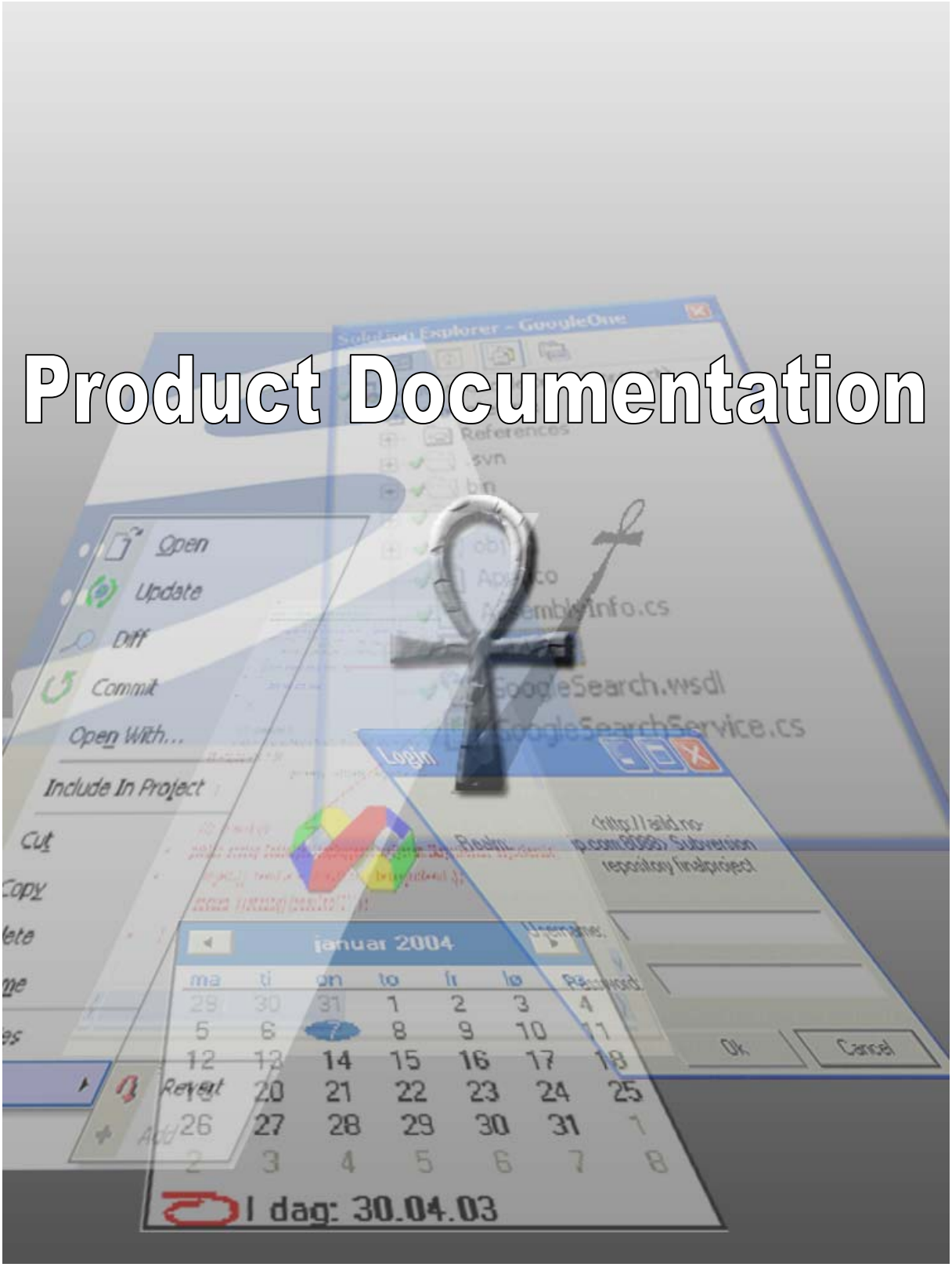
Product Documentation



Preface

This is a project report on Ankh, a Visual Studio .NET addin for the Subversion revision control system, which was carried out in the spring of 2003.

The development of Ankh was conducted as a final year project at HiO in cooperation with CollabNet. The system contains an addin/GUI layer written in the programming language C#, and a .NET wrapper layer divided in a low level part interfacing with SVN's client API coded in Managed C++ and an object-oriented part coded in C#.

This report is intended for computer personnel responsible for marketing, maintaining, updating, supporting and modifying the system. The purpose of this document is to give a description of:

- System design
- Installation
- Configuration

The code is documented in detail in an html representation of the source code available on the enclosed CD.

The test report describes in detail the various system tests.

Alternative approaches as well as challenges in the development process and reasons for the selected GUI are described in the process report.

A prior knowledge of revision control systems, preferably Subversion and VS.NET, is an advantage for obtaining a full understanding of this report. A dictionary enclosed in the appendix covers basic terminology.

Table of contents

1 Introduction

The introduction describes the background of the project, why Ankh should be used, how the program works and the navigation model.

1.1 Background

The background describes the aim of the project, the revision control system Subversion and the external employer CollabNet that funds the development of Subversion itself.

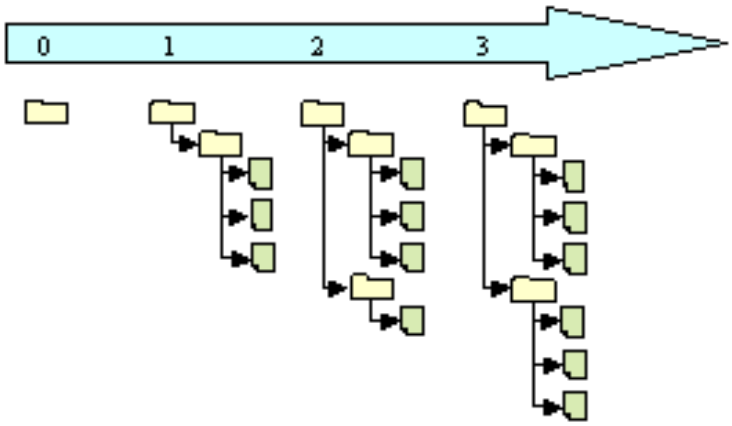
1.1.1 CollabNet

CollabNet is a California-based company that in the year 2000 provided the initial funding to begin development work on Subversion and which currently has several paid developers working on the Subversion project. CollabNet offers a wide range of products and services to improve software quality and accelerate time-to-market during product development. Their main product is SourceCast, which is a Web-based software development platform. Subversion is aimed to be the successor to an existing revision control system called CVS (Concurrent Versions System) and CollabNet intends to make it a part of the SourceCast platform.

