackageEditor`

3.4.5. Wizards

The package `org.typo3.forge.dev3.ui.wizards` contains `DEV3WizardPage` which acts as the abstract base class for all wizard pages of DEV3. This class acts as an observer and updates its status whenever the model (the `FLOW3Package` instance) changes by calling the abstract method `checkAll()`. This method must be implemented by the concrete pages. It returns a list of messages (i.e. informations, warnings and error). This list is then sorted by severity and the most severe message is presented to the user in the status bar. If the `checkAll()` method returns at least one error, this class also disables the "Finish" button.



Class Diagram of the package `org.typo3.forge.dev3.ui.wizards`

3.4.6. New Package Wizard

The package `org.typo3.forge.dev3.ui.wizards.newPackageWizard` contains the "New Package Wizard" and its pages.

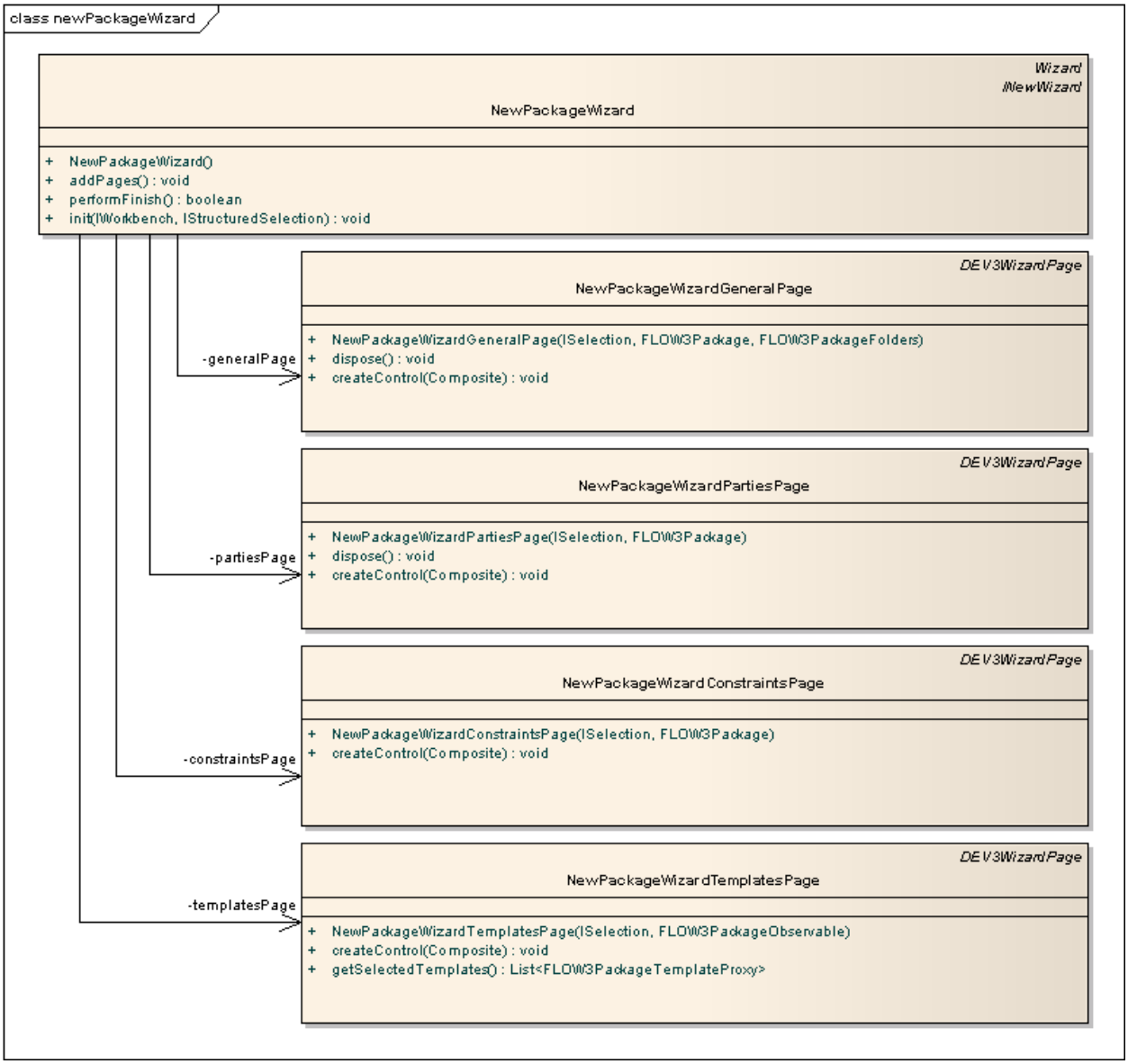
The class `NewPackageWizard` is the main class which is registered as wizard in `plugin.xml`. It extends `org.eclipse.jface.wizard.Wizard`, which is the standard class for Eclipse wizards. This class registers the four pages of the wizard:

1. `NewPackageWizardGeneralPage`: General information about the FLOW3 package, such as the key, the title, the description, etc.

2. `NewPackageWizardPartiesPage`: Information about the parties involved in the development of the package, notably the lead developer.

3. `NewPackageWizardConstraintsPage`: Information about system constraints (e.g. required memory, operating system, PHP version, etc.) and package constraints (i.e. packages which are needed for this package to run, conflicting packages or suggested packages).

4. `NewPackageWizardTemplatesPage`: Templates which are executed upon creation of the package. These templates can be provided by third-party plugins using the extension point `org.typo3.forge.dev3.flow3.package.templates`.



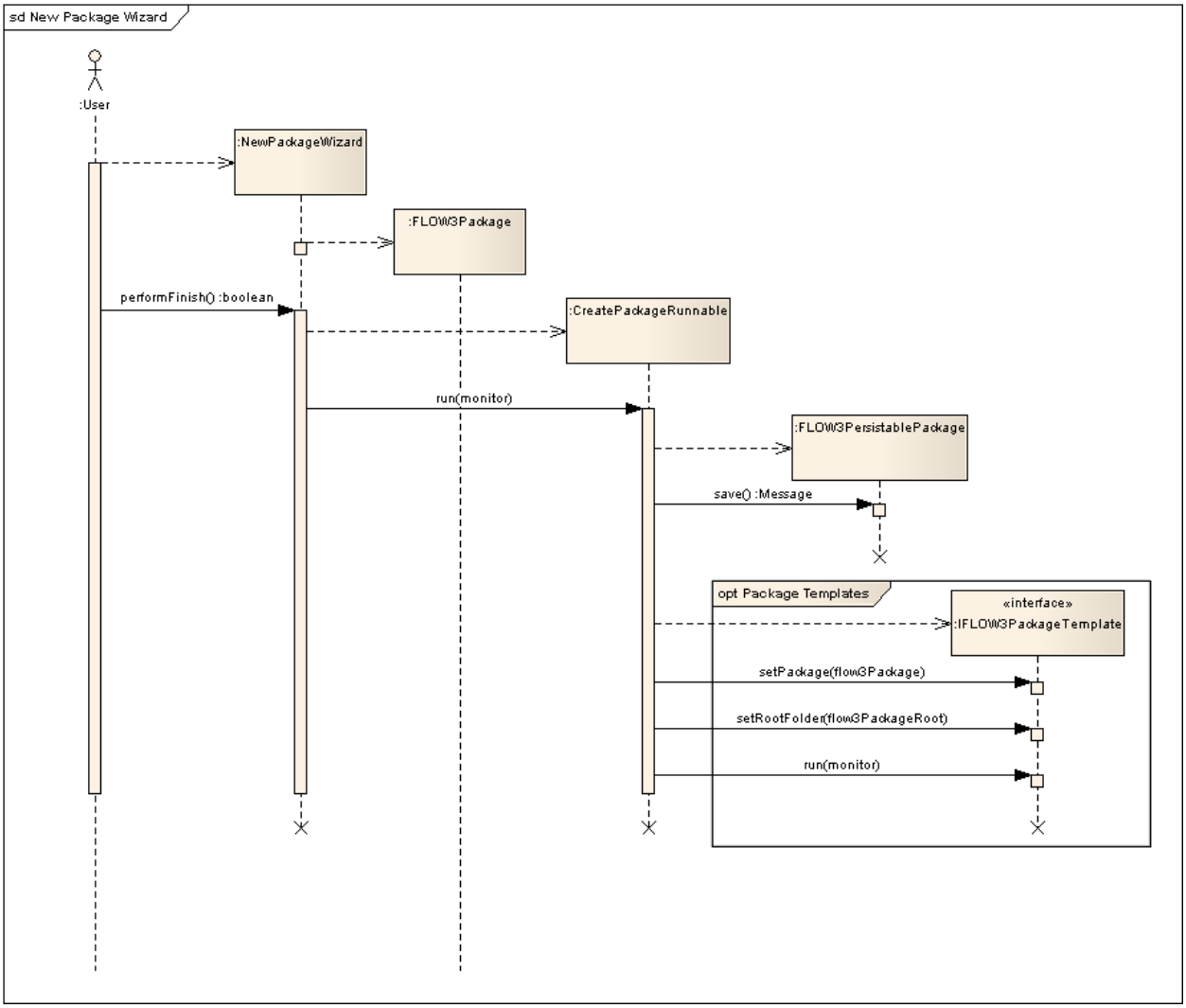
Class Diagram of the package `org.typo3.forge.dev3.ui.wizards.newPackageWizard`

The "New Package Wizard" is started by the user by selecting "File -> New -> Other -> FLOW3 Package". Eclipse then creates an instance of `NewPackageWizard`. The `NewPackageWizard` then creates instances of all wizard pages and a new instance of `FLOW3Package`, which serves as model for all wizard pages.

The user then enters all information in the respective wizard pages. When the user presses the "Finish" Button, the `performFinish` method is called by Eclipse. The `NewPackageWizard` now creates a `CreatePackageRunnable` instance and calls the `run()` method in a new thread.

The `CreatePackageRunnable` creates all folders, using Eclipse library methods. The `Plugin.xml` file is created by calling the `save()` method on the `FLOW3Package` instance. The `FLOW3PersistablePackage` uses the JAXB-generated classes to create the XML file.

If the user has selected any templates, they are instantiated and configured with the `FLOW3Package` and the root folder of the package. The `run()` method is called in the same thread as the `CreatePackageRunnable` instance runs in.



Sequence Diagram of the "New Package Wizard"

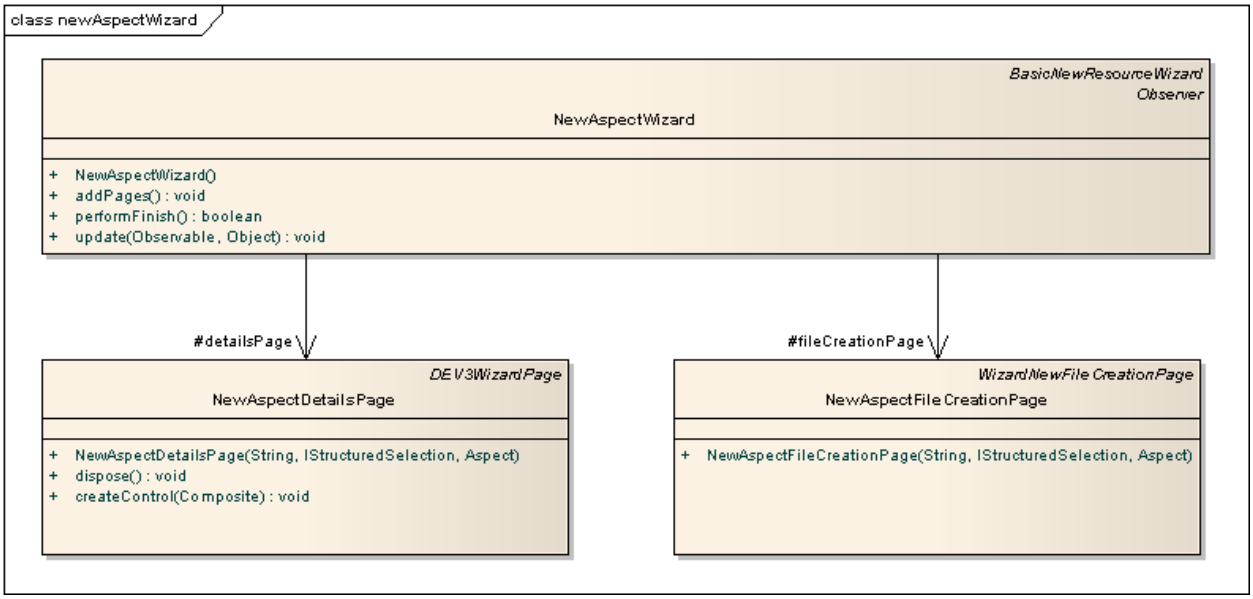
3.4.7. New Aspect Wizard

The package `org.typo3.forge.dev3.ui.wizards.newAspectWizard` contains the "New Aspect Wizard" and its pages.

The class `NewAspectWizard` is the main class which is registered as wizard in `plugin.xml`. It extends `org.eclipse.jface.wizard.Wizard`, which is the standard class for Eclipse wizards. This class registers the two pages of the wizard:

1. `NewAspectDetailsPage`: General information about the aspect, such as the name, the description, etc.

2. `NewAspectFileCreationPage`: This page is not extending `DEV3WizardPage` like all other wizard pages, but rather `org.eclipse.ui.dialogsWizardNewFileCreationPage`. This is because the functionality offered by this page (choose the location of a resource) is not specific to DEV3 and shall therefore be supplied in a consistent manner with other plug-ins.



Class Diagram of the package `org.typo3.forge.dev3.ui.wizards.newAspectWizard`

Chapter 4. User Interface

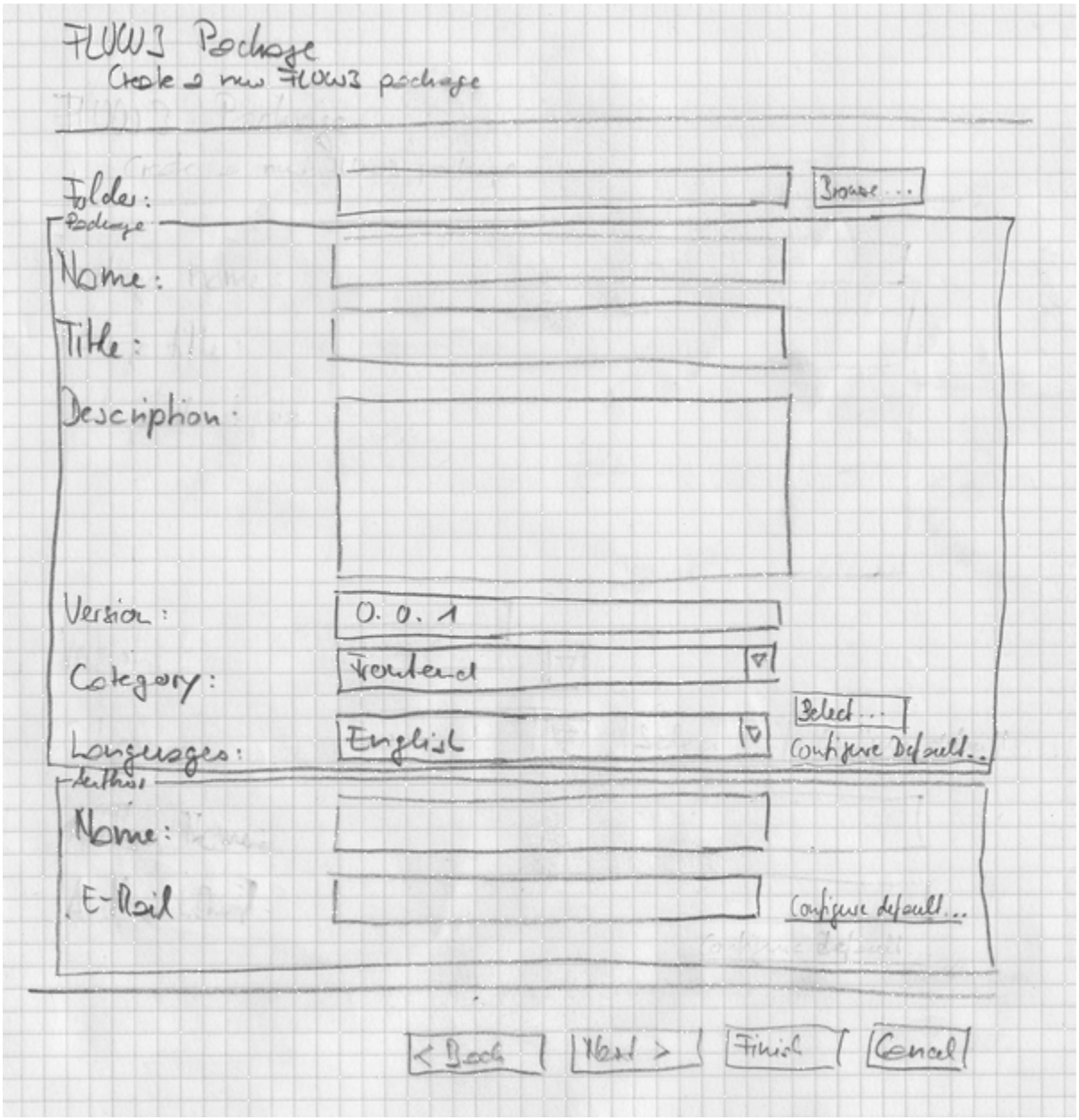
4.1. New Package Wizard

The “New Package Wizard” is started by selected File > New > Other > DEV3 > FLOW3 Package.

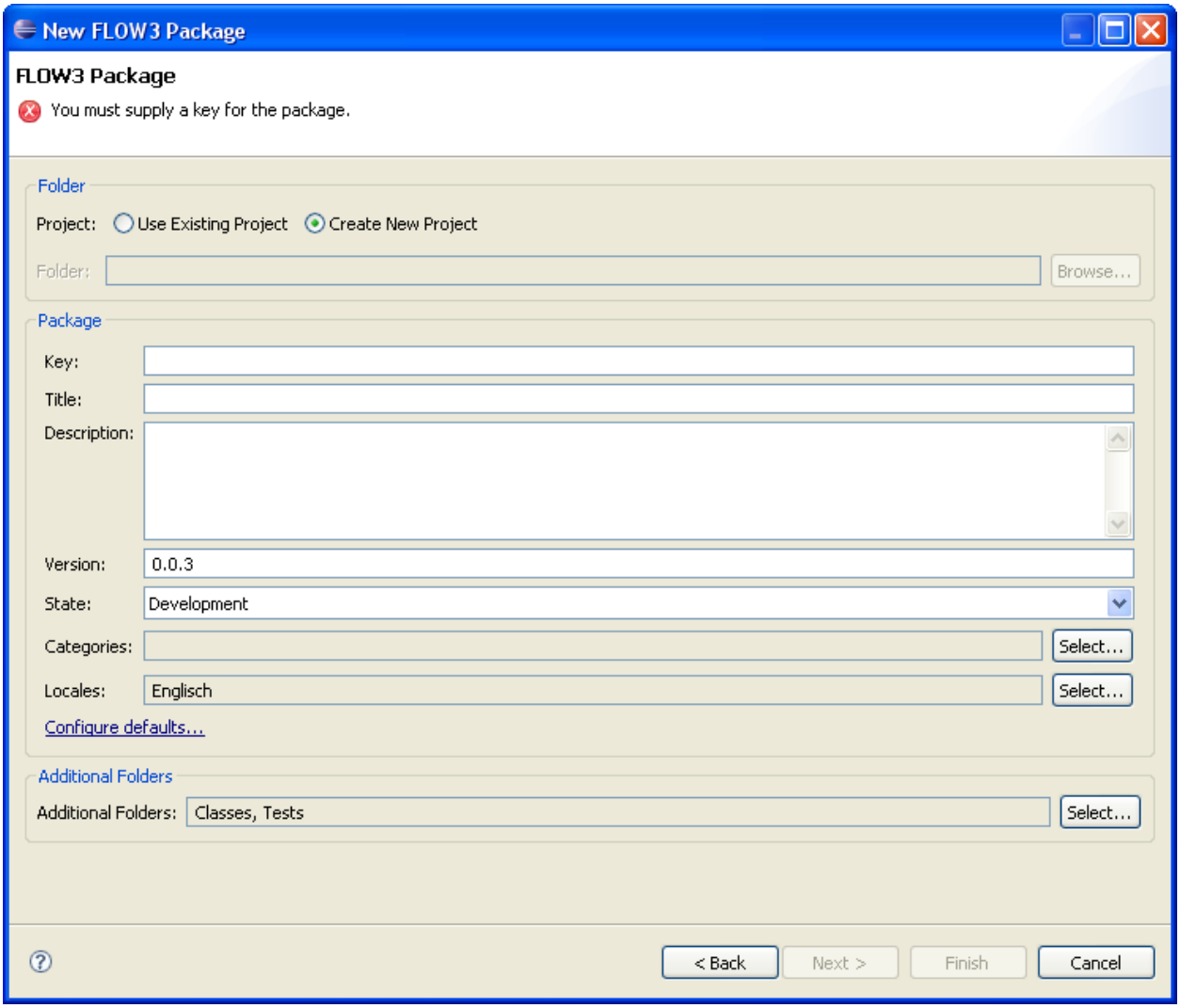
4.1.1. General Page

The first page of the “New Package Wizard” collects general information about the package.