1.1.2 Subversion

Subversion is a free open source revision control system. In a typical software development environment, many developers will be engaged in work on one code base. Revision control makes it easier for several people to share the same collection of files and folders in a code base. A revision control system is a centralised system for sharing information where the core is a repository.

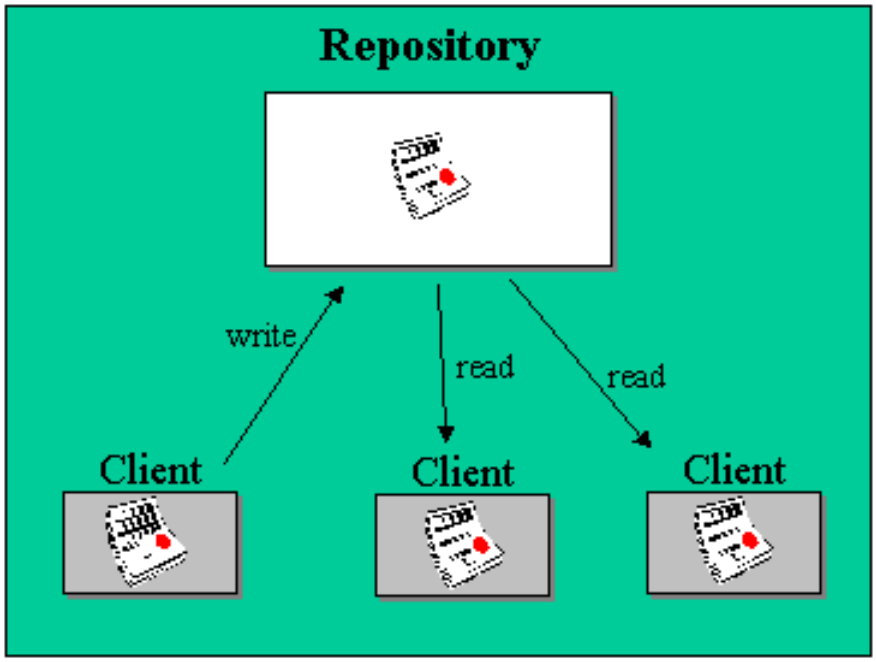
A **repository** is much like an ordinary file server, except that it remembers every change ever made to your files. In the figure below yellow boxes indicate folders and green boxes indicate files. The numbers on the blue arrow indicate version numbers.



Each individual programmer checks out a working copy from the repository. A **working copy** is a collection of files and folders under revision control. Each file tree in the figure above is an example of a working copy that can be checked out from the repository. During the development work cycle these working copies are updated and synchronised with the repository.

When a commit is performed your changes are sent from your working copy to the repository. A **commit** publishes, or makes your changes available, to your colleagues. When a commit is performed it is possible to write a log message that usually contain a short description of changes performed.

Using revision control makes it possible to get an amazing overview of the work performed at any time from anywhere. If one project member has a problem that is difficult to solve another project member can easily get information on what was performed and then support the other project member in solving the problem.



Subversion has a number of options. An attempt to analyse these options from the user's point of view is done in a mental model in the process report. The Ankh integration of the SVN options are analysed from the developer's point of view in a conceptual model in this report. Additional information can be obtained from http://subversion.tigris.org.

1.2 Aim

The aim of the Ankh project was to provide a level of integration for the Subversion version control system into the integrated development environment Visual Studio .NET. By making an addin for Visual Studio .NET, the revision control system is

available from inside Visual Studio, which is a clear improvement for many developers compared to the today's command prompt. This will also make Subversion available for a huge user base, users who might not even consider it otherwise.

The .NET wrapper layer is available as a standalone component so that it can easily be used in other .NET applications in the future.

1.3 Why use Ankh?

Ankh is a free open source tool that integrates the revision control system Subversion into VS.NET. Today there are several revision control systems available as an addin to VS.NET on the market, but none of them are free.

Subversion is the successor to a widely used revision control system named CVS. Subversion was initially founded to improve CVS. The initial goal of Subversion was to preserve the same functionality as CVS while fixing the most obvious flaws.

1.4 Navigation model and menus

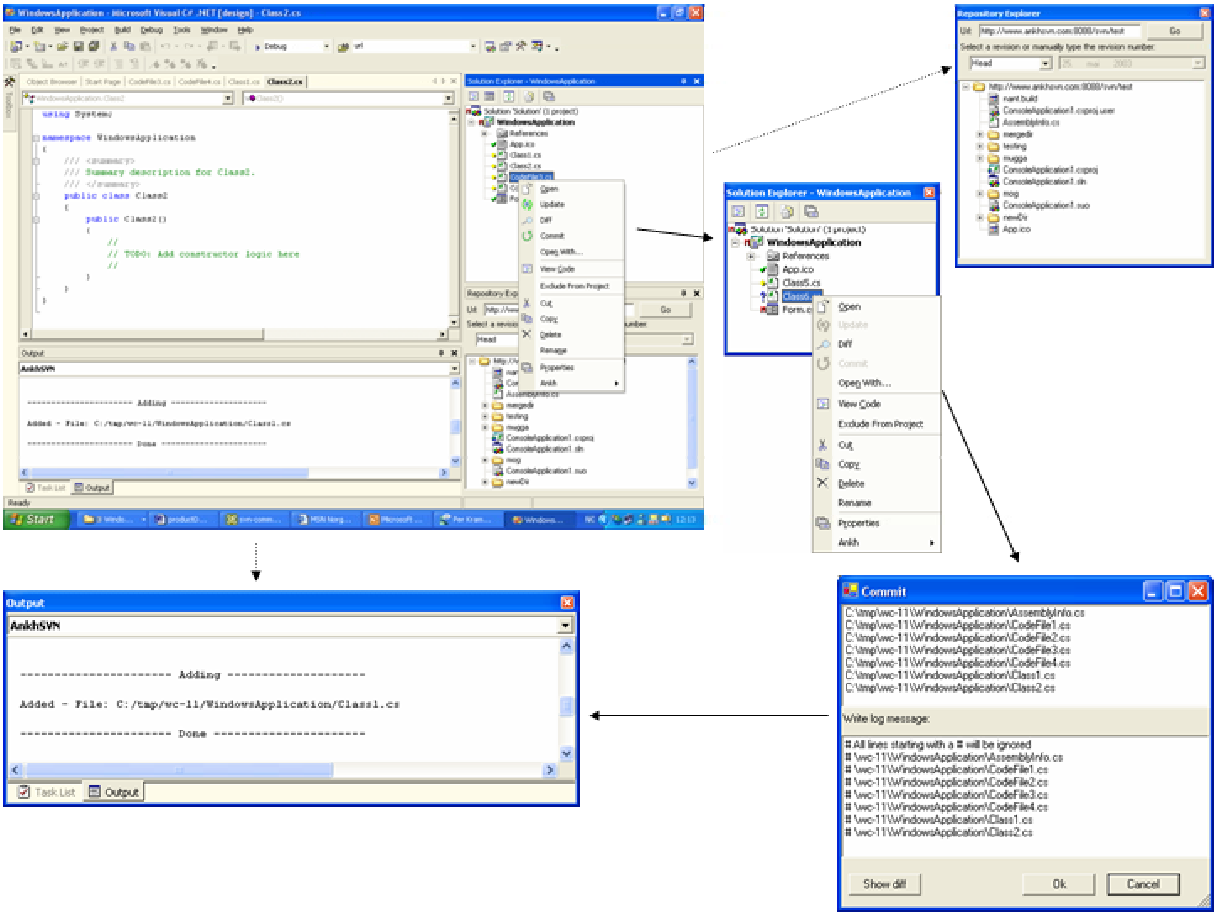
The integration of Ankh in VS.NET is done as smoothly as possible to avoid distracting the programmer. Ankh is hooked up to menus, output panes etc. that already exists in VS.NET therefore colours, dialog boxes, file tree view, menus and output windows in Ankh have almost the same design as used in VS.NET.

The following window types are used:

- multisquared windows (Windows where a special event in one window effects the output in another window.)
- multiwindow (Each window could be moved independently.)

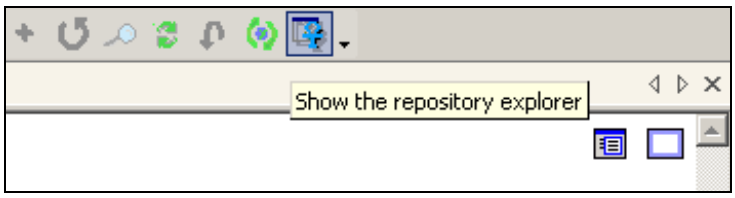
The menu choices are meant to be as functional and obvious as possible. For example the command "svn commit" is named "Commit" in the Ankh context menu. There might be some programmers that already are familiar with SVN, so the wording used in Ankh doesn't deviate considerably from the terminology used for the SVN command line client. We have deliberately avoided introducing new terms.

Our navigation model is described in the figure below. If the programmer right clicks on a file, directory, project or solution in Solution Explorer a menu will pop up. If the command requires further input, a dialog box will be displayed. The result of the action initiated from the menu or dialog box will be displayed in the AnkhSVN pane in the Output view of VS.NET. Ankh also supplies a tool window called the Repository Explorer, which allows a developer to browse the contents of a Subversion repository.



We have developed a set of icons that makes it easier to distinguish Ankh's menu options from the ones native to VS.NET itself.

Ankh provides tool tips on every menu option and on all text fields, combo boxes and buttons.



1.5 What can Ankh do for you?

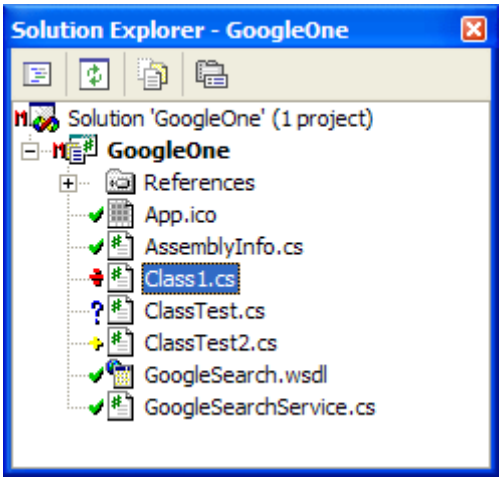
When we started to integrate Ankh into VS.NET we prioritised those options that were most frequently used on a day-to-day basis. These include the Subversion commands add, update, delete/remove, commit, diff, status, list and revert. We aimed to get a product that could be used as soon as possible. Ankh currently contains the following functionality:

- View status icons in working copy in Solution Explorer (Status icons in Solution Explorer)
- Add files, folders, individual projects and solution (item) to the working copy (Add)
- Update working copy item to include changes from last revision (Update)
- Examine changes of item in working copy (Diff)
- Revert changes in item in working copy (Revert)
- Delete/remove item from working copy (Delete)
- Commit changes to the repository (Commit)
- Show file tree view of repository in Repository Explorer (Repository Explorer)
- Authenticate the user if the repository used is restricted

Ankh primarily supports Visual Basic (VB), Visual C# (VC#) and Visual C++ (VC++) projects. Support for VC++ projects is weaker than VC# and VB projects, due to the fact that the automation model exposed by VS.NET is different for these. Future versions of Ankh will have better support for VC++ projects.

Status icons in Solution Explorer

Shows status of working copy files, folders, projects and solutions (item) in Solution Explorer.

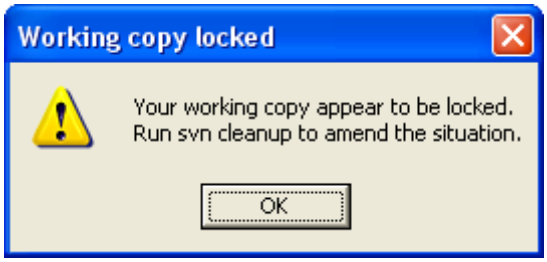


The meanings of the different status icons are listed below:

- Unversioned item: Item not under revision control.
- Versioned item: Item under revision control.
- Modified versioned item.
- Added and now versioned item.
- Deleted versioned item.
- Conflicted versioned item: If two users modify the section of the same file, one of the users will get a conflict when he updates his working copy.
- Locked versioned item: If a Subversion operation is interrupted for some reason, the working copy might be left in a locked state. This icon indicates that certain files are locked.