- Project: The user can choose to use an existing project for this package (e.g. a PHP-project created by PDT) or to create a new (generic) project by DEV3.
- Folder: If the user wants to use an existing project, the path to this folder must be entered here or can be selected by clicking the “Browse...” button. Mandatory.
- Key: The unique key of the package. Mandatory. Should start with an uppercase Letter and may only consist of numbers an letters, according to the FLOW3 Coding Guidelines.
- Title: A one-line title of the package. Optional.
- Description: Free text description of the package. Optional.
- Version: Version number of the package. Should be in the format “x.x.x”. Optional.
- State: State of the package (Development, Alpha, Beta, Release Candidate, Final, Obsolote). Mandatory.
- Categories: An unlimited list of categories this package belongs to. The categories can be defined in the preference page. Optional.
- Locales: An unlimited list of categories this package is translated to. The default values can be defined in the preference page. Optional.
- Configure Defaults: Link to the property page where the categories and default locales can be changed.
- Additional Folders: A list of additional folders to be created. Optional.



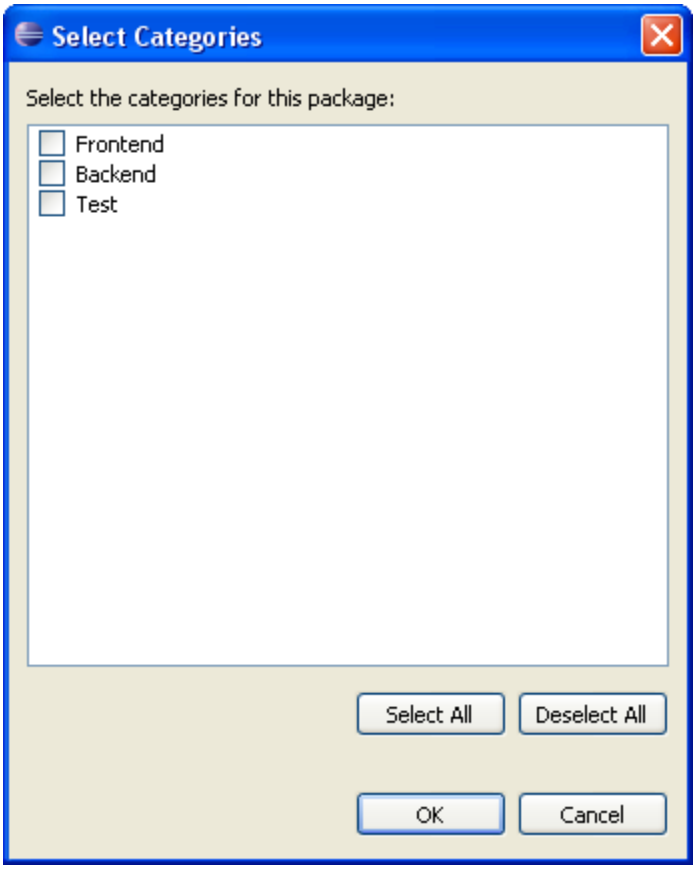
Draft of the “General Page” of the “New Package Wizard”



Implementation of the "General Page" of the "New Package Wizard"

Select Categories

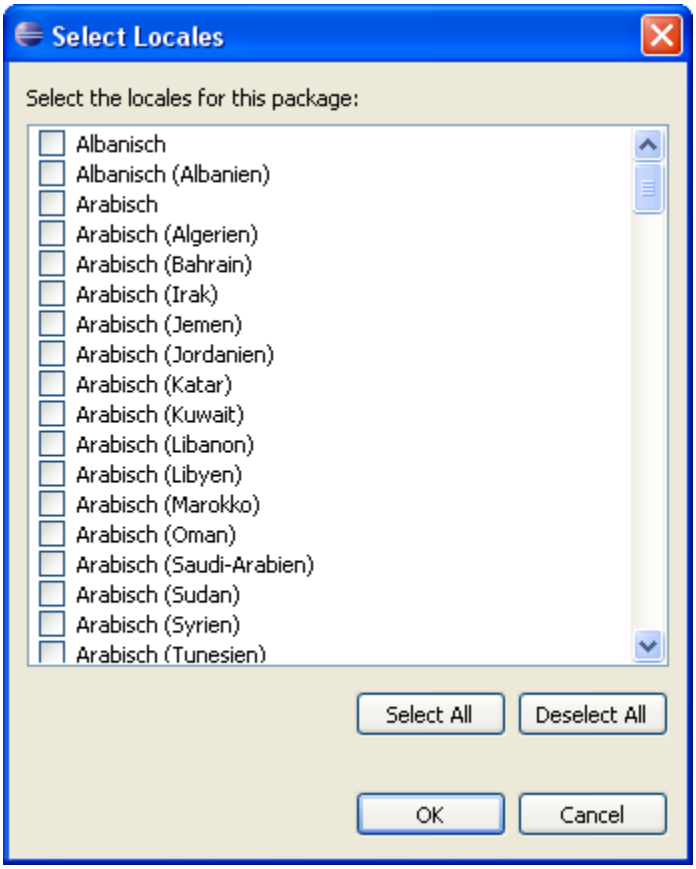
When the user clicks on the "Select..." button for Categories, the "Select Categories" dialog is opened. This dialog shows all categories defined in the preference page. The buttons "Select All" and "Deselect All" are convenience functions for fast selection/deselection. If the user clicks on Cancel, the selection is discarded. If the user selects "OK", the selected categories are transferred to the wizard.



Implementation of the "Select Categories" dialog "New Package Wizard"

Select Locales

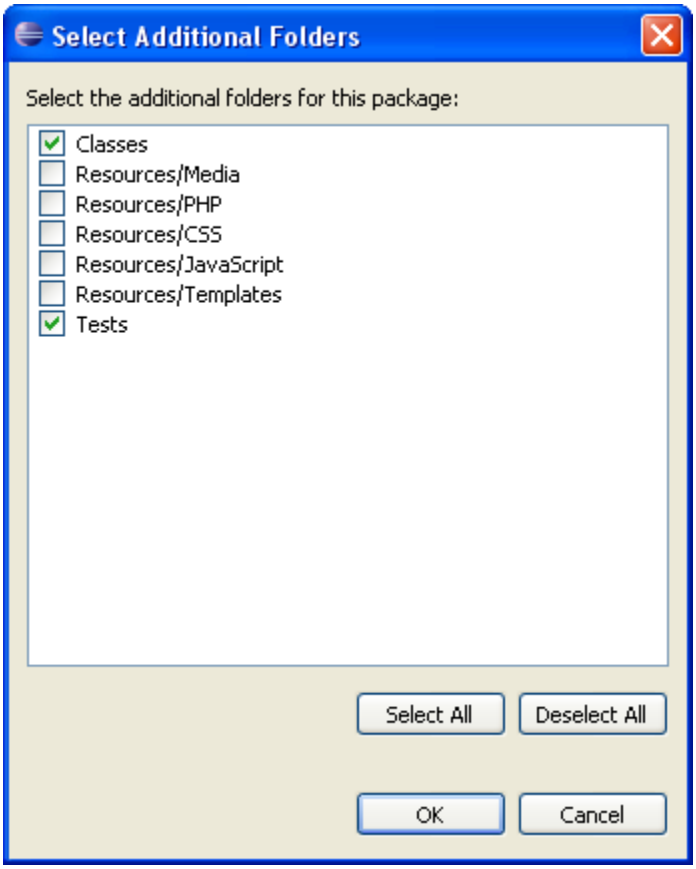
When the user clicks on the "Select..." button for Locales, the "Select Locales" dialog is opened. This dialog shows all available locales. The preferred locales as defined in the preference page are selected. The buttons "Select All" and "Deselect All" are convenience functions for fast selection/deselection. If the user clicks on "Cancel", the selection is discarded. If the user selects "OK", the selected locales are transferred to the wizard.



Implementation of the "Select Locales" dialog "New Package Wizard"

Select Additional Folders

When the user clicks on the "Select..." button for Additional Folders, the "Select Additional Folders" dialog is opened. This dialog shows all additional folders which can be created upon the creation of the package. By default, the folders "Classes" and "Tests" are selected. The buttons "Select All" and "Deselect All" are convenience functions for fast selection/deselection. If the user clicks on "Cancel", the selection is discarded. If the user selects "OK", the selected additional folders are transferred to the wizard.

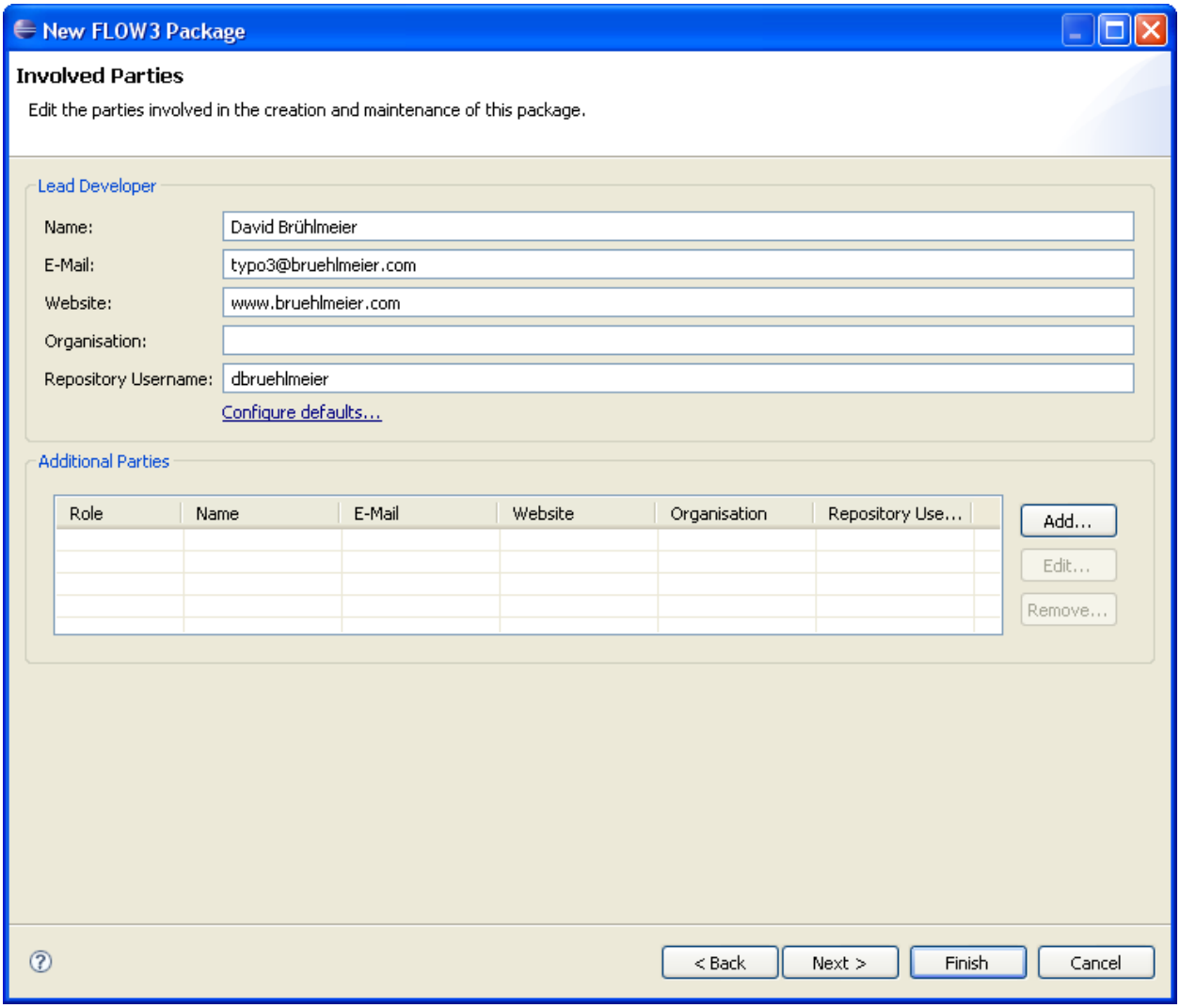


Implementation of the "Select Locales" dialog "New Package Wizard"

4.1.2. Involved Parties Page

The second page of the "New Package Wizard" allows the user to specify the parties which are involved in the development of this package.

- Name: Full name of the lead devloper of this package. Optional.
- E-Mail: E-Mail address of the lead developer of this package. If entered, the e-mail address is checked for proper format. Optional.
- Website: URL of the lead developer's website. Optional.
- Organisation: Organisation which the lead developer belongs to. Optional.
- Repository Username: Username of the lead developer for the FLOW3 repository (`forge.typo3.org`). Optional.
- Additional Parties: Table displaying the additional parties involved in the development of this package. Optional.

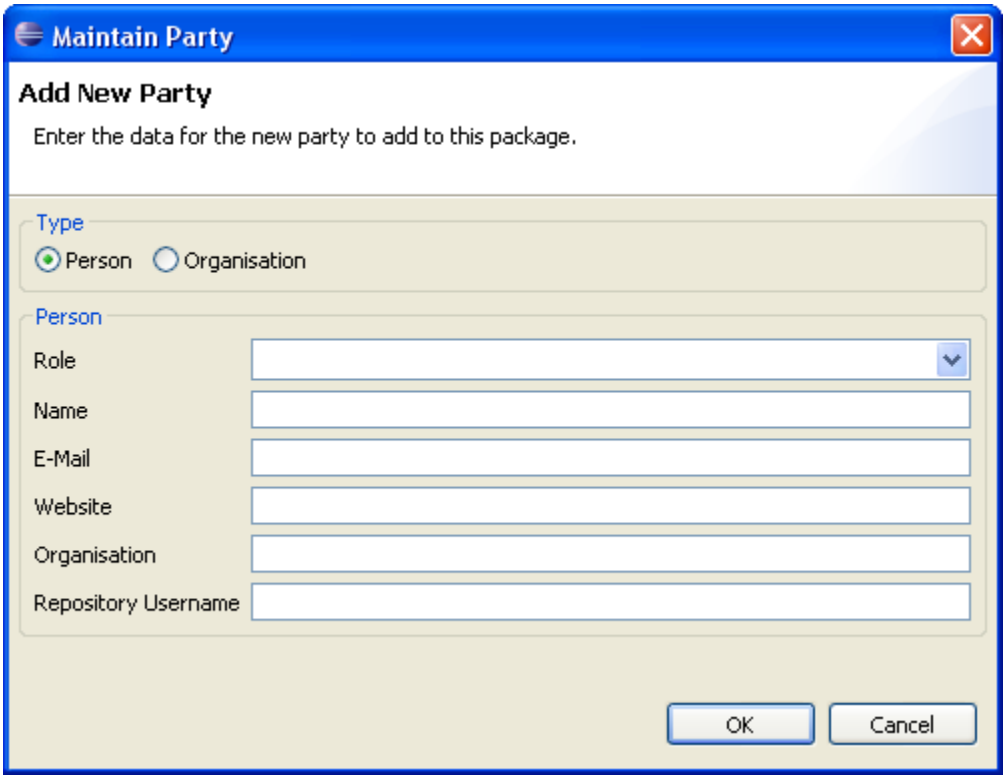


Implementation of the "Involved Parties Page" of the "New Package Wizard"

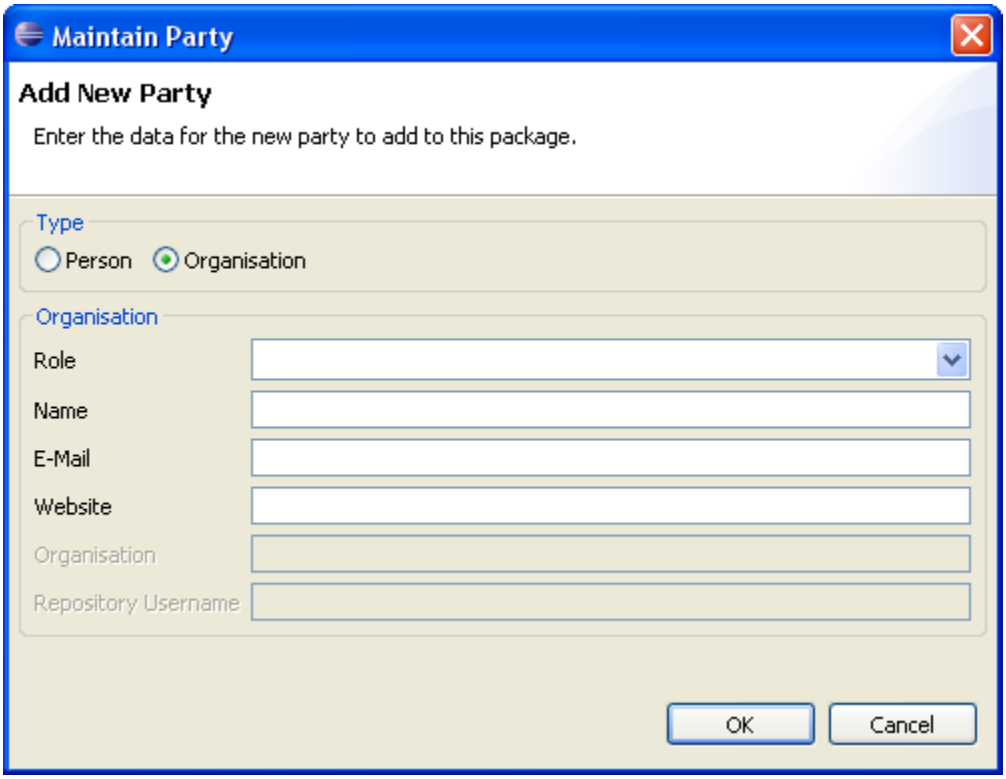
Add/Edit Party

When the user clicks on the "Add..." button, the "Maintain Party" dialog is opened.

- Type: Type of the party (Person or Organisation). When the user changes the type to "Organisation", the fields "Organisation" and "Repository Username" must be disabled. Mandatory.
- Role: Role of the party (Co-Developer, Lead Developer, Maintainer, Sponsor) in the development of this package. Optional.
- Name: Full name of the party. Optional.
- E-Mail: E-Mail address of the party. If entered, the e-mail address is checked for proper format. Optional.
- Website: URL of the party's website. Optional.
- Organisation: Organisation which the person belongs to. Only available for persons. Optional.
- Repository Username: Username of the person for the FLOW3 repository (`forge.typo3.org`). Only available for persons. Optional.



Implementation of the “Maintain Person” dialog “New Package Wizard”

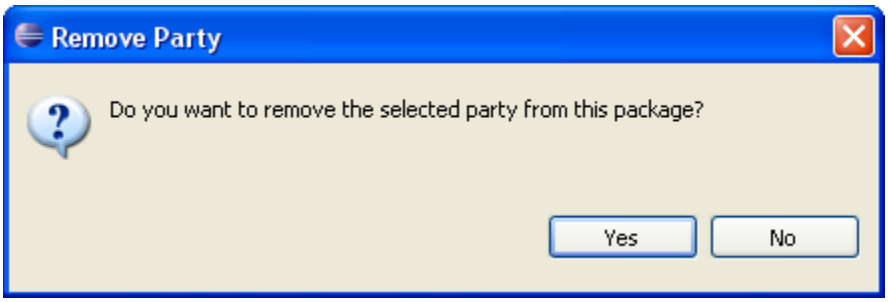


Implementation of the “Maintain Organisation” dialog “New Package Wizard”

When the user clicks on the “Edit...” button, the “Maintain Party” dialog is opened again, filled with the data of the selected party. If no party is selected, the “Edit...” button must be disabled.

Remove Party

When the user clicks on the “Remove...” button, the system asks for confirmation. If no party is selected, the “Remove...” button must be disabled.

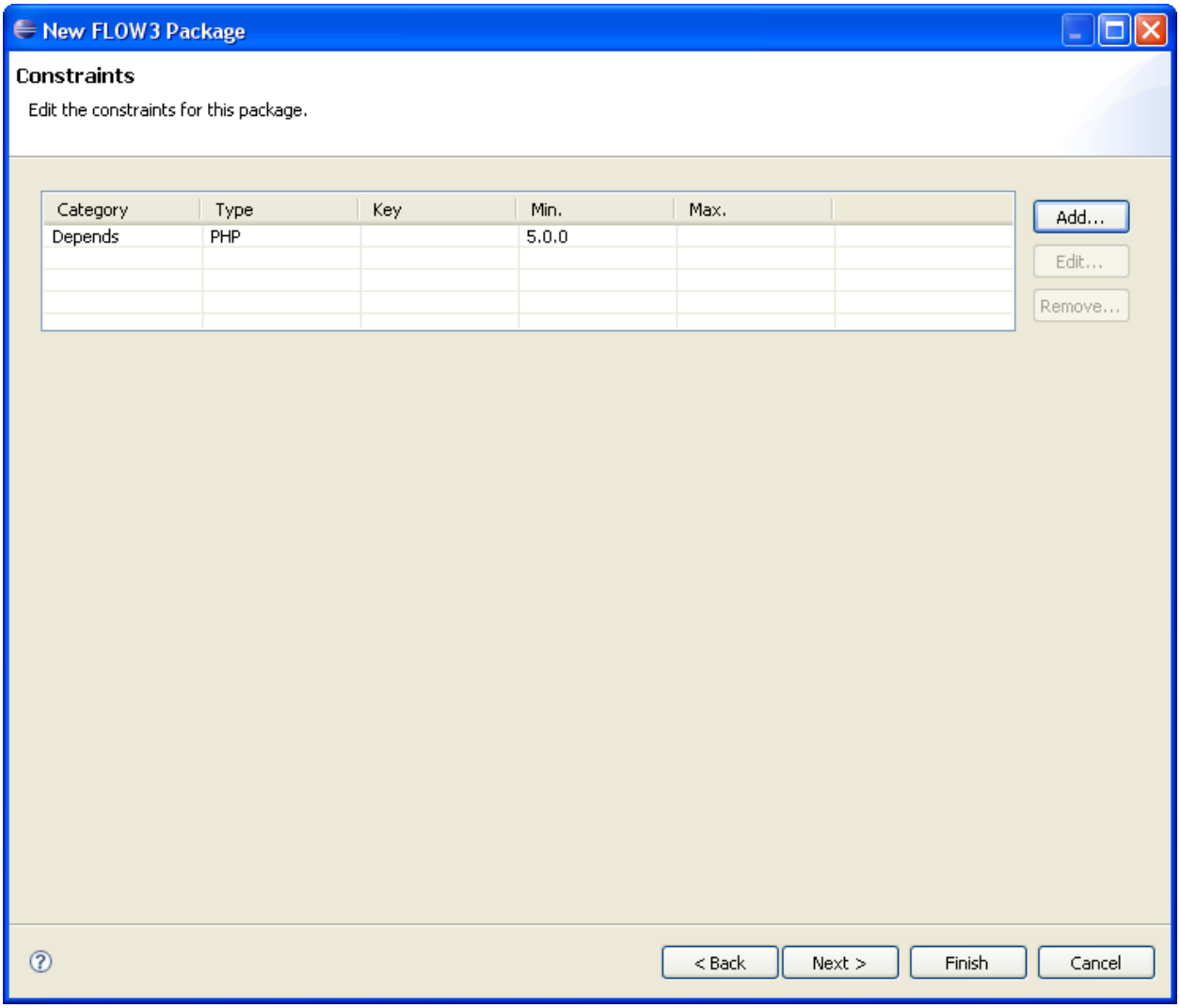


Implementation of the “Remove Party” dialog “New Package Wizard”

4.1.3. Constraints Page

The third page of the “New Package Wizard” allows the user to specify constraints for this package.