If an item is conflicted, the conflict has to be resolved. Currently this requires the user to run `svn resolve` from the command line. Future versions will support Resolve functionality from within the IDE.

If an item is locked a cleanup is required. Ankh gives the following warning in case of a lock:

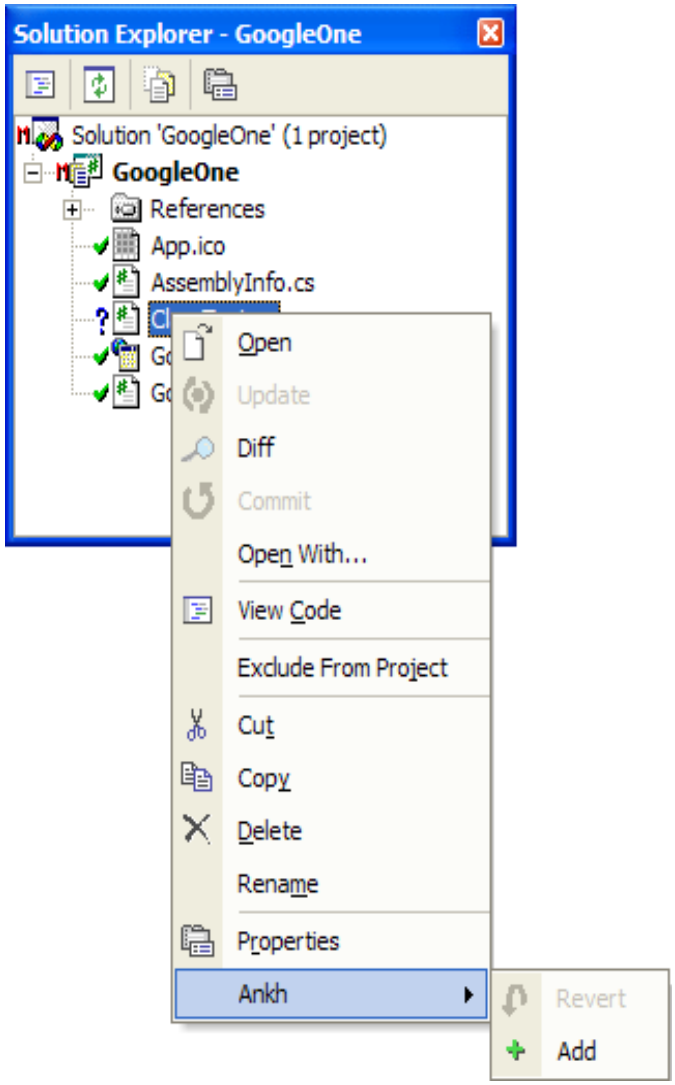


Ankh does not yet support performing cleanups from within the IDE.

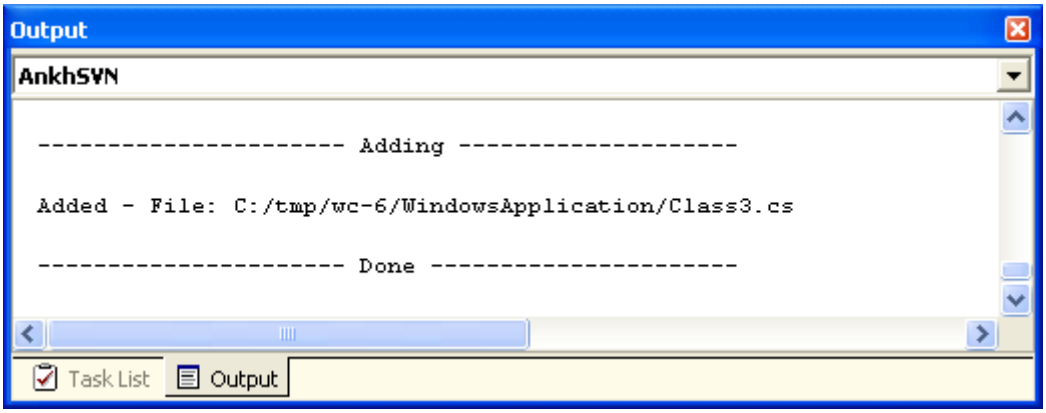
Ankh updates the status icons in Solution Explorer automatically after the status has changed on item(s). There is also an option called Refresh available on the Ankh submenu, which allows the user to refresh a solution, project or folder manually, in case Ankh fails to do so.

Add

The Add command adds items to your working copy and schedules them for addition to the repository. These items will be added to the repository in your next commit. If you add an item and change your mind before committing, you can unschedule the addition using Revert. Add works on both files and folders. The Add option is available from the item menu, which is displayed if an item is selected in Solution Explorer and right clicked.



Output from the Add command is displayed in the AnkhSVN output pane, as shown below.



Update

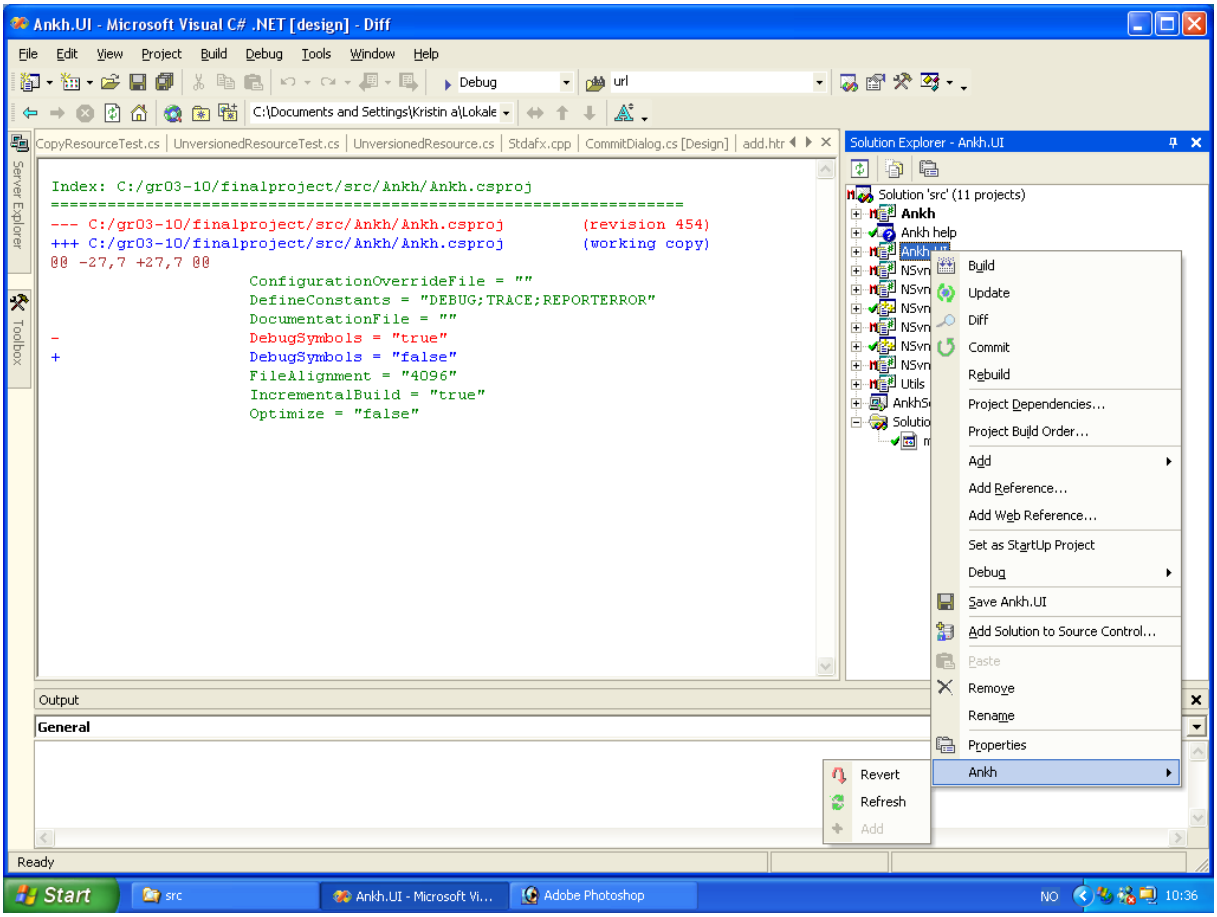
The Update command brings changes from the repository into your working copy. It brings your working copy up-to-date with the head revision (last revision in repository). Update can be performed on several items simultaneously and works recursively on folders, individual projects and the solution. The Update command is

available from the context menu for any item selected in the Solution Explorer. Output from the Update command is displayed in the AnkhSVN output pane.

.

Diff

The Diff command displays differences between two versions of the same item(s). Ankh compares the base revision (last updated or committed revision) and the working copy. Diffs are displayed using VS.NET's integrated web browser.



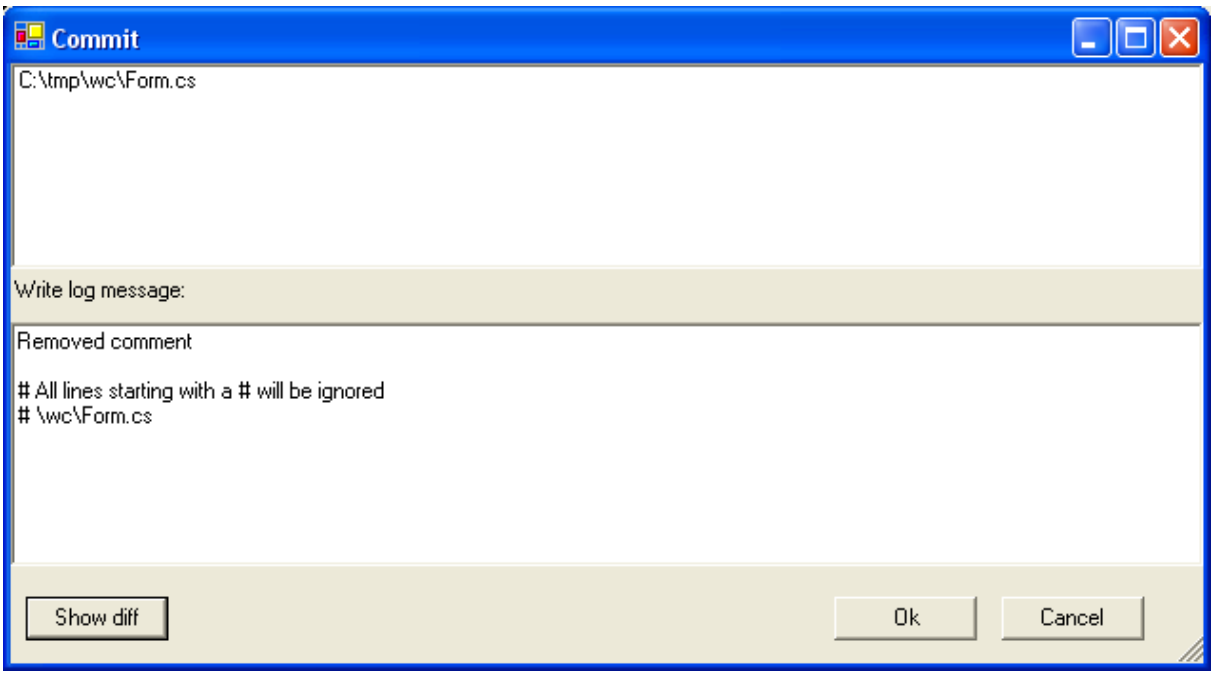
Red colour indicates text removed, green text indicates unchanged text and blue indicates new text added. Diff can be performed on several items and works recursively on folders, individual projects and the solution. The Diff command is available from the context menu of any item in the Solution Explorer.

Revert

The Revert command reverts all local changes performed since the last update or commit. Revert can be performed on several items and works recursively on folders, individual projects and the solution. The Revert command is available from the context menu of any item in the Solution Explorer. Output from the Revert command is displayed in the AnkhSVN output pane.