- Category: Category of the constraint (Depends, Suggests or Conflicts).
- Type: For system constraints, the type is displayed (PHP, PHP Extension, Operating System, Memory, PEAR). For package constraints, the constant “Package” is displayed.
- Key: The content for this field depends on the type of the category.
  - Package constraint: The package key is displayed.
  - System constraint type PHP: Nothing is displayed.
  - System constraint type PHP Extension: The name of the PHP Extension is displayed.
  - System constraint type Operating System: The name of the Operating System is displayed.
  - System constraint type Memory: Nothing is displayed.
  - System constraint type PEAR: The name of the PEAR Extension is displayed.
- Min.: The minimum version or requirement (e.g. for memory) is displayed.
- Max.: The maximum version is displayed.

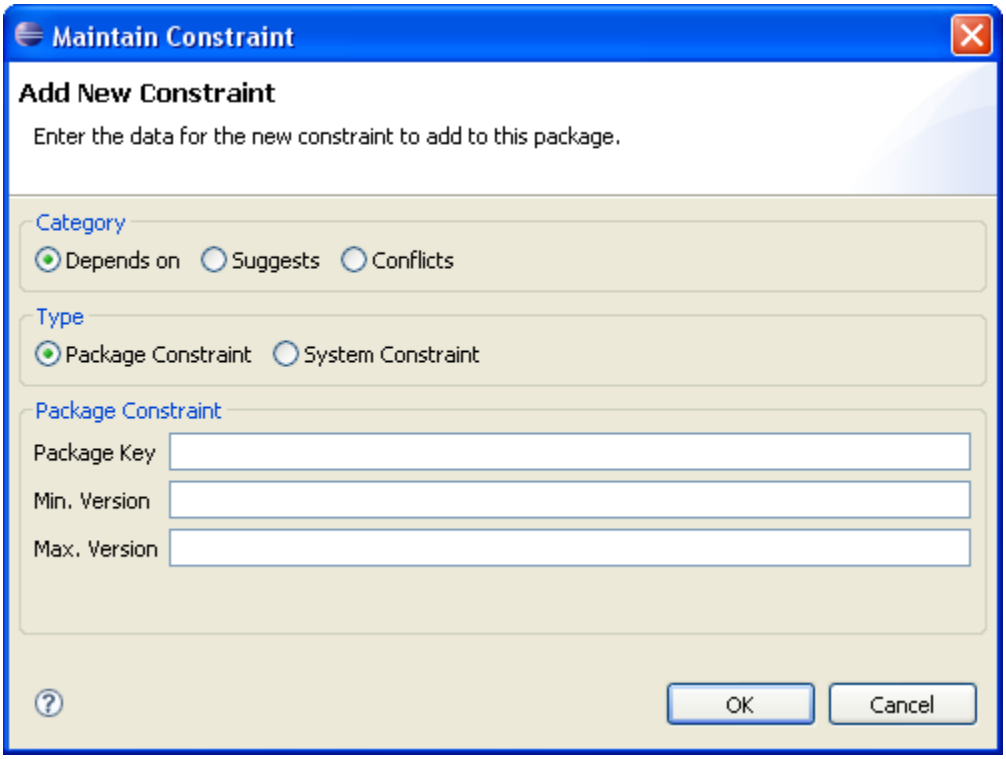


Implementation of the "Constraints Page" of the "New Package Wizard"

Add/Edit Constraint

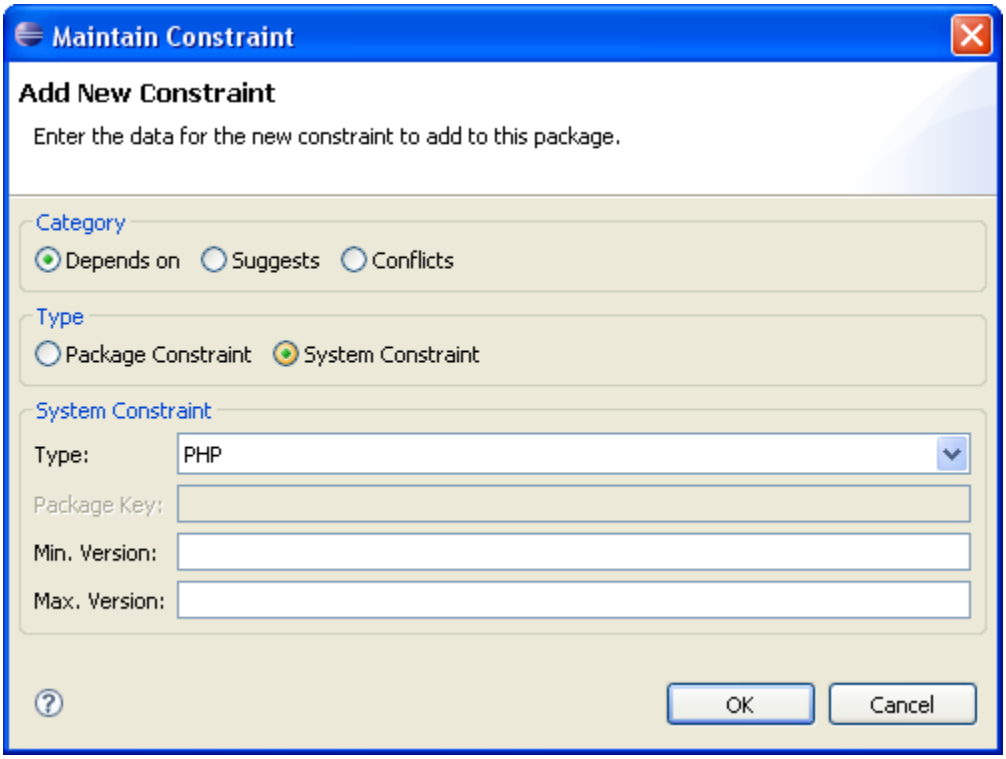
When the user clicks on the "Add..." button, the "Maintain Constraint" dialog is opened.

- Category: Category of the constraint (Depends on, Suggests or Conflicts). Mandatory.
- Type: Type of the constraint (Package Constraint or System Constraint). When the user selects "Package Constraint", the fields "Package Key", "Min. Version" and "Max. Version" must be displayed. When the user selects "System Constraint", the fields "Type", "Package Key", "Min. Version" and "Max. Version" must be displayed.
- Package Key: Key of the package for this package constraint. Mandatory for package constraints.
- Min. Version: Minimum version the package must have to fulfull the constraint. Optional.
- Max. Version: Maximum version the package must have to fulfill the constraint. Optional.



Implementation of the "Maintain Package Constraint" dialog "New Package Wizard"

- Type: Type of the system constraint (PHP, PHP Extension, Operating System, Memory, PEAR). Mandatory.
- Key:
  - For PHP: Disabled.
  - For PHP Extension: Name of the PHP Extension.
  - For Operating System: Name of the Operating System.
  - For Memory: Disabled.
  - For PEAR: Name of the PEAR Extension.
- Min. Version: Minimum version to fulfull the constraint. For memory: Minimal amount of memory to be available. Optional.
- Max. Version: Maximum version to fulfill the constraint. Optional.

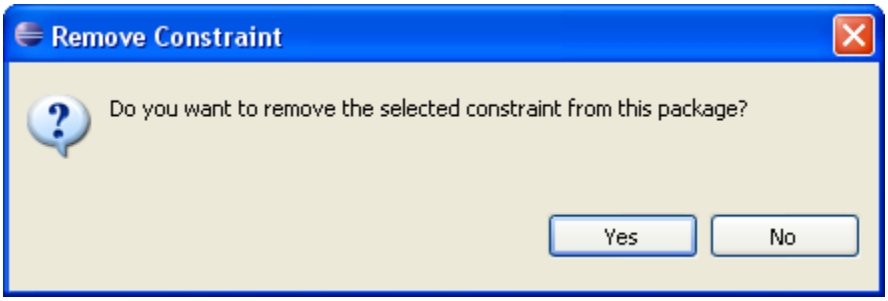


Implementation of the "Maintain System Constraint" dialog "New Package Wizard"

When the user clicks on the "Edit..." button, the "Maintain Constraint" dialog is opened again, filled with the data of the selected constraint. If no constraint is selected, the "Edit..." button must be disabled.

Remove Constraint

When the user clicks on the "Remove..." button, the system asks for confirmation. If no constraint is selected, the "Remove..." button must be disabled.

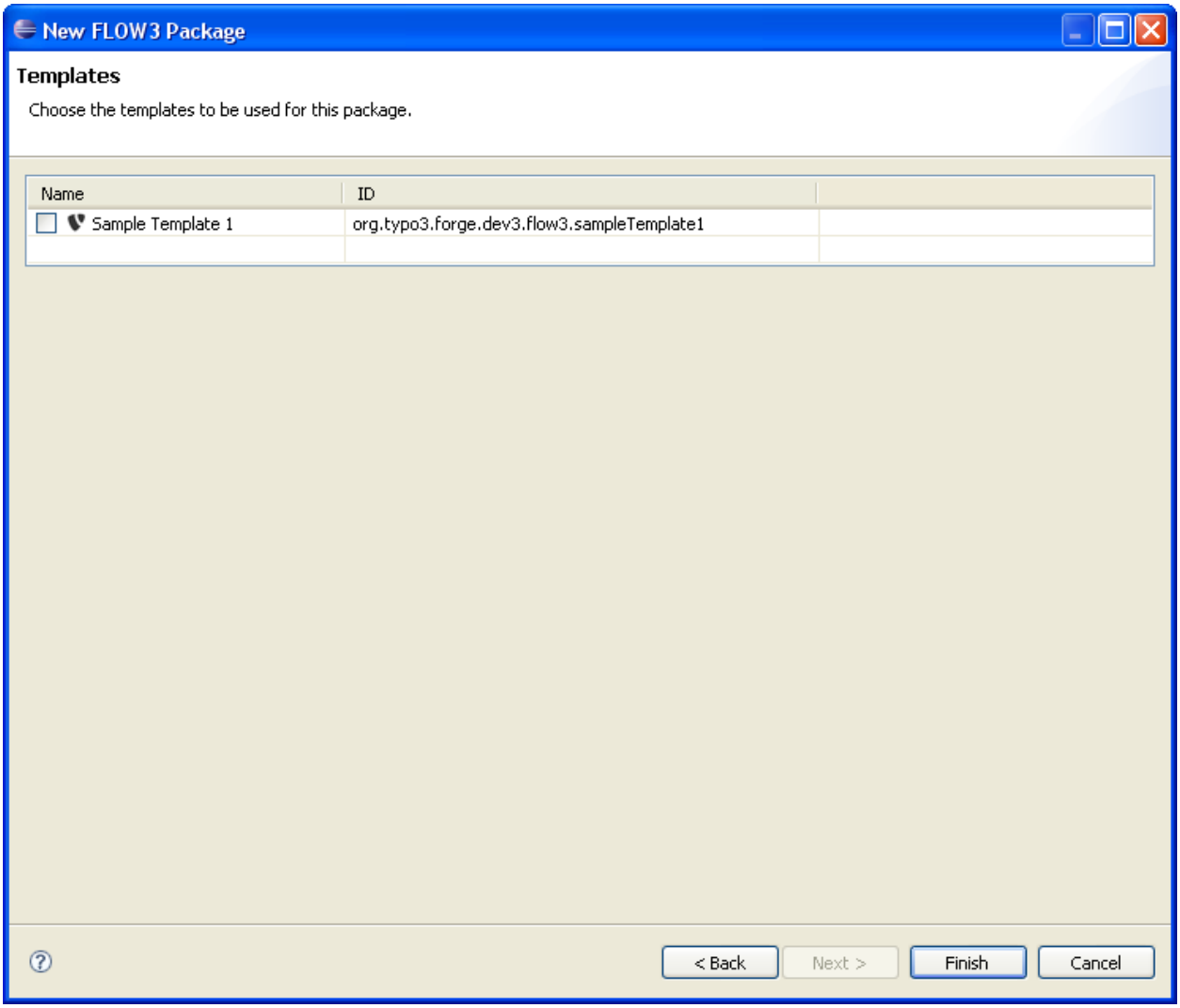


Implementation of the "Delete Constraint" dialog "New Package Wizard"

4.1.4. Templates Page

The fourth and final page of the "New Package Wizard" allows the user to select an unlimited amount of templates to be executed upon creation of this package.

- Name: Displays the icon and the name of the template.
- ID: Displays the ID under which the template was registered (usually a third-party extension).



Implementation of the "Templates Page" of the "New Package Wizard"

4.2. New Aspect Wizard

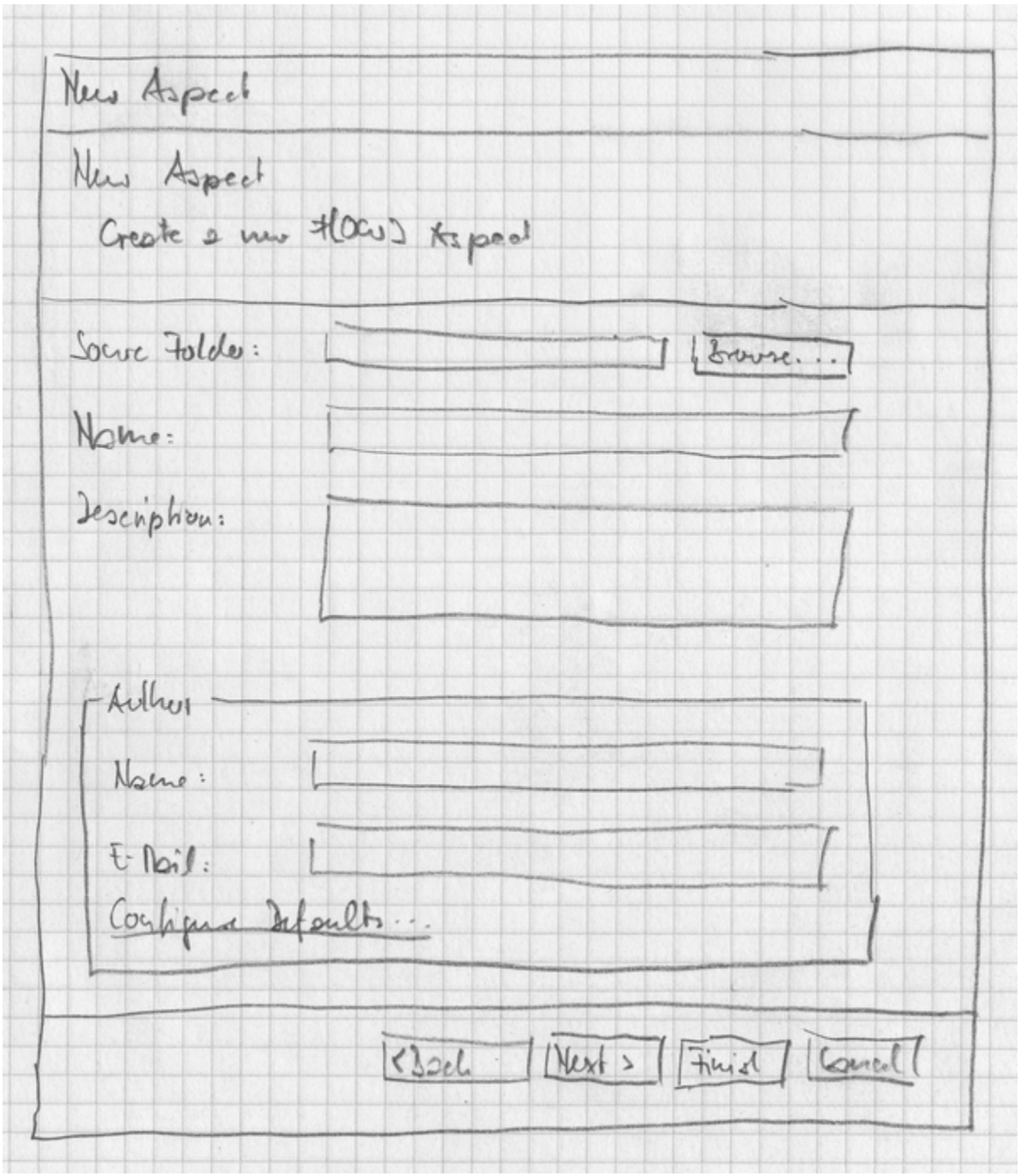
The "New Aspect Wizard" is started by selected File > New > Other > DEV3 > FLOW3 Aspect.

4.2.1. General Page

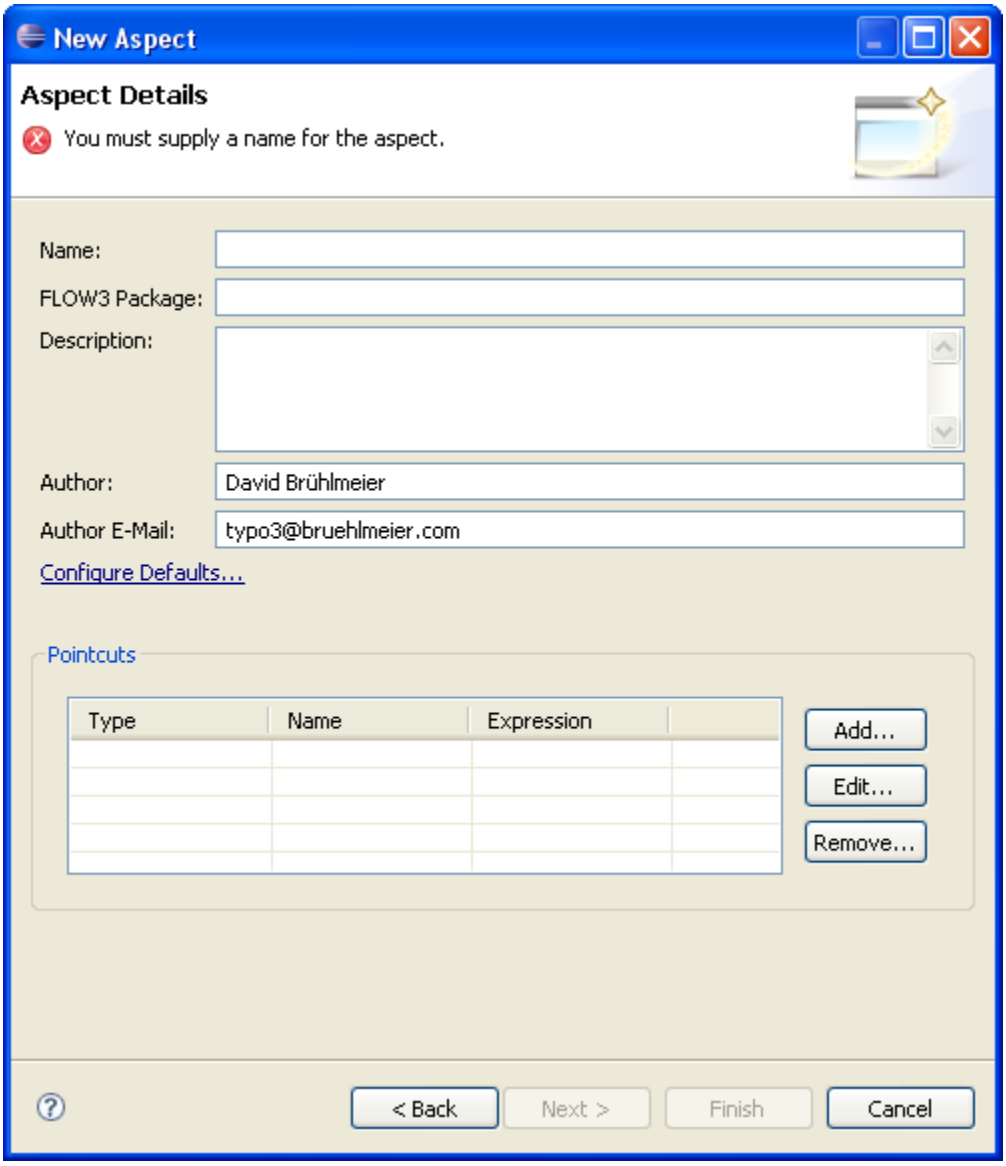
The first page of the "New Aspect Wizard" collects general information about the aspect.