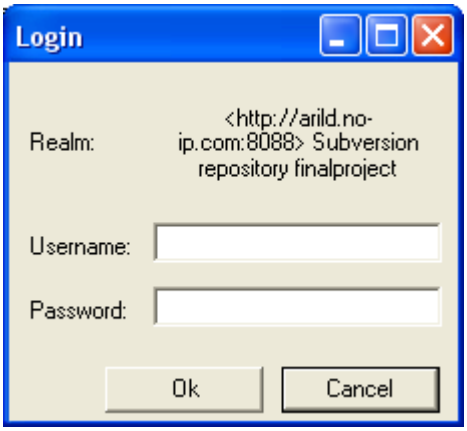
Delete/Remove
When a user deletes a file or folder from within VS.NET, the svn delete operation is also invoked on the item. This prevents the working copy from getting out of sync with VS. NET's Solution Explorer. Output from delete/remove operations is displayed in the AnkhSVN output pane.

Commit
The Commit command sends changes from your working copy to the repository. A commit operation can publish changes to any number of items. In your working copy, you can change files, contents, create, delete and then commit the complete set of changes as an atomic unit. It is possible to examine the items' changes (Diff) in the working copy from within the commit dialog. Commit works recursively on folders, individual projects and the solution. The Commit option is available from the context menu of any item in the Solution Explorer. Upon invoking the commit operation, a dialog appears in which the user enters a log message.



The upper text area displays the items that are to be committed, while the lower area is used for composing the log message that is to accompany the commit. All lines starting with a # will be ignored in the log message sent to the repository. The developer can see all changes made since the last commit by pressing the Diff button.

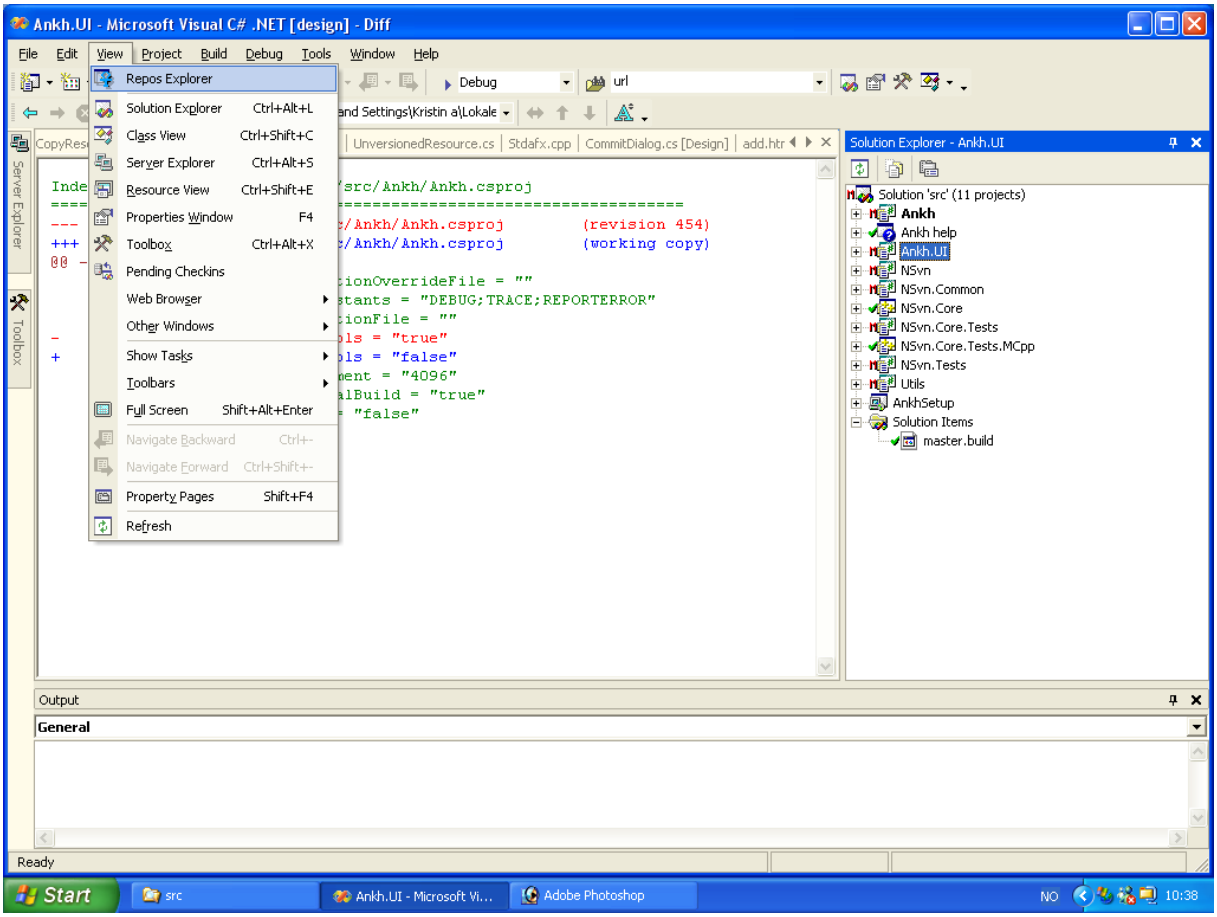
If the repository requires authentication for the commit the Login dialog will be displayed.



The Commit action initiated is confirmed in the AnkhSVN output pane.

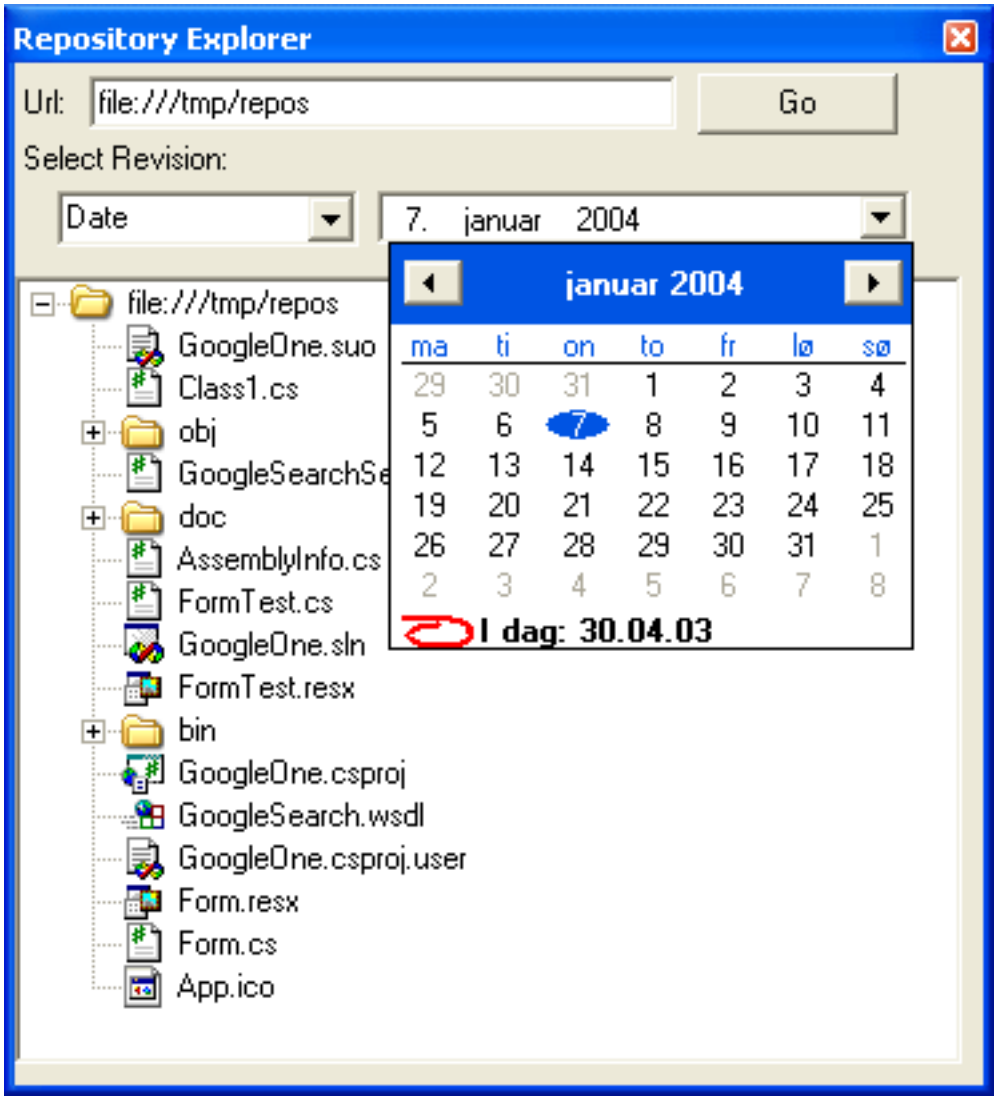
Repository Explorer
The Repository Explorer displays a file tree view of repository with file-type icons corresponding to those used by VS .NET for a valid URL and revision.

The Repository Explorer is available from View → Repos Explorer from the main menu bar in VS.NET.



The Repository Explorer can display any revision in the repository, with the revision given as head, as specific date or a revision number. The head revision is the latest revision present in the repository. If Date is selected in the dropdown the date-picker

is enabled and a calendar is available. Below is the Repository Explorer with date revision selected. The calendar is available.



Future versions of Ankh will support a large number of operations in the Repository Explorer, such as repository-side copies, moves (which forms the basis for tagging and branching operations in Subversion) and merging.

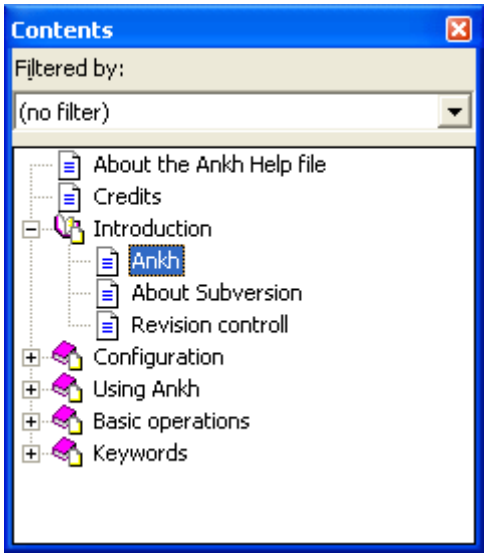
1.6 How does Ankh work?

This section is covered in Ankh Help integration.

1.7 Ankh Help

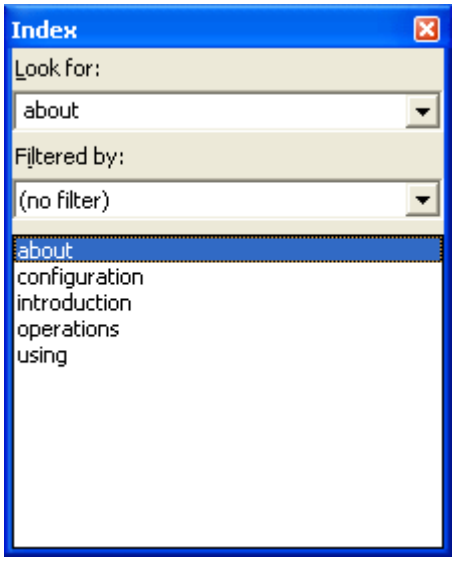
Ankh Help was meant to be a user guide integrated in VS.NET available from the main menu. The Ankh Help integration covers the basic Ankh operations. Some keywords are included in the Index view. The Index view works as a simple search where the search words are listed in the Index view.

In the Contents view you can see the contents of the Ankh Help. If a book icon is clicked twice the folder symbolised by a book icon will open and you can see the content of the “book”. In the example below Introduction is clicked twice and it’s content is displayed.

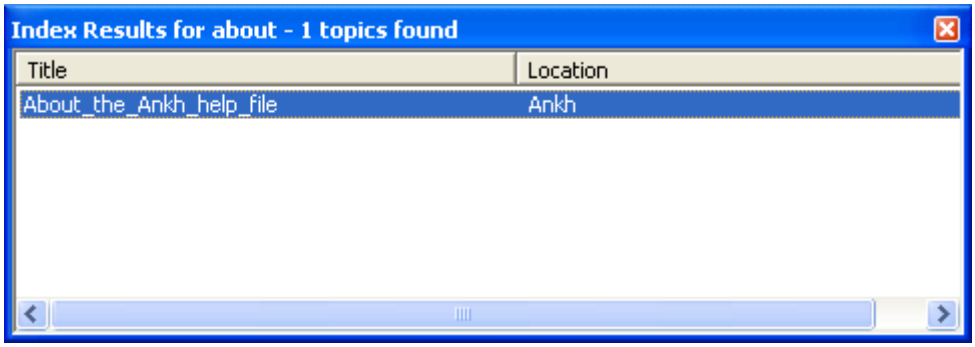


If “Ankh” is selected in the list of Contents the content of “Ankh” is displayed:

The Index view works as a simple search where the search words are listed in the Index view. We did not have time to enter all possible searching words into a base. The Index view gives you the possible searching words available. The articles connected to each search are listed in the Search view. In the figure shown below the keyword “about” is selected in the Index view.