- Name: The name of the aspect. Mandatory. Shall start with an uppercase letter and must contain only of letters and numbers, according to the FLOW3 coding guidelines.
- FLOW3 Package: The key of the package this aspect belongs to. Mandatory. Filled by the system in case a Meta/Package.xml file can be found.
- Description: Free text description of the aspect. Optional.
- Author: Name of the author of the aspect. Optional.

- Author E-Mail: E-Mail address of the aspect's author. Optional. Checked for proper format of the address.
- Configure Defaults: Link to the property page where the default author and default e-mail address can be changed.
- Pointcuts: List of pointcuts for this aspect. Optional.



Draft of the General Page of the “New Aspect Wizard”

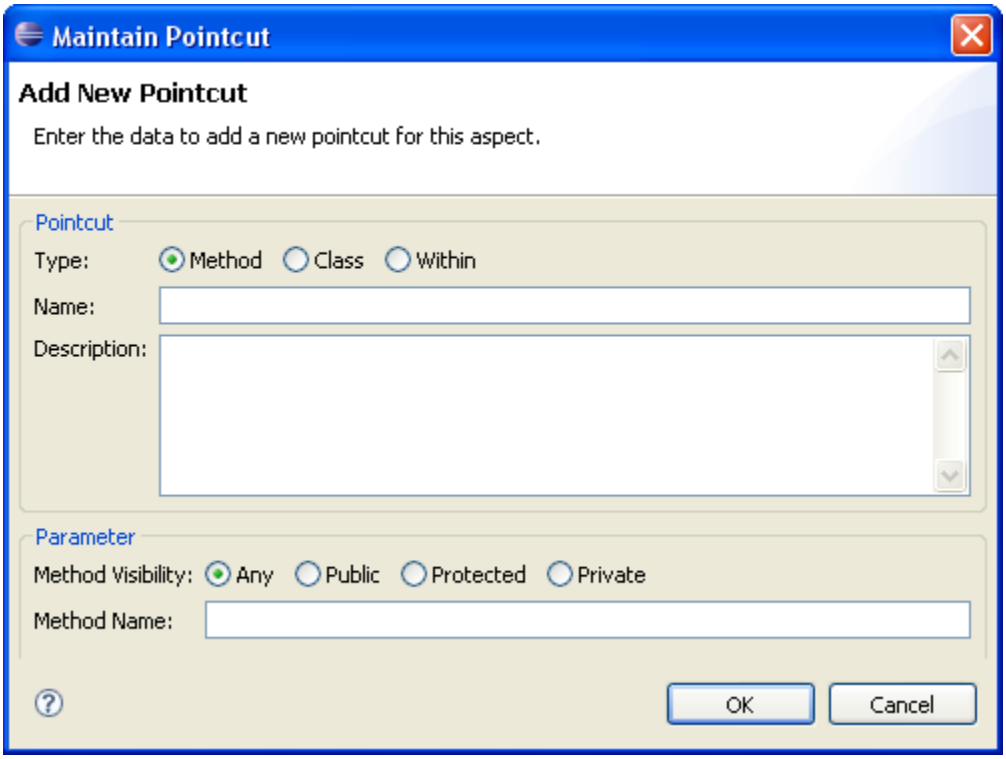


Implementation of the General Page of the "New Aspect Wizard"

Add/Edit Pointcut

When the user clicks on the "Add..." button, the "Maintain Pointcut" dialog is opened.

- Type: Type of the pointcut (Method, Class or Within). Mandatory. When the user changes the type to "Method", the "Method Visibiliy" radio buttons must be active, otherwise inactive.
- Name: Name of the pointcut. Optional.
- Description: Free text description of the pointcut. Optional.
- Method Visibility: Visibility of the pointcut (Any, Public, Protected, Private). Mandatory for "Method" pointcuts, not available for other pointcuts.
- Method name: Name of the method this pointcut refers to. Optional.

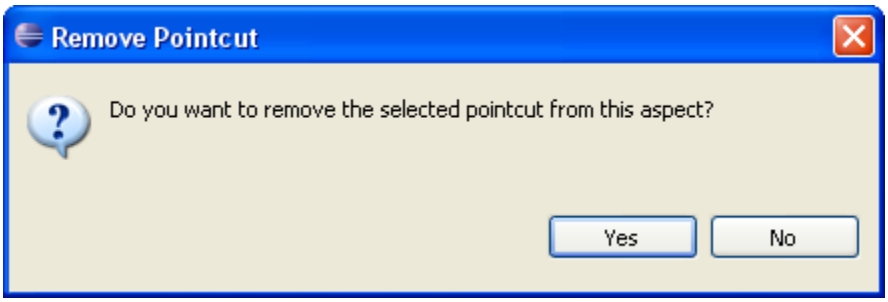


Implementation of the "Add Pointcuts" dialog "New Aspect Wizard"

When the user clicks on the "Edit..." button, the "Maintain Pointcut" dialog is opened again, filled with the data of the selected pointcut. If no pointcut is selected, the "Edit..." button must be disabled.

Remove Pointcut

When the user clicks on the "Remove..." button, the system asks for confirmation. If no pointcut is selected, the "Remove..." button must be disabled.

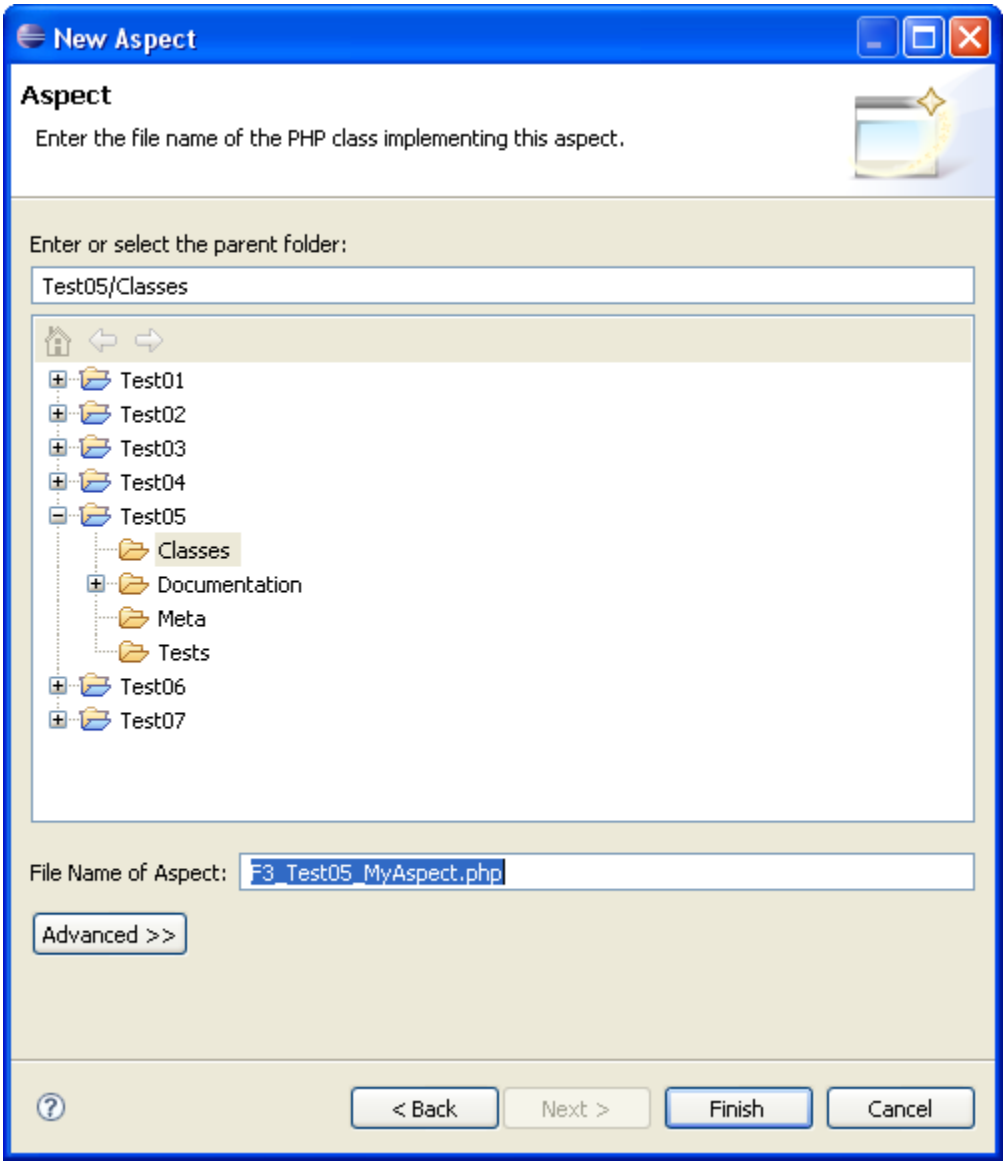


Implementation of the "Remove Pointcut" dialog "New Aspect Wizard"

4.2.2. File Page

The second page of the "New Aspect Wizard" is a standard "New Resource" page.

- Parent Folder: The user may enter the parent folder or select it from the tree viewer below. Mandatory.
- File Name of Aspect: The file name contains a suggestion from the system, based upon the package key and the name of the aspect, prefixed with the constant "F3", as requested by the FLOW3 coding guidelines, and suffixed by the constant ".php". Mandatory.



Implementation of the file page of the "New Aspect Wizard"

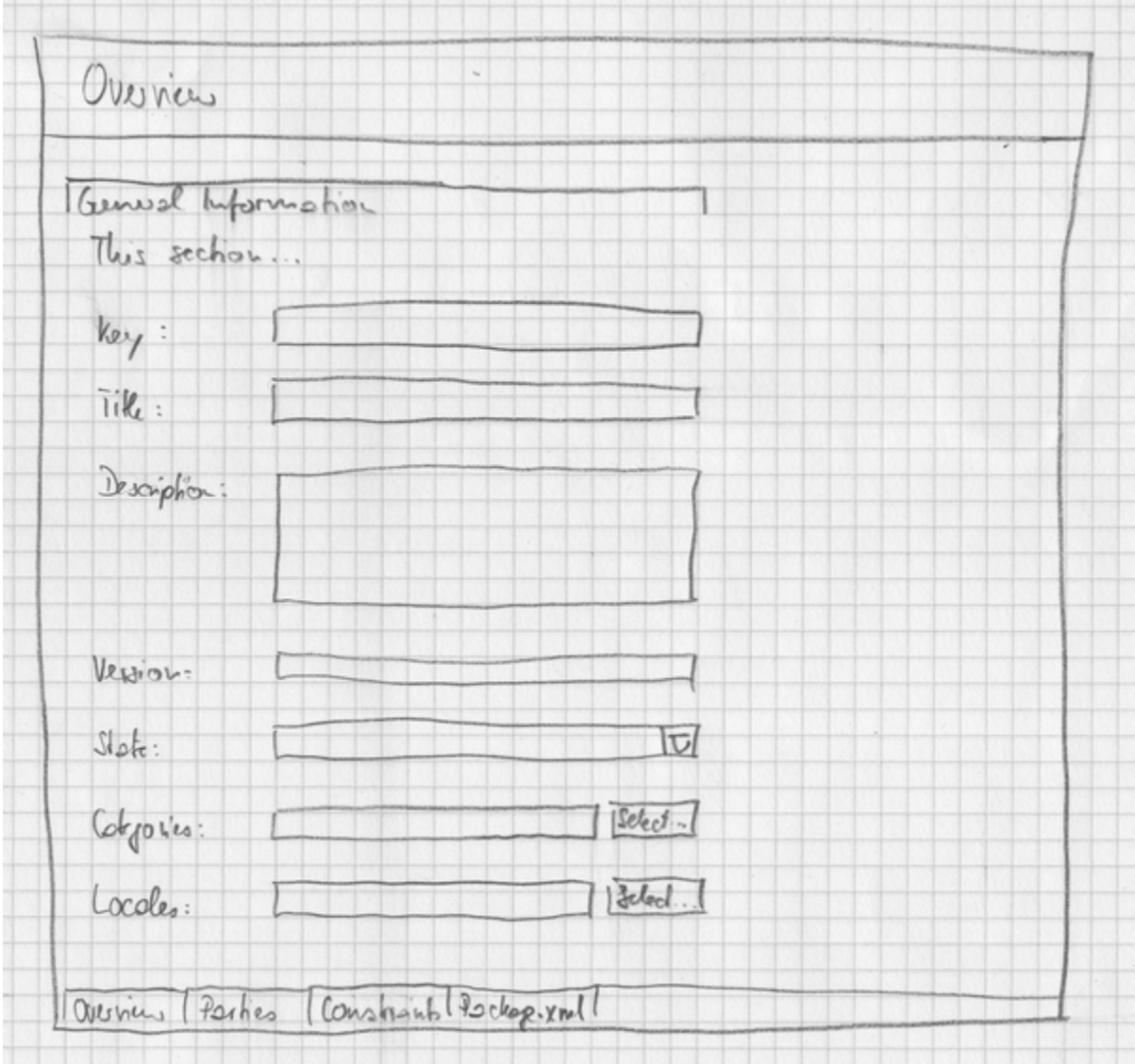
4.3. FLOW3 Package Editor

The "FLOW3 Package Editor" offers a user-friendly way to edit the central FLOW3 configuration file "Plugin.xml". It is started when double-clicking on a file called `Package.xml` or by right-clicking a file and then choosing Open with > FLOW3 Package Editor.

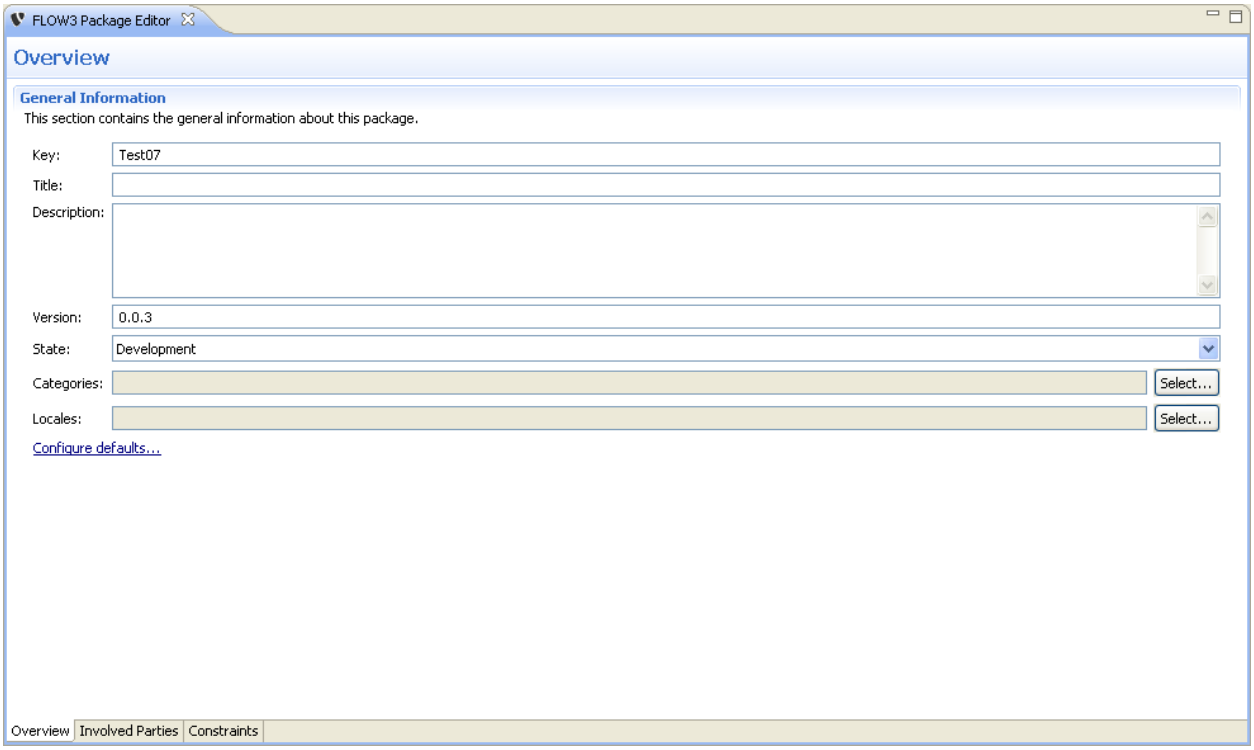
4.3.1. Overview Page

The overview page of the "FLOW3 Package Editor" displays general information about the package. The fields are the same as in the "General Page" of the "New Package Editor", with the exception of "Project" and "Folder". These fields make no sense for the editor, since they are determined by the existing location of the `Package.xml` file.

For a description of the fields, please refer to the New Package Wizard: General Page.



Draft of the “Overview Page” of the “FLOW3 Package Editor”

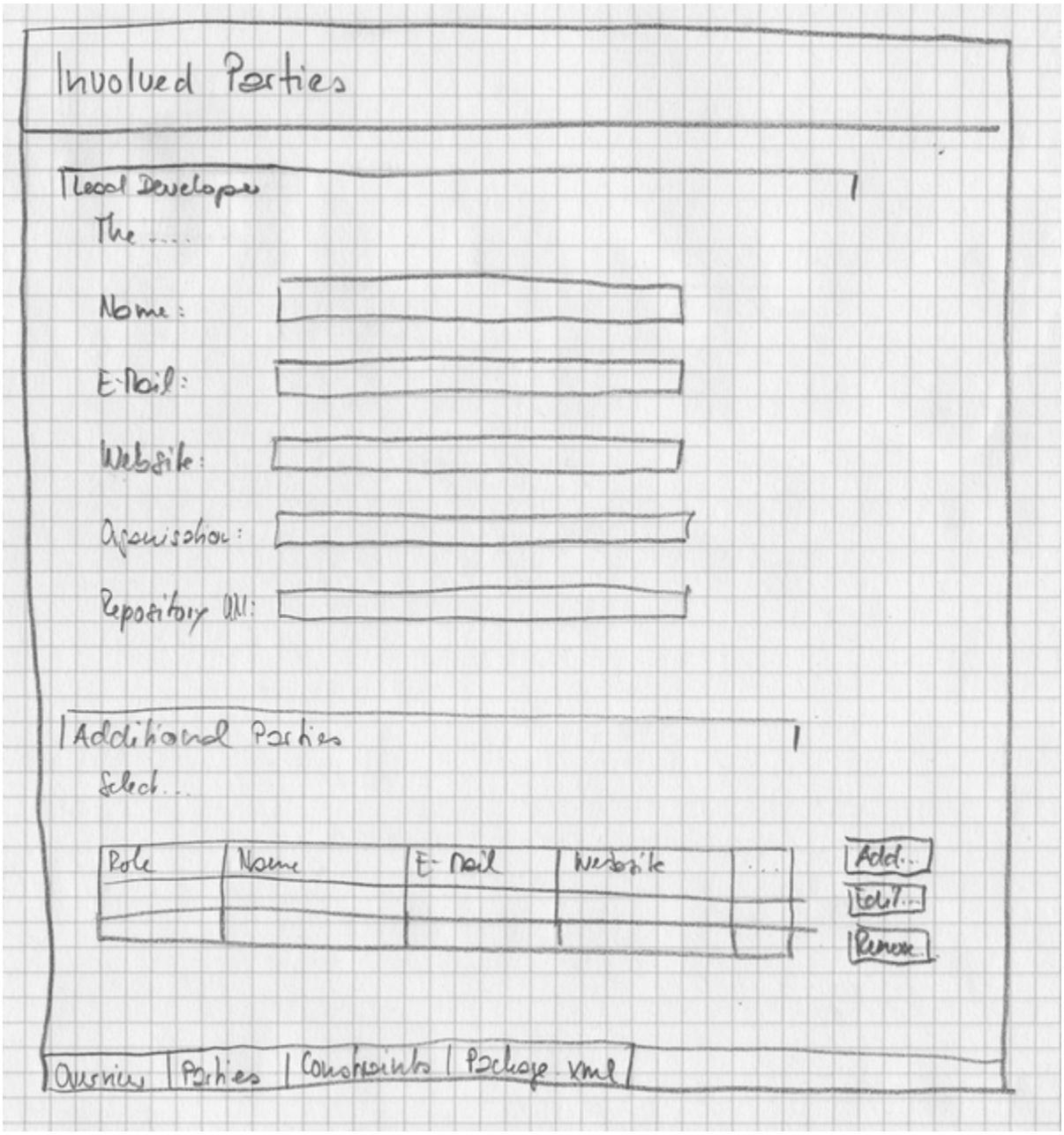


Implementation of the "Overview Page" of the "FLOW3 Package Editor"

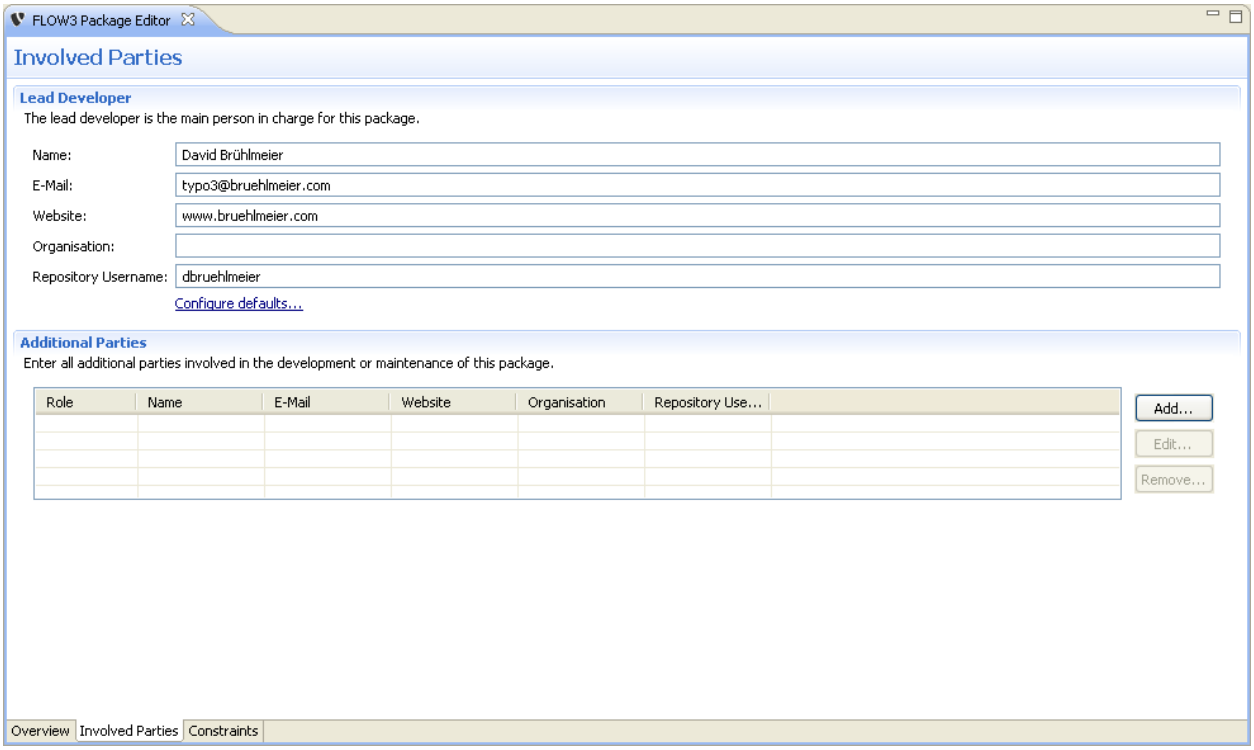
4.3.2. Involved Parties Page

The "Involved Parties Page" of the "FLOW3 Package Editor" displays the parties involved in the development of the package. The fields are the same as in the "Involved Parties Page" of the "New Package Editor".

For a description of the fields, please refer to the New Package Wizard: Involved Parties Page.



Draft of the “Involved Parties Page” of the “FLOW3 Package Editor”

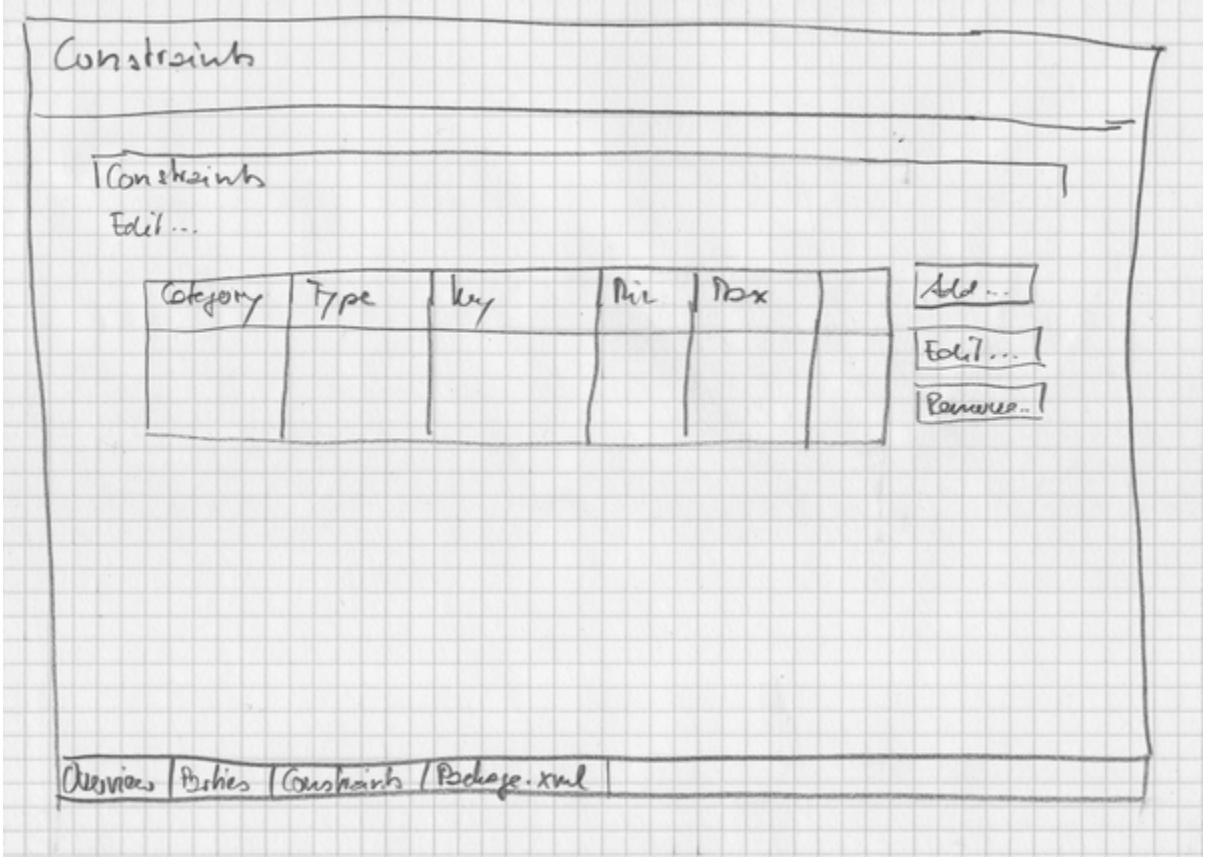


Implementation of the "Involved Parties Page" of the "FLOW3 Package Editor"

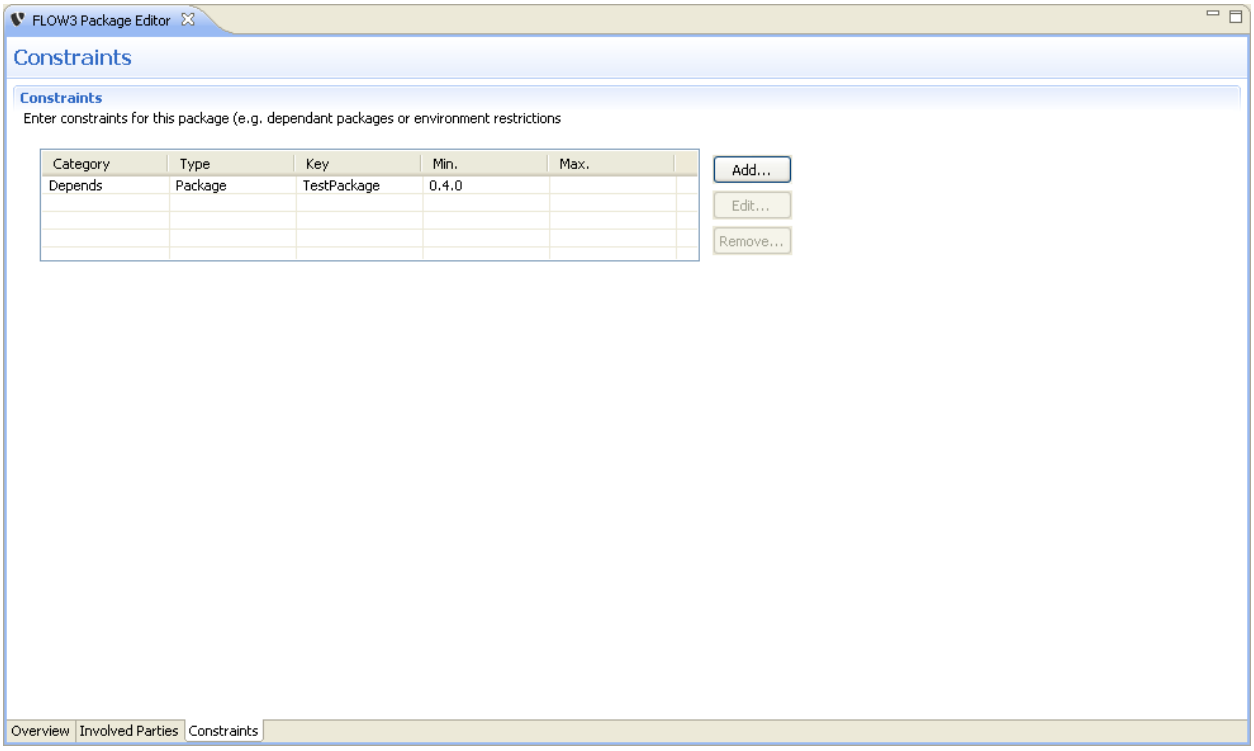
4.3.3. Constraints Page

The "Constraints Page" of the "FLOW3 Package Editor" displays the constraints for this package. The fields are the same as in the "Constraints Page" of the "New Package Editor".

For a description of the fields, please refer to the New Package Wizard: Constraints Page.



Draft of the “Constraints Page” of the “FLOW3 Package Editor”



Implementation of the “Constraints Page” of the “FLOW3 Package Editor”

4.4. TypoScript 2.0 Editor

The "TypoScript 2.0 Editor" offers basic features such as syntax highlighting and folding when editing TypoScript 2.0. It also offers a prototype implementation of problem marking and code completion as well as a basic outline view. The editor started when double-clicking on a file with the extension `.ts2` or by right-clicking a file and then choosing Open with > TypoScript 2.0 Editor.

4.4.1. Syntax Highlighting

The editor recognizes the following TypoScript elements and highlights them in different colors:

- Block comments: Brown (RGB 128/0/0)
- Single line comments: Brown-Green (RGB 128/128/0)
- Keywords: Blue (RGB 0/0/128)
- Strings: Green (RGB 0/128/0)
- Other: Black (RGB 0/0/0)

```
MyTypoScript.ts2
1  namespace: party = F3_Party_TypoScript
2
3  include: source = "FILE: mytyposcript.txt"
4
5  namespace: forms = F3_CoolForms_TypoScript
6  namespace:cms = F3_TYPO3_Library
7
8  /*
9  * A block comment.
10 */
11 namespace:test=F3_Test_Test
12 include:source = "typoScript.ts2"
13
14 firstObject = Text
15 firstObject {
16     value = "Go outside. The graphics are AMAZING!"
17 }
18
19 # A single line comment
20 secondObject {
21     subObject = Text
22     subObject.value = "27°C and a blue sky."
23 }
24
25 // Another single line comment
26 thirdObject {
27     subObject {
28         subSubObject {
29             someMessage = Text
30             someMessage.value = "Fully or hard tail?"
31         }
32     }
33 }
```

Syntax Highlighting of the "TypoScript 2.0 Editor"

4.4.2. Folding

The editor can fold block comments (starting with `/*` and ending with `*/`) and confinements (starting with `{` and ending with `}`), including nested confinements.

```
25 // Another single line comment
26 thirdObject {
33 }
```

Folding of the “TypoScript 2.0 Editor”

When hovering over a collapsed region, the invisble part is displayed as hover info.

```
25 // Another single line comment
26 {
33 }    subObject {
            subSubObject {
                someMessage = Text
                someMessage.value = "Fully or hard tail?"
            }
        }
```

Hovering folded regions of the “TypoScript 2.0 Editor”

4.4.3. Problem Marker

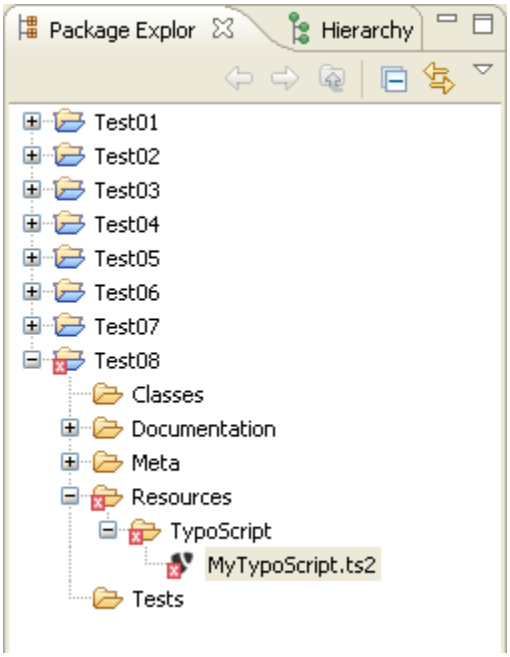
The TypoScript entered in the editor is parsed in the background. If an error is detected, it is displayed as a problem marker in the left margin.

Parsing currently stops at the first error detected.

```
MyTypoScript.ts2
1  namespace: party = F3_Party_TypoScript
2
3  include: source = "FILE: mytyposcript.txt"
4
5  namespace: forms = F3_CoolForms_TypoScript
6  namespace:cms = F3_TYPO3_Library
7
8  /*
9  * A block comment.
10 */
11 namespace:test=F3_Test_Test
12 The include type must be 'source'.oScript.ts2"
13
```

Problem marker of the “TypoScript 2.0 Editor”

Additionally, the TypoScript file with the parsing error is marked with a file marker and can therefore be easily recognized in the navigator view.



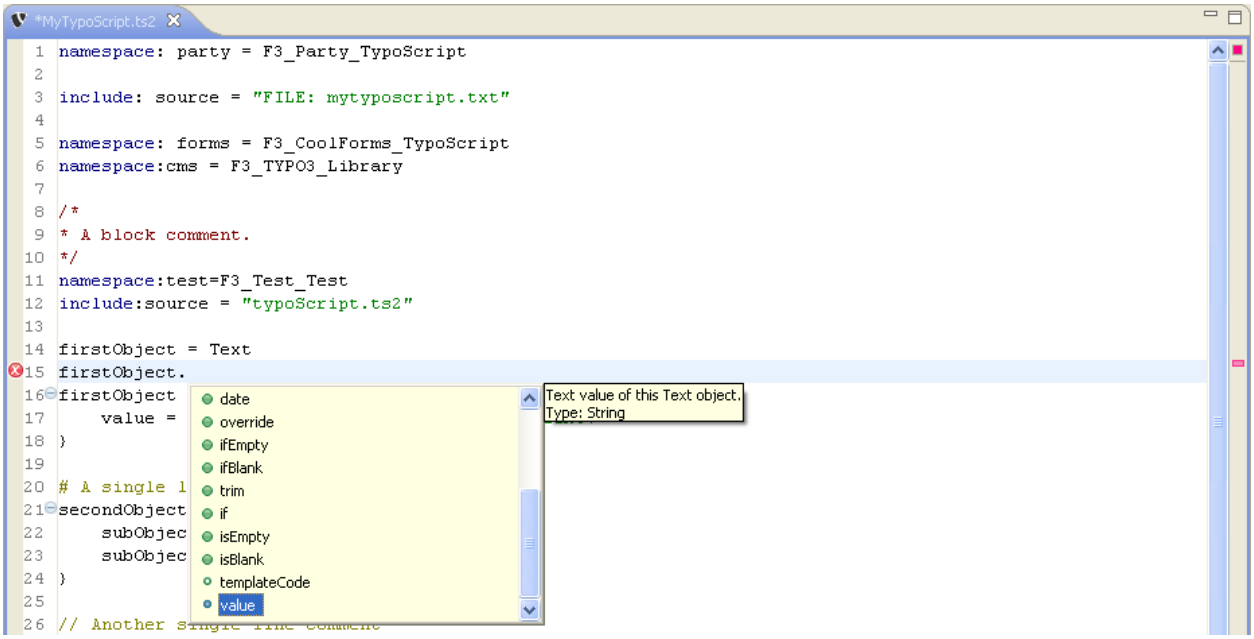
Navigator view with erroneous TypoScript file

4.4.4. Code Completion

The editor offers a prototype implementation of code completion. Currently, code completion is activated unconditionally whenever the user enters a dot. The proposals displayed are not filtered by the surrounding context. There are two types of proposals displayed:

- Properties: With a small green circle icon.
- Processors: With a large green dot icon.

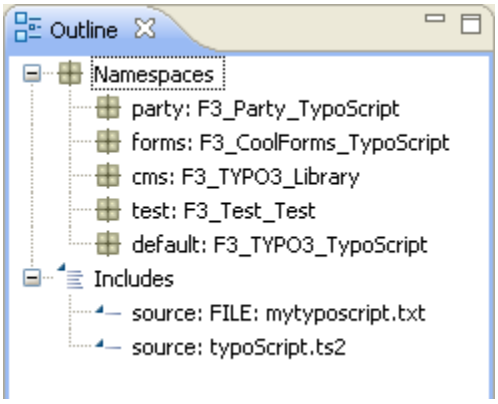
When hovering over a code completion proposal, an addition hover info is displayed with information about the property (free text and expected type) or about the processor (free text and description and type of the parameters, if any).



Code Completion of the "TypoScript 2.0 Editor"

4.4.5. Outline view

The outline view shows the list of namespace declarations and a list of include declarations. The "default" namespace is always displayed. If it has not been explicitly defined, the build-in default namespace of FLOW3 is displayed.



Outline view of the "TypoScript 2.0 Editor"

Chapter 5. Tests

5.1. Functional Tests

The functional tests are based on the use-cases as published in the requirement specification (revision 004 of 2008-04-02). Since all functional requirements are assigned to at least one use-case, these tests cover all functional requirements.

5.1.1. UC001: Create Package

Basic Path

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: Test01
- Title: Test 01
- Description: First Test
- Button: Finish

Expected Result

- Project with name "Test01" created
- Folders `Classes`, `Documentation`, `Documentation/en`, `Meta` and `Tests` created
- DocBook templates in folder `Documentation/en` with the project key and description created
- `Package.xml` with the key, the title and the description created

Actual Result

✓ Passed

Execution 2008-07-23 09:39 / David Brühlmeier

Basic Path With Additional Folders

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: Test02
- Title: Test 02
- Description: Second Test
- Additional Folders: Button Select > Check `Resources/Media`
- Button: Finish

Expected Result

- Project with name "Test02" created

- Folders `Classes`, `Documentation`, `Documentation/en`, `Meta`, `Resources/Media` and `Tests` created
- DocBook templates in folder `Documentation/en` with the project key and description created
- `Package.xml` with the key, the title and the description created

Actual Result

Passed

Execution 2008-07-23 09:46 / David Brühlmeier

Basic Path With Additional Locales

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: Test03
- Title: Test 03
- Description: Third Test
- Locales: Button Select > Check "Deutsch (Schweiz)"
- Button: Finish

Expected Result

- Project with name "Test03" created
- Folders `Classes`, `Documentation`, `Documentation/en`, `Documentation/de_CH`, `Meta` and `Tests` created
- DocBook templates in folder `Documentation/en` and `Documentation/de_CH` with the project key and description created
- `Package.xml` with the key, the title and the description created

Actual Result

Passed

Execution 2008-07-23 09:49 / David Brühlmeier

Basic Path With Additional Parties

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: Test04
- Title: Test 04
- Description: Fourth Test

- Button: Next
- Additional Parties: Button Add
  - Type: Person
  - Role: Co-Developer
  - Name: Mani Matter
  - Button: OK
- Button: Finish

Expected Result

- Project with name "Test04" created
- Folders `Classes`, `Documentation`, `Documentation/en`, `Meta` and `Tests` created
- DocBook templates in folder `Documentation/en` with the project key and description created
- `Package.xml` with the key, the title and the description created. The file also contains the person "Mani Matter" in the role "Co-Developer".

Actual Result

Passed

Execution

2008-07-23 09:53 / David Brühlmeier

Basic Path With Constraints

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: Test05
- Title: Test 05
- Description: Fifth Test
- Button: Next
- Button: Next
- Button: Add
  - Category: Conflicts
  - Type: System Constraint
  - System Constraint: Operating System
  - Operating System: Win32
  - Button: OK

- Button: Finish

Expected Result

- Project with name “Test05” created
- Folders `Classes`, `Documentation`, `Documentation/en`, `Meta` and `Tests` created
- DocBook templates in folder `Documentation/en` with the project key and description created
- `Package.xml` with the key, the title and the description created. The file also contains the conflicting constraint.

Actual Result

Passed

Execution 2008-07-23 09:58 / David Brühlmeier

Basic Path With Templates

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: Test06
- Title: Test 06
- Description: Sixth Test
- Button: Next
- Button: Next
- Button: Next
- Check Sample Template 1
- Button: Finish

Expected Result

- Project with name “Test06” created
- Folders `Classes`, `Documentation`, `Documentation/en`, `Meta` and `Tests` created
- DocBook templates in folder `Documentation/en` with the project key and description created
- `Package.xml` with the key, the title and the description created
- File `Sample1.txt` in the root of the project created.

Actual Result

Passed

Execution 2008-07-23 10:01 / David Brühlmeier

Basic Path With Existing Project

Test Description

- Menu File > New > Other > PHP > PHP Project
- Project Name: PHP1
- Button: Finish
- Menu File > New > Other > DEV3 > FLOW3 Package
- Use Existing Project
- Folder: Browse > Select Project PHP1 > OK
- Key: Test07
- Title: Test 07
- Description: Seventh Test
- Button: Finish

Expected Result

- No new project was created
- Folders `Classes`, `Documentation`, `Documentation/en`, `Meta` and `Tests` created within project PHP1, subfolder `Test07`
- DocBook templates in folder `Documentation/en` with the project key and description created within project PHP1, subfolder `Test07`
- `Package.xml` with the key, the title and the description created within project PHP1, subfolder `Test07/Meta`

Actual Result

✓ Passed

Execution

2008-07-23 10:07 / David Brühlmeier

Basic Path (Minimal Input)

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: Test08
- Locales: Select > Deselect All > OK
- Additional Folders: Select > Deselect All > OK
- Button: Finish

Expected Result

- Project with name “Test08” created
- Folder `Meta` created
- `Package.xml` created

Actual Result

Passed

Execution 2008-07-23 13:08 / David Brühlmeier

Basic Path (Default Values)

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project

Expected Result

- Radio Button: Create new Project is selected
- Version: According to the property DEV3 > FLOW3 > Defaults for New Packages
- State: Development
- Locales: According to the property DEV3 > FLOW3 > Default Locales
- Lead Developer: According to the property DEV3 > FLOW3 > Defaults for New Packages

Actual Result

Passed

Execution 2008-07-23 10:19 / David Brühlmeier

Alternate Path (Missing Key)

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: (blank)

Expected Result

- An error message is displayed, informing the user that the package key is mandatory
- The button "Finish" is disabled

Actual Result

Passed

Execution 2008-07-23 10:09 / David Brühlmeier

Alternate Path (Invalid Key)

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: My Key With Blanks

Expected Result

- An error message is displayed, informing the user that the package key must only consist of letters and numbers
- The button "Finish" is disabled

Actual Result

Passed

Execution 2008-07-23 10:10 / David Brühlmeier

Alternate Path (Improper Key)

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: myKeyStartsWithLowercase

Expected Result

- An warning is displayed, informing the user that the package key should start with an uppercase letter
- The button "Finish" is enabled

Actual Result

Passed

Execution 2008-07-23 10:12 / David Brühlmeier

Alternate Path (Improper Version Number)

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: MyPerfectKey
- Version: 0.1.5.alpha

Expected Result

- An warning is displayed, informing the user that the version number should be in the format 0.0.0
- The button "Finish" is enabled

Actual Result

Passed

Execution 2008-07-23 10:13 / David Brühlmeier

Alternate Path (Missing Version Number)

Test Description

- Menu File > New > Other > DEV3 > FLOW3 Package
- Create New Project
- Key: MyPerfectKey
- Version: (blank)

Expected Result

- An error message is displayed, informing the user that the version number is mandatory
- The button "Finish" is disabled

Actual Result ✓ Passed

Execution 2008-07-23 10:14 / David Brühlmeier

5.1.2. UC002: Edit Configuration

Basic Path

Test Description
- Double-Click on `Package.xml` file

Expected Result
- A new editor window opens
- The editor title is "FLOW3 Package Editor"
- The editor displays three subpages: "Overview", "Involved Parties" and "Constraints"
- The information displayed corresponds to the information stored in the `Package.xml` file

Actual Result ✓ Passed

Execution 2008-07-23 10:28 / David Brühlmeier

Basic Path (Dirty Marker)

Test Description
- Double-Click on `Package.xml` file
- Edit any field in the editor

Expected Result
- An asteriks is displayed to the left of the editor title
- The save button and the menu File > Save entry are enabled

Actual Result ✓ Passed

Execution 2008-07-23 10:31 / David Brühlmeier

Basic Path (Consistent Save)

Test Description
- Double-Click on `Package.xml` file
- Edit any field in the editor
- Click on the save button

Expected Result
- The asteriks to the left of the editor title disappears
- The save button and the menu File > Save entry are disabled
- The Package.xml file is valid according to http://typo3.org/ns/2008/flow3/package/Package.rng

Actual Result ✓ Passed

Execution 2008-07-23 10:43 / David Brühlmeier

5.1.3. UC003: Edit PHP Code

Basic Path

Test Description
- Double-Click on a `*.php` file
- Enter "f3" and hit Ctrl+Space

Expected Result
- A list of FLOW3-related code snippets is displayed

Actual Result ✓ Passed

Execution 2008-07-23 10:50 / David Brühlmeier

5.1.4. UC005: Create Aspect

Basic Path

Test Description
- Select the `Classes` folder within an existing project
- Menu File > New > Other > DEV3 > FLOW3 Aspect

Expected Result
- A wizard is opened
- FLOW3 Package: Contains the key of the FLOW3 package in which the `Classes` directory is located
- Author: Is filled according to the preferences in DEV3 > FLOW3
- Author E-Mail: Is filled according to the preferences in DEV3 > FLOW3

Actual Result ✓ Passed

Execution 2008-07-23 11:02 / David Brühlmeier

Basic Path With Saving

Test Description
- Select the `Classes` folder within an existing project
- Menu File > New > Other > DEV3 > FLOW3 Aspect
- Name: MyAspect
- FLOW3 Package: MyPackage
- Button: Finish

Expected Result
- A new file `F3_MyPackage_MyAspect.php` is created in the folder `Classes`

- An editor is opened, displaying the contents of this new file
- The file contains an empty PHP class `F3_MyPackage_MyAspect`, implementing the interface `F3_FLOW3_AOPAspectInterface`
- The class is prefixed with a block comment with the following information:
  - Empty description
  - `@author` tag with the author name and author e-mail according to the preferences in DEV3 > FLOW3
  - Empty `@aspect` tag
  - `@package` tag with the key of the FLOW3 package in which the `Classes` directory is located

Actual Result

Passed

Execution 2008-07-23 10:55 / David Brühlmeier

Basic Path With Pointcuts

Test Description

- Select the `Classes` folder within an existing project
- Menu File > New > Other > DEV3 > FLOW3 Aspect
- Name: MyAspect
- FLOW3 Package: MyPackage
- Description: Lorem ipsum aspectum
- Pointcuts: Button Add
  - Type: Method
  - Name: myPointcut
  - Description: Lorem ipsum pointcutum
  - Method Visibility: Public
  - Method Name: *
  - Button: OK
- Button: Finish

Expected Result

- A new file `F3_MyPackage_MyAspect.php` is created in the folder `Classes`
- An editor is opened, displaying the contents of this new file
- The file contains a PHP class `F3_MyPackage_MyAspect`, implementing the interface `F3_FLOW3_AOPAspectInterface`

- The class is prefixed with a block comment with the following information:
  - Description: Lorem ipsum aspectum
  - `@author` tag with the author name and author e-mail according to the preferences in DEV3 > FLOW3
  - Empty `@aspect` tag
  - `@package` tag with the key of the FLOW3 package in which the `Classes` directory is located
- The class contains a single public function `myPointcut` with no arguments and no body.
- The function is prefixed with a block comment with the following information:
  - Description: Lorem ipsum pointcutum
  - `@pointcut` tag: `method(public *)`

Actual Result

Passed

Execution 2008-07-23 11:12 / David Brühlmeier

Basic Path (File Name)

Test Description

- Select the `Classes` folder of the package "MyPackage"
- Menu File > New > Other > DEV3 > FLOW3 Aspect
- Name: MyAspect
- Button: Next

Expected Result

- Parent folder: `MyPackage/Classes`
- File Name Of Aspect: `F3_MyPackage_MyAspect.php`

Actual Result

Passed

Execution 2008-07-23 11:24 / David Brühlmeier

Alternate Path (Missing Name)

Test Description

- Select the `Classes` folder within an existing project
- Menu File > New > Other > DEV3 > FLOW3 Aspect
- Name: (blank)

Expected Result

- An error message is displayed, informing the user that the aspect name is mandatory
- The button "Finish" is disabled

Actual Result

Passed

Execution 2008-07-23 11:16 / David Brühlmeier

Alternate Path (Invalid Name)

Test Description

- Select the `Classes` folder within an existing project
- Menu File > New > Other > DEV3 > FLOW3 Aspect
- Name: My Aspect Name With Blanks

Expected Result

- An error message is displayed, informing the user that the aspect name must only consist of letters and numbers
- The button "Finish" is disabled

Actual Result

Passed

Execution 2008-07-23 11:17 / David Brühlmeier

Alternate Path (Improper Name)

Test Description

- Select the `Classes` folder within an existing project
- Menu File > New > Other > DEV3 > FLOW3 Aspect
- Name: myApspectNameStartsLowercase

Expected Result

- An warning is displayed, informing the user that the package key should start with an uppercase letter
- The button "Finish" is enabled

Actual Result

Passed

Execution 2008-07-23 11:18 / David Brühlmeier

Alternate Path (Invalid File Name)

Test Description

- Select the `Classes` folder within an existing project
- Menu File > New > Other > DEV3 > FLOW3 Aspect
- Name: MyAspect
- Button: Next
- File Name Of Aspect: (blank)

Expected Result

- An error message is displayed, informing the user that the file name is mandatory
- The button "Finish" is disabled

Actual Result

Passed

Execution 2008-07-23 11:25 / David Brühlmeier

Alternate Path (Invalid Folder Name)

Test Description

- Select the `Classes` folder within an existing project
- Menu File > New > Other > DEV3 > FLOW3 Aspect
- Name: MyAspect
- Button: Next
- Parent Folder: (blank)

Expected Result

- An error message is displayed, informing the user that the parent folder is mandatory
- The button "Finish" is disabled

Actual Result

Passed

Execution 2008-07-23 11:26 / David Brühlmeier

5.1.5. UC007: Edit TypoScript 2.0

Basic Path

Test Description

- Double-Click on a `*.ts2` file

Expected Result

- A new editor window opens
- The editor title is the file name of the TypoScript file
- The editor displays no subpages
- The information displayed corresponds to the information stored in the `*.ts2` file

Actual Result

Passed

Execution 2008-07-23 12:07 / David Brühlmeier

Basic Path (Dirty Marker)

Test Description

- Double-Click on a `*.ts2` file
- Edit anything in the editor

Expected Result

- An asteriks is displayed to the left of the editor title
- The save button and the menu File > Save entry are enabled

Actual Result

Passed

Execution 2008-07-23 12:09 / David Brühlmeier

Basic Path (Problem Marker)

Test Description

- Double-Click on an empty `*.ts2` file
- Enter: "This is not valid TypoScript"

Expected Result

- An error marker is displayed at the first line
- The `*.ts2` file is marked with an error cross
- The Problems View displays the error message, including the correct file name and the correct line number
- When hovering over the error marker, a pop-up with the same error message as in the Problems View is displayed

Actual Result

Passed. However, due to the prototype implementation of the TypoScript parser, the error message displayed is often not helpful.

Execution 2008-07-23 12:25 / David Brühlmeier

Basic Path (Folding)

Test Description

- Double-Click on an empty `*.ts2` file
- Enter: `object {` and newline
- Enter: `value = "MyValue"` and newline
- Enter: `}`

Expected Result

- In the left margin of the editor, a folding icon (a circled minus sign) is displayed
- When clicking on the folding icon, the second line (`value = "MyValue"`) disappears
- When hovering over the left margin of the folded region, the folded line appears as pop-up

Actual Result

Folding works, but hovering over a folded region does not always display the correct information. See `bug 1111`.

Execution 2008-07-23 12:40 / David Brühlmeier

Basic Path (Syntax Highlighting)

Test Description

- Double-Click on an empty `*.ts2` file

- Enter: `/*` and newline
- Enter: `* Block Comment` and newline
- Enter: `*/` and newline
- Enter: `// Single Line Comment` and newline
- Enter: `# Single Line Comment` and newline
- Enter: `include: source = "include.ts2"` and newline
- Enter: `namespace: test = F3_Test_TypoScript`

Expected Result

- The block comment must be displayed in brown (RGB 128/0/0)
- The single line comments must be displayed in brown/green (RGB 128/128/0)
- The keywords `include` and `namespace` must be displayed in blue (RGB 0/0/128)
- The string `"include.ts2"` must be displayed in green (RGB 0/128/0)
- All other characters must be displayed in black (RGB 0/0/0)

Actual Result

Passed

Execution

2008-07-23 12:49 / David Brühlmeier

Basic Path (Content Assist)

Test Description

- Double-Click on a `*.ts2` file
- Enter: `myObject.`

Expected Result

- After a short delay, the content assistant pop-up must be displayed.
- The content assistant must contain an number of proposals (properties and processors)
- Properties and processors must have different icons
- When selecting a proposal, an additional pop-up must open, displaying a description of the property/processor, including the expected/returned data types.

Actual Result

Passed. However, due to the prototype implementation of the content assistant, the proposals are not filtered according to context information.

Execution

2008-07-23 14:07 / David Brühlmeier

Outline View (Default Namespace)

Test Description

- Double-Click on an empty `*.ts2` file
- Window > Show View > Outline

Expected Result
- The Outline View is displayed
- The Outline View shows the following content:
  - Namespaces
    - default: F3_TYPO3_TypoScript
  - Includes (empty)

Actual Result

Passed

Execution 2008-07-23 12:54 / David Brühlmeier

Outline View (Override Default Namespace)

Test Description
- Double-Click on an empty `*.ts2` file
- Enter: namespace: `default = F3_Test_TypoScript`
- Save
- Window > Show View > Outline

Expected Result
- The Outline View is displayed
- The Outline View shows the following content:
  - Namespaces
    - default: F3_Test_TypoScript
  - Includes (empty)

Actual Result

Passed

Execution 2008-07-23 12:54 / David Brühlmeier

Outline View (Namespaces and Includes)

Test Description
- Double-Click on an empty `*.ts2` file
- Enter: `namespace: test = F3_Test_TypoScript` and newline
- Enter: `include: source = "firstInclude.ts2"` and newline
- Enter: `include: source = "secondInclude.ts2"`
- Save
- Window > Show View > Outline

Expected Result
- The Outline View is displayed
- The Outline View shows the following content:

- Namespaces
  - default: F3_TYPO3_TypoScript
  - test: F3_Test_TypoScript
- Includes
  - source: firstInclude.ts2
  - source: secondInclude.ts2

Actual Result

Passed

Execution 2008-07-23 12:59 / David Brühlmeier

Outline View (Positioning)

Test Description

- Double-Click on an empty `*.ts2` file
- Enter: `namespace: test = F3_Test_TypoScript` and newline
- Enter: `include: source = "firstInclude.ts2"` and newline
- Enter: `include: source = "secondInclude.ts2"`
- Enter as many newlines necessary for the typed lines to be scrolled beyond the viewport.
- Save
- Window > Show View > Outline
- Click on `source: secondInclude.ts2` in the Outline View

Expected Result

- The editor scrolls to the top of the file
- The second line is selected

Actual Result

Passed

Execution 2008-07-23 13:02 / David Brühlmeier

5.1.6. UC010: Delete Element

Basic Path

Test Description

- Select the element to be deleted (i.e. the `Package.xml` file or an Aspect) in the Navigator View
- Right-Click and and select “Delete” or hit the Delete key
- Button: Yes

Expected Result
- The element is deleted

Actual Result

Passed

Execution 2008-07-23 17:24 / David Brühlmeier

Alternate Path (Cancel)

Test Description
- Select the element to be deleted (i.e. the `Package.xml` file or an Aspect) in the Navigator View
- Right-Click and and select "Delete" or hit the Delete key
- Button: No

Expected Result
- The element is not deleted

Actual Result

Passed

Execution 2008-07-23 17:26 / David Brühlmeier

5.2. Non-Functional Tests

These tests cover the non-functional requirements as published in the requirement specification (revision 004 of 2008-04-02).

5.2.1. Usability

RQ018: Custom Templates

Test Description
- Create a new Eclipse plug-in
- Implement the extension point `org.typo3.forge.dev3.flow3.package.templates`
- Run DEV3
- Menu File > New > Other > DEV3 > FLOW3 Package
- Go to the last page of the wizard

Expected Result
- The template provided by the new extension is displayed

Actual Result

Passed

Execution 2008-07-23 13:21 / David Brühlmeier

RQ021: Multi-Language

Test Description
- Start Eclipse with the option -nl de_DE

- Menu File > New > Other > DEV3 > FLOW3 Package

Expected Result
- The labels must be displayed in German

Actual Result

Multi-language support is not yet implemented. See `bug 1112`.

Execution 2008-07-23 13:40 / David Brühlmeier

RQ022: Online Documentation

Test Description
- Menu Help > Help Contents
- Select "DEV3 User Guide"

Expected Result
- The DEV3 User Guide is displayed

Actual Result

Passed.

Execution 2008-07-24 19:02 / David Brühlmeier

RQ023: Shortcuts

DEV3 does not yet offer any shortcuts (not needed).

5.2.2. Reliability

RQ024: Error Logs

Test Description
- Go to the installation directory of Eclipse
- Go to the directory plugins\org.typo3.forge.dev3\schema
- Rename the file `TypoScript.xml` to `TypoScript00.xml`
- Start Eclipse
- Double-Click on a `*.ts2` file
- Enter `.` in the TypoScript 2.0 Editor

Expected Result
- No code completion is displayed
- In the logfile `{Eclipse Workspace}\.metadata\.log`, there must be the following exception found: `java.io.IOException: Cannot find schema/TypoScript.xml`

Actual Result

Passed

Execution 2008-07-23 14:01 / David Brühlmeier

RQ025: Wrong Or Insufficient Input

Covered by the following tests:

- Create Package (UC001): Improper Key
- Create Package (UC001): Improper Version Number
- Create Package (UC001): Invalid Key
- Create Package (UC001): Missing Key
- Create Package (UC001): Missing Version Number
- Create Aspect (UC005): Improper Name
- Create Aspect (UC005): Invalid File Name
- Create Aspect (UC005): Invalid Folder Name
- Create Aspect (UC005): Invalid Name
- Create Aspect (UC005): Missing Name

5.2.3. Performance

RQ027: Lazy Initialization

Test Description	• Go the the source distribution of DEV3 • Open the file `org.typo3.forge.dev3.core.DEV3Plugin.java` • Go to the method `getTypoScript()`
Expected Result	• The instance of `org.typo3.forge.dev3.internal.jaxb.typoScript.TypoScript` must be created using the Singleton pattern. • The instance of `org.typo3.forge.dev3.internal.jaxb.typoScript.TypoScript` is only created if needed.
Actual Result	✓ Passed
Execution	2008-07-23 14:28 / David Brühlmeier

5.2.4. Supportability

RQ028: Coding Guidelines

The Eclipse Coding Guidelines were respected.

RQ029: Unit Tests

Test Description	• Install Eclemma (see `http://www.eclemma.org/`)

- Import the source distribution of DEV3
- Open the plug-in `org.typo3.forge.dev3.tests`
- Select the package `org.typo3.forge.dev3.tests.flow3`
- From the Eclemma menu, choose "Coverage As" > "JUnit Test"

Expected Result

- The code coverage of the classes in the `org.typo3.forge.dev3.model.*` packages must be at least 75%.

Actual Result

Code Coverage Statistic

- `org.typo3.forge.dev3.model.aspect: 82.5%`
- `org.typo3.forge.dev3.model.docBook: 0.0%`
- `org.typo3.forge.dev3.model.flow3Package: 52.9%`
- `org.typo3.forge.dev3.model.php: 68.9%`
- `org.typo3.forge.dev3.model.typoScript: 68.5%`

Execution

2008-07-23 14:38 / David Brühlmeier

RQ030: Update Site

Test Description

- Menu Help > Software Updates > Find and Install...
- Search for new features to install
- Button: Next
- Button: New Remote Site
- Name: DEV3 Update Site
- URL: http://www.dev3.org/update/
- Button: OK
- Button: Finish
- Select DEV3 Udpate Site
- Button: Next
- I accept the terms in the license agreement
- Button: Next
- Button: Finish
- Button: Install
- Button: Yes

- Menu Help > About Eclipse Platform
- Button: Plug-in Details

Expected Result

- The plug-in with the ID `org.typo3.forge.dev3` must be in the list of installed plug-ins.

Actual Result

Passed

Execution

2008-07-23 14:46 / David Brühlmeier

Chapter 6. Tracking

This chapter tracks the fulfillment of the requirement specification (revision 004 of 2008-04-02). It has the same structure as the requirement specification to facilitate navigation. Each use-case and each requirement is assessed as "Fully Implemented", "Partially Implemented" or "Not Implemented". For all features which are not fully implemented, a reason for the gap is documented. Furthermore, all implemented features are verified by tests, as documented in Chapter 5, *Tests*.

6.1. Use-Cases

6.1.1. UC001: Create Package

Assessment  Fully implemented.

Verified By Functional Tests: UC001 (Create Package)

6.1.2. UC002: Edit Configuration

Assessment  Fully implemented.

Verified By Functional Tests: UC002 (Edit Configuration)

6.1.3. UC003: Edit PHP Code

Assessment  Partially implemented.

One basic design decision for DEV3 was to not implement an own PHP editor (see Design Principle: No PHP Editor). It would be a waste of resources to do that, since there are good PHP editors available, even as open-source such as PDT. Another principle is that DEV3 should stay independant of the PHP editor used (see Design Principle: Independant of PHP Editor). These two principles are somewhat conflicting and lead to the fact that DEV3 cannot implement editor-specific features unless they can be realized with an extension point of the generic editor provided by Eclipse.

For this reason, I have decided to provide coding templates to implement this use-case. Coding templates are generic and can be used in any editor. The FLOW3 devlopment team has already used such templates internally, and they are now included in the distribution of DEV3.

Verified By Functional Tests: UC003 (Edit PHP Code)

6.1.4. UC005: Create Aspect

Assessment  Fully implemented.

Verified By Functional Tests: UC005 (Create Aspect)

6.1.5. UC006: Maintain Aspect

Assessment  Not implemented.

The reason for not implementing this use-case is the same as for only partially implementing UC003 “Edit PHP Code”. Noteably, all the functionality specified this use-case is already implemented in UC005 “Create Aspect”. If the design principle to stay neutral from a specific PHP editor should one day be abandoned, it would be relatively easy to re-use this functionality in the context of an editor as well as in the existing wizard.

Verified By n/a

6.1.6. UC007: Edit TypoScript 2.0

Assessment  Partially implemented.

After completion of the second iteration, it became clear that there would not be enough time to fully implement the TypoScript 2.0 editor. Mr. Seewer and I came to the agreement that it would still be useful to implement this use-case, even if it was just as a proof-of-concept. Consequently, I started by the implementation of a top-down parser for TypoScript 2.0. In the process of this implementation, I had to refine the ENBF grammar which was defined more as a draft than anything else. The parser is now fully implemented, but with the limitation that it stops parsing at the first error it encounters. This is hardly acceptable for a decent editor, but enough for a prototype.

Syntax highlighting is fully implemented, but the coloring scheme is currently hard-coded. This should be made configurable through the standard Eclipse preferences.

Folding is also fully implemented, with the restriction of the bug documented in the test Edit TypoScript 2.0 (UC007):Basic Path (Folding).

The Outline View is also implemented as a prototype. It currently shows only namespaces and includes. What remains to be answered is the almost philosophical question: “What's the purpose of the Outline View?” Personally, I see the Outline View as a tool for fast navigation in a large resource. This means it is essential that all major elements of the resource can be selected in the Outline View. In the case of TypoScript, this brings up a fundamental problem: Elements can be defined, i.e. overwritten, at any time. The question is: “What should be displayed in the Outline View if an element has been overwritten by another one?” Logically, the overwritten element does not exist any more. If it is not displayed, however, then the purposose of the Outline View being a navigation tool is defeated. If *is* displayed, the outline view becomes nothing more than a 1:1 representation of what can be seen in the editor.

Content Completion is also implemented as a prototype. One major aspect here is the fact that the content proposals should be based upon three factors:

1. The installed packages on the target platform
2. The TypoScript classes contained in the installed packages
3. The current context in which the content assistant is called

The first aspect requires the knowledge of the installed packages on the target platform. This could be solved in several ways, e.g.

- The packages are listed in the preferences.

- FLOW3 core implements a Webservice through which the installed packages can be queried. The URL of the target installation (e.g. in the preferences) would then be enough for DEV3 to determine the relevant packages.

- The target installation is local and DEV3 only needs to know the location(s) of the installation (e.g. in the preferences, similiar to the "Build Path" in JDT).

The second aspect can be solved in two ways:

- DEV3 scans the PHP code to find TypoScript classes (all TypoScript classes are implementing the interface `F3_TypoScript_ObjectInterface`, either directly or indirectly).

  This approach has some major concerns:

  - All PHP code must be parsed. Because the interface might be implemented indirectly, even the inheritance hierachy must be identified and parsed. This could be simplified a bit if the FLOW3 coding guidelines would require a specific annotation (e.g. `@TypoScriptObject`) for such classes.

  - All PHP must be actually available to DEV3. This can be quite a disadvantage if working with remote installations, because all packages would have to be fully downloaded before content proposals become possible.

- FLOW3 requires an additional file, e.g. `Meta/TypoScript.xml`, in the case where TypoScript classes are part of the package and in case content proposals should be offered.

  This is the suggestion I made to the FLOW3 development team, including a draft of a schema. It would drastically simplify the implementation of content proposals, not only for DEV3 but for any TypoScript editor (e.g. for an online editor implemented in AJAX). The disadvantage of this approach is the redundancy created between the PHP source code and the `TypoScript.xml` file. This might be solved by a new feature for DEV3 which parses the PHP code in the background and constantly updates the `TypoScript.xml` file (like the creation of the `.class` files from the `.java` files in JDT). Of course, this would again lead to the need of parsing PHP code with the problems described above.

  The third aspect requires parsing of the TypoScript into an appropriate model. This could already be done with the current parser/evaluator, but with the severe limitation that parsing stops at the first error encountered. After the experiences I made with the implementation of this editor I'm convinced that a top-down parser cannot be used in the context of an editor. In order to provide the user with meaningful error messages, a much more "clever" and flexible approach is needed. This means that offering context-sensitive content proposals requires the redesign of the current parser/evaluator.

Verified By Functional Tests: UC007 (Edit TypoScript 2.0)

6.1.7. UC010: Delete Element

Assessment  Fully implemented.

Verified By Functional Tests: UC010 (Delete Element)

6.2. Functional Requirements

6.2.1. New Package Wizard

RQ001: New Package Wizard

Assessment  Fully implemented.

Verified By Functional Tests: UC001 (Create Package)

RQ002: Mandatory Directories/Files

Assessment  Fully implemented.

Verified By Create Package (UC001): Missing Key

RQ003: Optional Directories

Assessment  Fully implemented.

Verified By Create Package (UC001): Additional Folders

RQ004: Templates

Assessment  Fully implemented.

Verified By Create Package (UC001): Templates

RQ031: Default Values For New Package Wizard

Assessment  Fully implemented.

Verified By Create Package (UC001): Default Values

RQ005: Check For Valid Package Keys

Assessment  Fully implemented.

Verified By
- Create Package (UC001): Missing Key
- Create Package (UC001): Invalid Key
- Create Package (UC001): Improper Key

RQ006: Check For Valid Version Numbers

Assessment  Fully implemented.

Verified By
- Create Package (UC001): Improper Version Number
- Create Package (UC001): Missing Version Number

6.2.2. New Aspect Wizard

RQ007: Check For Valid Aspect Names

Assessment

Fully implemented.

Verified By
- Create Aspect (UC005): Missing Name
- Create Aspect (UC005): Invalid Name
- Create Aspect (UC005): Improper Name

RQ008: New Aspect Wizard

Assessment

Fully implemented.

Verified By Functional Tests: UC005 (Create Aspect)

6.2.3. PHP Editor

RQ010: AOP Annotations

Assessment

Not implemented.

Please refer to the explanations in Tracking UC006: Maintain Aspect.

Verified By n/a

RQ012: New Exceptions

Assessment

Not implemented.

Please refer to the explanataion in Tracking UC003: Edit PHP Code.

Verified By n/a

6.2.4. Configuration Editor

RQ013: Configuration Editor

Assessment

Fully implemented.

Verified By Functional Tests: UC002 (Edit Configuration)

RQ014: Configuration Editor GUI

Assessment  Fully implemented.

Verified By Edit Configuration (UC002): Basic Path

6.2.5. TypoScript 2.0 Editor

RQ015: TypoScript 2.0 Editor

Assessment  Fully implemented.

Verified By Functional Tests: UC007 (Edit TypoScript 2.0)

RQ016: Content Assist

Assessment  Fully implemented.

Verified By Edit TypoScript 2.0 (UC007): Content Assist

RQ017: Syntax Highlighting

Assessment  Fully implemented.

Verified By Edit TypoScript 2.0 (UC007): Syntax Highlighting

6.3. Non-Functional Requirements

6.3.1. Usability

RQ018: Custom Templates

Assessment  Partially implemented.

Templates are currently only provided by the "New Package Wizard". The reason is that I became unsure about the value of this feature for the creation of aspects. Creating an aspect in FLOW3 means nothing more than creating a PHP class. I could not figure a real-world requirement for providing additional functionality to this in the form of templates. Perhaps it might be more useful to provide an extension point where a third-party plug-in could add additional code for the PHP class.

Verified By
- Create Package (UC001): Templates
- Non-Functional Tests: RQ018 (Custom Templates)

RQ021: Multi-Language

Assessment  Not implemented.

The implementation of multi-language support has been dropped because of the time constraint for this master thesis. I have planned to add this feature in the next version of DEV3.

Verified By Non-Functional Tests: RQ021 (Multi-Language)

RQ022: Online Documentation

Assessment  Fully implemented.

Verified By Non-Functional Tests: RQ022 (Online Documentation)

RQ023: Shortcuts

Assessment  Not implemented.

During the implementation of the DEV3 features, I did not encounter a functionality which would be meaningful for a specific shortcut.

Verified By Non-Functional Tests: RQ023 (Shortcuts)

6.3.2. Reliability

RQ024: Error Logs

Assessment  Fully implemented.

Verified By Non-Functional Tests: RQ024 (Error Logs)

RQ0254 Wrong Or Insufficient Input

Assessment  Fully implemented.

Verified By Non-Functional Tests: RQ025 (Wrong Or Insufficient Input)

6.3.3. Performance

RQ027: Lazy Initialization

Assessment  Fully implemented.

Verified By Non-Functional Tests: RQ027 (Lazy Initialization)

6.3.4. Supportability

RQ028: Coding Guidelines

Assessment  Fully implemented.

Verified By Non-Functional Tests: RQ028 (Coding Guidelines)

RQ029: Unit Tests

Assessment  Partially implemented.

I have tried to apply to the principle of "Test Driven Development" for DEV3. However, I did not fully succeed to follow this principle (see Section 7.5, "Agile Development"). Furthermore, automated testing of UI classes is quite complex. For these reasons, I decided to limit unit tests for all model classes. For all model packages except `org.typo3.forge.dev3.model.docBook`, I have reached an average code coverage of about 70%, which is slightly less than I had anticipated. The DocBook model is a bit special because it conflicts with the Design Principle: Model independant from Eclipse. In order to test this model, Eclipse would have to be run in headless mode, something I had not yet implemented in the current tests. This is the reason why this package is yet uncovered by unit tests.

Verified By Non-Functional Tests: RQ029 (Unit Tests)

RQ030: Update Site

Assessment  Fully implemented.

Verified By Non-Functional Tests: RQ030 (Update Site)

Chapter 7. Conclusion

7.1. Requirements Specification

I chose to write the requirement specification in the way taught in the specialization level course "Requirements Engineering", using use-cases, functional and non-functional requirements. In my opinion, this is a solid methodology and was absolutely sufficient for a project of this size. Perhaps some UI prototypes as taught in the specialization level course "Usability" might have been helpful, but their absence was not critical.

7.2. Project Site

I used the project site of TYPO3, `forge.typo3.org`, as the central platform for this master thesis and its resulting product DEV3. Forge offers a Subversion repository, and all code and documentation was committed to this repository during the course of development. This is (or should be) the standard way to use in any project and it served its purpose very well. It also allows to temporarily change code and then revert to the previous state if required.

Forge also offers features to track issues and to keep a roadmap. I effectively used both of these features, which resulted in a high level of transparency for all involved parties.

7.3. Time Management

During the course of this master thesis, I had reserved 1.5 days per week to work on it. I usually worked every Tuesday and every other Monday for DEV3. Having two full days in a row helped increasing my efficiency, because I could stay focused on a certain point for a longer period of time.

After the second iteration, it became clear to me that I would not be able to finish everything I had planned in the requirement specification. In the following discussion with Mr. Seewer, it became clear that it would be too much of a shortcoming to completely drop the implementation of the TypoScript 2.0 editor. We therefore agreed to limit this implementation to a prototype. This was a good decision, because I did learn a lot about the pitfalls in developing an editor for Eclipse.

All in all, my time mangement worked out as planned, even though it was quite tight towards the end.

7.4. UI Development

The development of UI elements took much longer than anticipated. This was partly due to the fact that Eclipse uses SWT, and in the GUI course we had learned Swing. After the first iteration, I decided to use a graphical editor (see Section 7.7.6, "SWT Designer"), which helped, but still a lot of manual coding was necessary.

I only know C# and Visual Studio from the short introductory course we had, must I still have the impression that UI Development in C# is much easier than in Java. This is probably mostly due to the excellent graphical tools in Visual Studio and less a problem of the language itself. But it still amazes me that something so commonplace as UI development is still so much of a hassle in Java.

7.5. Agile Development

FLOW3 is developed using many principles of agile development. I was curious about this development model and decided to try out the principle of "Test Driven Development" for my master thesis.

Test Driven Development in its pure form requires automated tests for everything being developed. The tests have to be written *before* the implementation is started. The implementation is completed once the unit-tests are successful. One major advantage of this principle is the fact that it forces the developer to think about the

design of a class in the form of a unit-test. Another obvious advantage is that the test coverage of the final product will be very high. And these unit-tests allow for refactoring, which is another major principle in agile development.

Unfortunately, I did not succeed in truly applying this principle for the following reasons:

- Test Driven Development requires a change of mind. I have been developing for quite some time and I could not get used to start developing by writing a (failing) test, even though I do realize the benefits. I suppose it takes time for this mindshift to take place.
- I have never developed an Eclipse plug-in before. I understood many of the concepts only once I had actually used them. This made it very difficult to design a class as detailled as to be able to write a unit-test.
- Any changes in the design require twice as much effort once the unit-tests are written, because not only the classes need to be changed but also the unit-tests. Personally, I think this is one of the biggest disadvantages of Test Driven Development.

7.6. Documentation

FLOW3 packages are documented in DocBook, an open XML format for technical documentation. This format was new to me and I decided to use it for my master thesis to get to know it better.

All of this documentation is written in DocBook and then rendered to different output formats using XSL (see Section 7.8, "Other Tools"). The major advantage of this approach is that the DocBook sources can easily be commited to Subversion, including a meaningful diff, because everything is XML and therefore text-based. Another major advantage is the relatively simple creation of different output formats.

However, with a hindsight, I am not sure if I would take the same decision again. It took me a lot of time to get to know DocBook and the DocBook XSL stylesheets with their customization possibilities. Furthermore, I did run into a number of minor problems when rendering which all in all cost me even more time. The requirements were all documented in Enterprise Architect and exported as XMI, so I had to write stylesheets to transform XMI to DocBook, which again cost me time.

I guess for strictly technical documentation, such as the documentation of a FLOW3 package, DocBook is a good choice. For the documentation of this master thesis, it was probably an overkill. I view it as an investment in my personal skills.

7.7. Eclipse Plug-Ins

7.7.1. Eclipse JDT/PDE

Obviously, Eclipse `JDT` was used for Java development and `PDE` for plug-in development. Both are very powerful and sophisticated tools and both are provided as open-source. It was a pleasure to work with these tools and a good opportunity to get to know JDT even better.

7.7.2. JUnit

I used `JUnit` integrated in Eclipse for all unit tests. JUnit is the standard for unit testing of Java and there was no feature missing to me.

7.7.3. Eclemma

I used `Eclemma` to evaluate the test coverage of my JUnit tests. It is an open-source tool, fulfills all my requirements and is easy to use. I especially appreciated the coloring feature which made it easy to spot which codelines were not yet covered by tests.

7.7.4. Subclipse

The `Subclipse` plug-in for Eclipse offers a direct integration of a Subversion repository. It's open-source, works very well and is reliable, I can only recommend it.

7.7.5. Ant

I used `Ant` for the building process of the documentation only. To build the plug-in, I used the regular build process offered by PDE. Ant is very powerful and it took me some time to get to know it. But since it is a very reliable tool, it was fun to work with it once I got to know it better.

7.7.6. SWT Designer

I started the project coding all the SWT UI elements by hand. After the first iteration, I decided to use a graphical designer. Since the open-source Visual Editor does not seem to be developed any more, I decided to use `SWT Designer` by Instantiations, a tool I had already used in the GUI course. I purchased an academic license which was reasonably priced.

This tool helped a lot developing the "raw" version of a new UI element, but there usually still remains a lot of manual coding.

7.7.7. Oxygen

For authoring of XML files, except for DocBook, I used the Eclipse plug-in of `Oxygen`. I was able to use the academic license we had received in the XML course. It is a very powerful tool with an excellent XML editor and it helped me a lot, especially when writing XSL stylesheets. However, I think it is overpriced and I would not have used it if I had to purchase it.

7.8. Other Tools

7.8.1. Enterprise Architect

`Enterprise Architect` is a CASE (Comuter Aided Software Engineering) tool by Sparx Systems. This tool must be licensed, but the fees are relatively low (between EUR 99 and EUR 179), especially when compared to other products such as Rational Rose. For my master thesis, I was able to purchase an academic license for an even lower price.

I used Enterprise Architect for the complete Inception phase, i.e. use-case modelling and requirements specification. Also the glossary is contained in Enterprise Architect. I did not use the forward-engineering features to generate code from the UML models, but reverse-engineered the finished code for documentation purposes. All UML graphics in this document are generated by Enterprise Architect.

In my view, Enterprise Architect is a very useful and easy-to-use CASE tool. It offers a lot of features and supports many different programming languages and all of that at an excellent cost/performance ratio. I will definitely use it again for my next projects.

7.8.2. XMLmind XML Editor

For the authoring of the DocBook files, I used the `XML Editor` by XMLmind. This is actually a generic XML editor, which offers some additional features for DocBook. I used the free "Personal Edition" which offered all I needed. This tool basically offers a semi-WYSIWYG way to edit DocBook files, which is the good side. The bad side is its usability. It took me a long time to get used to some of the "concepts" used in this tool and I still get annoyed when I can't figure out some of the most basic things. Unfortunately, there does not seem to be a better tool for this purpose around, at least not a free one. It's still better than editing DocBook XML in a text editor, but it's not a very efficient tool to work with.

7.8.3. DocBook XSL

The `DocBook XSL` stylesheets are offered to the world as open source. They support rendering DocBook to different output formats, such as PDF (through XSL-FO), HTML, Eclipse Help, Text, etc. Unfortunately, the documentation must be licensed. I have decided to purchase a license and this was a good decision. The stylesheets are very powerful and have a modular architecture which allows for customization on different levels. All of that would have been difficult to understand for me without the documentation. Even *with* the documentation, it did take me some time to get everything up and running, but once it was in place, generating output from DocBook was easy. When working with DocBook, these stylesheets are a must.

Appendix A. Actors

A.1. User

ID AC003

Description This actor is used in the documentation to represent any actor. The other actors are derived from this actor.

Use-Cases ---

A.2. Developer

ID AC001

Description A developer is the main user of the system. The developer is interested in efficiently creating and maintaining FLOW3 packages and typically has good to expert skills in PHP. He might also maintain content, although this is not one of his typical tasks.

Use-Cases
- Edit Content Repository (UC004)
- Delete Element (UC010)
- Edit PHP Code (UC003)
- Create Aspect (UC005)
- Maintain Aspect (UC006)
- Edit Configuration (UC002)
- Create Package (UC001)
- Edit TypoScript 2.0 (UC007)
- Create Content (UC008)
- Maintain Content (UC009)

A.3. Editor

ID AC002

Description An editor is responsible for creating, editing and deleting content. He is not interested in the technical background of the system and typically is not a developer. His main focus is to efficiently use the system to communicate.

Use-Cases
- Edit TypoScript 2.0 (UC007)
- Maintain Content (UC009)
- Create Content (UC008)
- Delete Element (UC010)

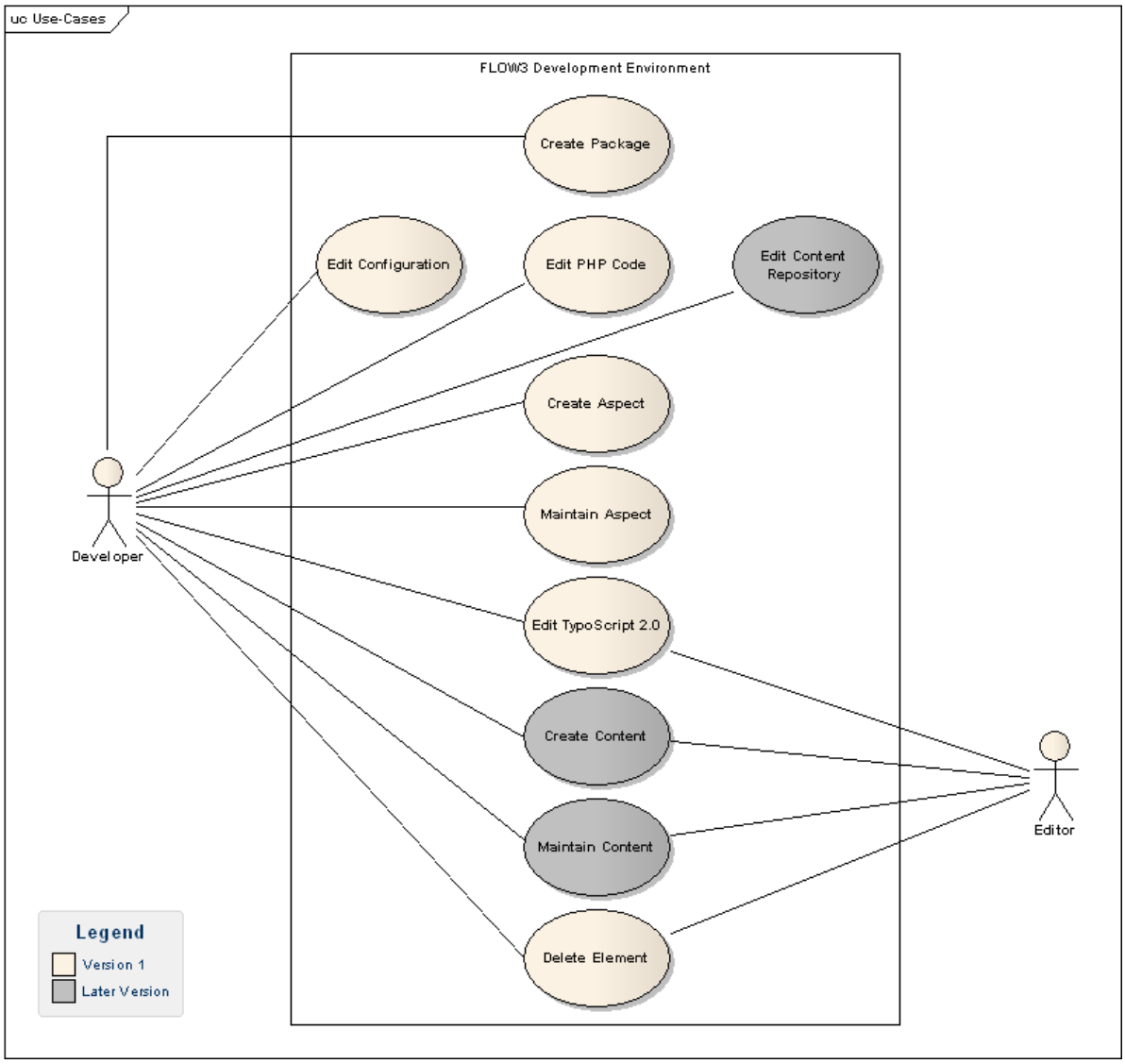
Appendix B. Use-Cases

This chapter contains the detailled descriptions of all use-cases. The use-cases which will be developed during this Master Thesis are marked in the diagram below and in the descriptions. Each use-cases is assigend to a certain version number. The deadlines for the three versions planned are as follows:

- Version 0.1: 29.04.2008
- Version 0.2: 03.06.2008
- Version 0.3: 14.07.2008

The references from/to actors, use-cases and requirements are links and can be used to easily navigate within this document.

B.1. Overview



Use-Cases for the FLOW3 Development Environment in its final version

B.2. Create Package

ID	UC001
Version	This use-case is planned to be implemented in version 0.1 of the FLOW3DE.
Description	This use-case covers the creation of new packages.
Actors	Developer (AC001)
Preconditions	• A project must be available. Please note: The creation of a project is outside the scope of the FLOW3DE. Any type of Eclipse project can be used. Normally, it will be a PHP project created by Zend PDE, but it can be just a generic project or a project created by another PHP plugin.
Basic Path	1. Select folder in which the new package will be created. 2. Click on "Create new Package" wizard. 3. Fill in the package key (mandatory). 4. Fill in the package title (mandatory). 5. Fill in the package description (mandatory). 6. Fill in the version of the package (default = 0.0.1, mandatory) 7. Select a category (optional). 8. Fill in the name of the author (optional). 9. Fill in the e-mail address of the author (optional). 10. Select the additional folders which shall be created by the wizard. 11. Optionally choose the templates which shall be generated by the wizard. 12. Click on Finish.
Alternate Paths	• Wrong Or Missing Information 1. The user enters wrong information or doesn't enter a required information. 2. The system disables the "Finish" button and displays a message explaining how to solve the problem.
Postconditions	• All files and directories required by the FLOW3 Framework, such as the Package.xml file, are created • If the actor has selected templates, the chosen scaffolding classes are created.
Additional Requirements	• Custom Templates (RQ018) • Mandatory Directories/Files (RQ002) • Check For Valid Version Numbers (RQ006)

- Check For Valid Package Keys (RQ005)
- Templates (RQ004)
- New Package Wizard (RQ001)
- Default Values For New Package Wizard (RQ031)
- Optional Directories (RQ003)

B.3. Edit Configuration

ID	UC002
Version	This use-case is planned to be implemented in version 0.2 of the FLOW3DE.
Description	This use-case covers editing the meta information of a specific package.
Actors	Developer (AC001)
Preconditions	• The Package.xml file must be available.
Basic Path	1. The user double-clicks on the Package.xml file. 2. The system opens a specialized editor for this file. 3. The user changes the configuration as needed. 4. The user clicks on "Save".
Alternate Paths	---
Postconditions	• The Package.xml file is saved in the latest state. • The updated Package.xml is well-formed XML and is valid according to the appropriate schema.
Additional Requirements	• Configuration Editor GUI (RQ014) • Configuration Editor (RQ013)

B.4. Edit PHP Code

ID	UC003
Version	This use-case is planned to be implemented in version 0.2 of the FLOW3DE.
Description	This use-case covers editing PHP code within a package. Please note: The FLOW3DE will not provide a PHP editor on its own, but it will extend existing plugins, such as the freely available PDT (PHP Development Tools) plugin.
Actors	Developer (AC001)
Preconditions	• The PHP file to be edited must be available.
Basic Path	1. Double-click on the PHP file to be edited. 2. Make the changes to the PHP code. 3. Click on "Save".
Alternate Paths	---
Postconditions	• The PHP file must be saved in the latest state.
Additional Requirements	• Check For CGL-Compliance (RQ011) • AOP Annotations (RQ010) • New Exceptions (RQ012)

B.5. Edit Content Repository

This use-case will be part of a future release of the FLOW3DE. At the current state of development, it is still quite unsure what steps will be needed for this use-cases. For this reason, this use-case does not yet contain more specific requirements.

ID	UC004
Description	This use-case covers editing Content Repository (CR) definitions (the structure, not the content).
Actors	Developer (AC001)
Preconditions	• The content repository must be available (either locally or remotely).
Basic Path	---
Alternate Paths	---
Postconditions	• The changes to the content repository layout are saved consistently.
Additional Requirements	---

B.6. Create Aspect

ID	UC005
Version	This use-case is planned to be implemented in version 0.2 of the FLOW3DE.
Description	This use-case covers the creation of new Aspects.
Actors	Developer (AC001)
Preconditions	• The package for which the aspect shall be created must be available.
Basic Path	1. Choose the package for which the aspect shall be created. 2. Click on "Create new Aspect" wizard. 3. Enter the name of the aspect (mandatory) 4. Enter the description of the aspect (optional) 5. Enter the name of the author (optional) 6. Enter the e-mail address of the author (optional) 7. Choose to add pointcuts (optional) 8. Click on Finish.
Alternate Paths	• Wrong Or Missing Information 1. The user enters wrong information or doesn't enter a required information. 2. The system disables the "Finish" button and displays a message explaining how to solve the problem.
Postconditions	• The aspect is saved consistently.
Additional Requirements	• New Aspect Wizard (RQ008) • Check For Valid Aspect Names (RQ007) • Custom Templates (RQ018)

B.7. Maintain Aspect

ID	UC006
Version	This use-case is planned to be implemented in version 0.2 of the FLOW3DE.
Description	This use-case covers the whole lifecycle of an aspect, excluding its creation.
Actors	Developer (AC001)
Preconditions	• The aspect to be edited must be available.
Basic Path	1. Choose the aspect for which the advice shall be created. 2. Select "Create new Advice". 3. Enter the name of the advice (mandatory). 4. Define the advice type: "Before", "After returning", "After throwing", "After" or "Around" (mandatory). 5. Either refer to a named pointcut, define a new pointcut expression through selection or type in a custom pointcut expression. 6. Click on "Save".
Alternate Paths	• Create New "Class"-Pointcut 1. Choose the aspect for which the pointcut shall be created. 2. Select "Create new Pointcut". 3. Enter the name of the pointcut. 4. Choose the pointcut expression designator "Class". 5. Either directly enter the pointcut expression or choose a class name from a list. 6. Click on "Save". • Create New "Method"-Pointcut 1. Choose the aspect for which the pointcut shall be created. 2. Select "Create new Pointcut". 3. Enter the name of the pointcut. 4. Choose the pointcut expression designator "Method". 5. Optionally chose the visibility of the method ("public", "protected" or "private"). 6. Either directly enter the pointcut expression or choose a class name/ method name from a list. 7. Click on "Save".

- Create New "Within"-Pointcut
  1. Choose the aspect for which the pointcut shall be created.
  2. Select "Create new Pointcut".
  3. Enter the name of the pointcut.
  4. Choose the pointcut expression designator "Within".
  5. Either directly enter the pointcut expression or choose an interface/class name from a list.
  6. Click on "Save".

Postconditions

- The edited aspect is saved consistently, including any newly created advices or pointcuts.

Additional Requirements

B.8. Edit TypoScript 2.0

ID	UC007
Version	This use-case is planned to be implemented in version 0.3 of the FLOW3DE.
Description	This use-case covers editing TypoScript 2.0.
Actors	Editor (AC002), Developer (AC001)
Preconditions	• The package in which the TypoScript 2.0 definition is a part of must be available.
Basic Path	1. Select a package. 2. Double-click on the TypoScript 2.0 file. 3. The system opens a specialized TypoScript 2.0 editor. 4. Make all the changes needed. 5. Click "Save".
Alternate Paths	---
Postconditions	• The changed TypoScript 2.0 definitions are saved consistently.
Additional Requirements	• Syntax Highlighting (RQ017) • Content Assist (RQ016) • TypoScript 2.0 Editor (RQ015)

B.9. Create Content

This use-case will be part of a future release of the FLOW3DE. At the current state of development, it is still quite unsure what steps will be needed for this use-cases. For this reason, this use-case does not yet contain more specific requirements.

ID	UC008
Description	This use-case covers creating content (like a page, an article, etc) in the Content Repository.
Actors	Editor (AC002), Developer (AC001)
Preconditions	• The content repository must be available (either locally or remotely).
Basic Path	1. Choose the content repository. 2. Select "Create new content". 3. Choose the content type. 4. Enter the information based on the content type selected. 5. Click "Save".
Alternate Paths	---
Postconditions	• The newly created content is saved to the content repository.
Additional Requirements	• New Content Wizard (RQ009) • Custom Templates (RQ018)

B.10. Maintain Content

This use-case will be part of a future release of the FLOW3DE. At the current state of development, it is still quite unsure what steps will be needed for this use-cases. For this reason, this use-case does not yet contain more specific requirements.

ID	UC009
Description	This use-case covers creating, editing and deleting content in the Content Repository (CR).
Actors	Editor (AC002), Developer (AC001)
Preconditions	• The content repository must be available (either locally or remotely).
Basic Path	1. Select the content repository. 2. Select the content element to be edited. 3. Click "Edit". 4. Change the information in the content element. 5. Click "Save".
Alternate Paths	---
Postconditions	• The changed created content is saved to the content repository.
Additional Requirements	---

B.11. Delete Element

ID	UC010
Version	This use-case is planned to be implemented in version 0.3 of the FLOW3DE.
Description	This use-case covers the deletion of an element, such as a package, an aspect, a content element, etc.
Actors	Developer (AC001), Editor (AC002)
Preconditions	• The element to be deleted must be available.
Basic Path	1. Select the element to be deleted. 2. Choose "Delete". 3. The system asks for confirmation. 4. Click "OK".
Alternate Paths	• Abort Deletion 1. Select the element to be deleted. 2. Choose "Delete". 3. The system asks for confirmation. 4. Click "Cancel". 5. The system cancels the action without deleting the element.
Postconditions	• The element is deleted consistently from the system.
Additional Requirements	---

Appendix C. Additional Requirements

C.1. Functional Requirements

C.1.1. New Package Wizard

New Package Wizard

ID	RQ001
Description	A wizard must be provided to create a new package.
Realizing Use-Cases	• Create Package (UC001)

Mandatory Directories/Files

ID	RQ002
Description	The "New Package Wizard" must always create the directory "Meta" with the "Package.xml" file.
Realizing Use-Cases	• Create Package (UC001)

Optional Directories

ID	RQ003
Description	The "New Package Wizard" may optionally create the following directories: • Configuration • Documentation • Documentation/Manual/en_EN/ • Resources • Resources/Media • Resources/Templates • Resources/PHP • Resources/Java • Tests
Realizing Use-Cases	• Create Package (UC001)

Templates

ID	RQ004
Description	The "New Package Wizard" must optionally provide templates for various elements of a package, such as a Content Repository Design, TypoScript 2.0 snippets, etc.
Realizing Use-Cases	• Create Package (UC001)

Default Values For New Package Wizard

ID RQ031

Description For the following values, defaults must be configurable as project and/or as workspace defaults:

- Languages
- Author Name
- Author E-Mail
- Additional Folders

Realizing Use-Cases
- Create Package (UC001)

Check For Valid Package Keys

ID RQ005

Description The system must check if the package key entered by the user is valid. If the key is not valid, a warning must be issued, but the "Finish" button must not be disabled.

A valid package key must only use the characters a-z, A-Z and 0-9 and it must always start with an uppercase character.

Realizing Use-Cases
- Create Package (UC001)

Check For Valid Version Numbers

ID RQ006

Description The system must check if the version number entered by the user is valid. If the version number is not valid, a warning must be issued, but the "Finish" button must not be disabled.

Valid version numbers always contain three digits, separated by dots. No other characters are allowed. No negative digits are allowed.

Realizing Use-Cases
- Create Package (UC001)

C.1.2. New Aspect Wizard

Check For Valid Aspect Names

ID RQ007

Description The system must check if the aspect name entered by the user is valid. If the aspect name is not valid, the "Finish" button of the wizard must be disabled and the user must be notified.

Realizing Use-Cases
- Create Aspect (UC005)

New Aspect Wizard

ID RQ008

Description	A wizard must be provided to create a new aspect.
Realizing Use-Cases	• Create Aspect (UC005)

C.1.3. New Content Wizard

New Content Wizard

This requirement will be part of a future release of the FLOW3DE.

ID	RQ009
Description	A wizard must be provided to create new content elements.
Realizing Use-Cases	• Create Content (UC008)

C.1.4. PHP Editor

AOP Annotations

ID	RQ010
Description	The PHP Editor must support FLOW3-specific AOP annotations.
Realizing Use-Cases	• Edit PHP Code (UC003)

Check For CGL-Compliance

This requirement will be part of a future release of the FLOW3DE.

ID	RQ011
Description	The code entered in the PHP Editor must automatically be checked for compliance with the FLOW3 Coding Guidelines (CGL). Any problems must be marked in real-time and whenever possible a suggestion to correct the problem shall be available.
Realizing Use-Cases	• Edit PHP Code (UC003)

New Exceptions

ID	RQ012
Description	The PHP Editor must support the creation of new FLOW3 exceptions with a unique identifier (i.e. the current UNIX timestamp).
Realizing Use-Cases	• Edit PHP Code (UC003)

C.1.5. Configuration Editor

Configuration Editor

ID	RQ013

Description	An editor must be provided to make all configuration for a specific package. The editor must be able to read and edit all relevant package configuration files, such as the "Package.xml" file.
Realizing Use-Cases	• Edit Configuration (UC002)

Configuration Editor GUI

ID	RQ014
Description	The GUI of the Configuration Editor shall follow the patterns used in the PDE (Plugin Development Environment) of Eclipse.
Realizing Use-Cases	• Edit Configuration (UC002)

C.1.6. TypoScript 2.0 Editor

TypoScript 2.0 Editor

ID	RQ015
Description	An editor for TypoScript 2.0 must be provided.
Realizing Use-Cases	• Edit TypoScript 2.0 (UC007)

Content Assist

ID	RQ016
Description	The TypoScript 2.0 editor must provide content assist ("auto completion"). The content assist must recognize the currently installed packages and account for the TypoScript elements provided by these packages.
Realizing Use-Cases	• Edit TypoScript 2.0 (UC007)

Syntax Highlighting

ID	RQ017
Description	The TypoScript 2.0 editor must provide syntax highlighting.
Realizing Use-Cases	• Edit TypoScript 2.0 (UC007)

C.2. Non-Functional Requirements

C.2.1. Usability

Custom Templates

ID	RQ018
Description	It must be possible to provide custom templates for any of the wizards that offer templates.
Realizing Use-Cases	• Create Package (UC001) • Create Aspect (UC005)

- Create Content (UC008)

File-System Settings

This requirement will be part of a future release of the FLOW3DE.

ID	RQ019
Description	The settings of the local or remote file-system (and the connection properties for the remote file-system) must be made on project level. It shall be possible to define default settings on the workspace level.
Realizing Use-Cases	---

Local Or Remote File-System

This requirement will be part of a future release of the FLOW3DE.

ID	RQ020
Description	It must be possible to either work on the local or any remote file system.
Realizing Use-Cases	---

Multi-Language

ID	RQ021
Description	The system's GUI must support multiple languages.
Realizing Use-Cases	---

Online Documentation

ID	RQ022
Description	The system must provide an online documentation integrated into the Eclipse help system.
Realizing Use-Cases	---

Shortcuts

ID	RQ023
Description	All frequently used features of the system shall be accessible through keyboard shortcuts, without using the mouse.
Realizing Use-Cases	---

C.2.2. Reliability

Error Logs

ID	RQ024

Description	Upon system failures, the system shall produce error logs with all the debugging information needed to located and reproduce the error situation.
Realizing Use-Cases	---

Wrong Or Insufficient Input

ID	RQ025
Description	Upon wrong or insufficient user input, the system shall provide a concise error message including suggestions on how to solve the problem.
Realizing Use-Cases	---

C.2.3. Performance

Access Remote File System

ID	RQ026
Description	Accessing the remote file system shall take no longer than one second per request on average.
Realizing Use-Cases	---

Lazy Initialization

ID	RQ027
Description	The system must adhere to the general concept of "Lazy Initialization" of Eclipse to avoid long startup times.
Realizing Use-Cases	---

C.2.4. Supportability

Coding Guidelines

ID	RQ028
Description	The system shall strictly adhere to the Eclipse Coding Guidelines.
Realizing Use-Cases	---

Unit-Tests

ID	RQ029
Description	The system's main functionality must be tested by automated unit-tests. The code coverage shall be no less than 75 percent.
Realizing Use-Cases	---

Update Site

ID	RQ030

Description	The system must provide an update mechanism using an Eclipse Update Site.
Realizing Use-Cases	---

Appendix D. Glossary

Advice	An advice is the action taken by an aspect at a particular join point. Advices are implemented as methods of the aspect class. These methods are executed before and / or after the join point is reached.
Advice Chain	If more than one around advice exists for a join point, they are called in an onion-like advice chain: The first around advice probably executes some before-code, then calls the second around advice which calls the target method. The target method returns a result which can be modified by the second around advice, is returned to the first around advice which finally returns the result to the initiator of the method call. Any around advice may decide to proceed or break the chain and modify results if necessary.
After Advice	An after advice is executed after the target method has been called, no matter if an exception was thrown or not.
After Returning Advice	An after returning advice is executed after returning from the target method. The result of the target method invocation is available to the after returning advice, but it can't change it. If the target method throws an exception, the after returning advice is not executed.
After Throwing Advice	An after throwing advice is only executed if the target method throwed an exception. The after throwing advice may fetch the exception type from the join point object.
Around Advice	An around advice is wrapped around the execution of the target method. It may execute code before and after the invocation of the target method and may ultimately prevent the original method from being executed at all. An around advice is also responsible for calling other around advices at the same join point and returning either the original or a modified result for the target method.
Aspect	An aspect is the part of the application which cross-cuts the core concerns of multiple objects. In FLOW3, aspects are implemented as regular classes which are tagged by the @aspect annotation. The methods of an aspect class represent advices, the properties act as an anchor for introductions.
Aspect Oriented Programming (AOP)	Aspect-Oriented Programming (AOP) is a programming paradigm which complements Object-Oriented Programming (OOP) by separating concerns of a software application to improve modularization. The separation of concerns (SoC) aims for making a software easier to maintain by grouping features and behaviour into manageable parts which all have a specific purpose and business to take care of.
Before Advice	A before advice is executed before the target method is being called, but cannot prevent the target method from being executed.
CGL	CGL stands for "Coding Guidelines". The FLOW3 CGL are available online: http://5-0.dev.typo3.org/guide/bk03pt05ch05.html There is also a package which checks FLOW3 code for CGL compliance. It is written in PHP: http://forge.typo3.org/projects/show/packages-flow3cgl
Content	In the context of a Content Management System such as TYPO3, "content" refers to any element which is visible on the website, such as an article, a news item or a calendar entry.

Content Repository (CR)	The Content Repository (CR) is the blackbox where records (pages, news items) and files are stored. From outside it looks like a kind of object database - internally it's based on relational databases like MySQL and the file system.
FLOW3	FLOW3 is an application framework written in PHP. It stems from the TYPO3 community. The version 5.0 of TYPO3 is going to be a complete rewrite, based on a new architecture. The foundation of this new version is going to be FLOW3.
FLOW3 Development Environment (FLOW3DE)	Eclipse plugin for the development and maintenance of FLOW3 packages.
Introduction	An introduction redeclares the target class to implement an additional interface. By declaring an introduction it is possible to introduce new interfaces and an implementation of the required methods without touching the code of the original class.
Join Point	A join point is a point in the flow of a program. Examples are the execution of a method or the throw of an exception. In FLOW3, join points are represented by the T3_FLOW3_AOPJoinPoint object which contains more information about the circumstances like name of the called method, the passed arguments or type of the exception thrown. A join point is an event which occurs during the program flow, not a definition which defines that point.
Package	FLOW3 is a package-based system. Packages act as a container for many different purposes: Most of them contain PHP code which adds certain functionality, others only contain documentation and yet other packages consist of templates, images or other resources.
Pointcut	The pointcut defines a set of joinpoints which need to be matched before running an advice. The pointcut is configured by a pointcut expression which defines when and where an advice should be executed. FLOW3 uses methods in an aspect class as anchors for pointcut declarations.
Pointcut expression	A poincut expression is the condition under which a joinpoint should match. It may, for example, define that joinpoints only match on the execution of a (target-) method with a certain name. Pointcut expressions are used in pointcut- and advice declarations.
Target	A class or method being adviced by one or more aspects is referred to as a target class /-method.
TypoScript	TypoScript is a configuration language of TYPO3. In version 2.0, TypoScript has become fully object oriented and is now part of FLOW3.
Web Services Eclipse File System (WSEFS)	The WSEFS provides an abstracted client-server based file system integrated into Eclipse. It was developed as a master thesis by Andreas Tschirpke at the University of Liverpool.