Results of the selected keyword “about” is visible in the Index Results view:



The included CD has a version of the Help documentation optimised for web.

2 The Ankh project overview

This chapter gives an overview of the Ankh project. Topics covered include system architecture, conceptual models, use cases, testing and future developments.

2.1 Development tools

This subsection describes the development tools used during the development of Ankh.

2.1.1 Visual Studio .NET

Visual Studio .NET is a development environment for several coding languages. Visual Basic .NET (VB), Visual C++ .NET (VC++), and Visual C# .NET (VC#) all use the same integrated development environment (IDE), which allows them to share tools and facilities in the creation of mixed-language solutions. In addition, these languages leverage the functionality of the .NET Framework, which provides access to key technologies that simplify the development of ASP.NET Web applications and XML Web services.

2.1.2 VS.NET Addin architecture

Visual Studio NET provides an open development model that allows developers to extend the functionality of the IDE through special components called addins. Since VS.NET itself is implemented in COM, addins also need to be implemented using COM. Fortunately, .NET allows you to expose .NET assemblies as COM components very easily and that is what we did during the development of Ankh. VS.NET also allows you to develop addins using C++ and the Active Template Library (ATL), but this is a lot harder, while it doesn't give any significant benefits over writing it in C#.

2.1.3 NUnit

NUnit is a free open source unit-testing framework for all .Net languages. It was initially ported from JUnit, which is a unit-testing tool for Java(and JUnit itself was originally ported from Smalltalk…). We used NUnit 2.0 during the development of Ankh to unit test the classes in the NSvn wrapper layer.

2.1.4 NAnt

NAnt is a free .NET build tool, and serves the same purpose as `make` in traditional environments. Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed. The various tasks include compiling, linking, copying files, creating zip files and cleaning the source tree from files created by the build process. NAnt is used to generate regular builds of Ankh and NSvn that don't rely on having VS.NET installed on the build computer.

As a proof of concept, we also developed a NAnt svncheckout task using the NSvn libraries. This task checks out a source tree from a Subversion repository. The task below checks out the source code for the svncheckout task itself:

```
<svncheckout localDir="C:\tmp"
url="http://www.ankhsvn.com:8088/svn/finalproject/trunk/tools/SvnTasks" />
```

2.1.5 C#

Microsoft C# (pronounced C sharp) is a new programming language designed for building a wide range of enterprise applications that run on the .NET Framework. C# code is compiled as managed code, which means it benefits from the services of the common language runtime. These services include language interoperability, garbage collection, enhanced security, and improved versioning support. C# was used for the main parts of Ankh except from the lower level part of the wrapper.

2.1.6 Managed C++

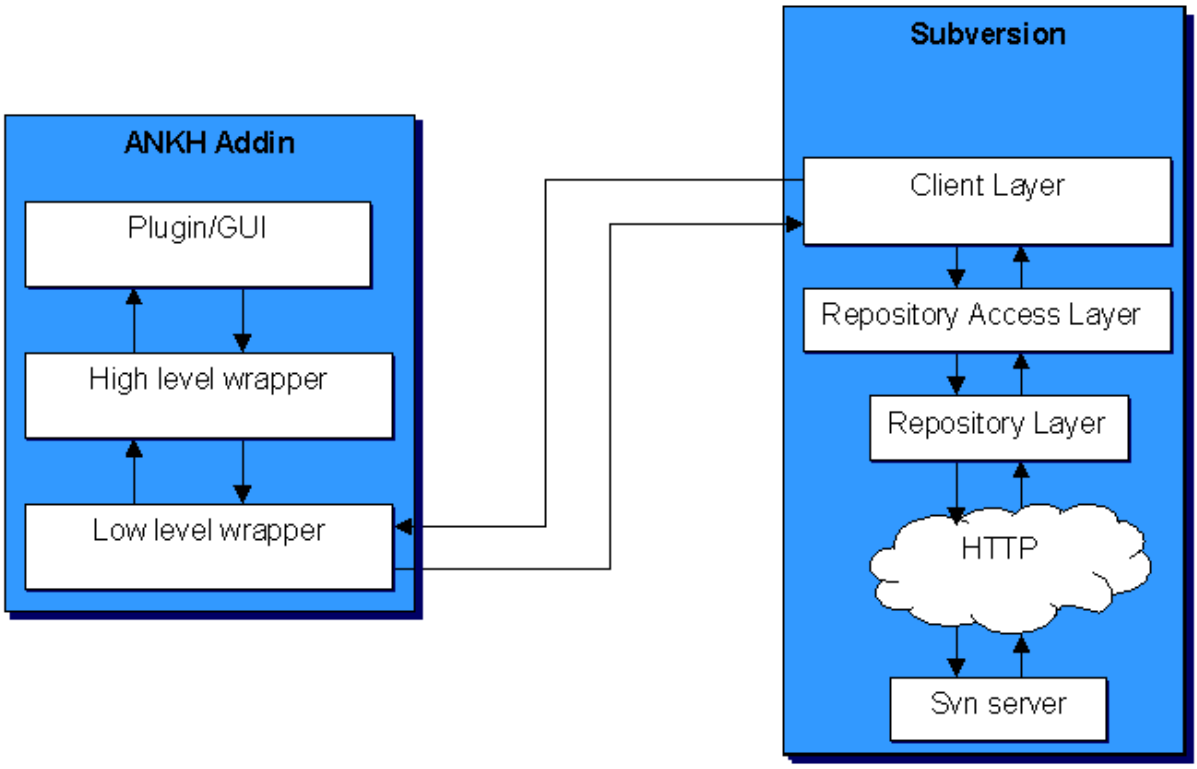
Managed Extensions for C++ are enhancements made to the C++ language, giving C++ full access to .NET functionality. Managed Extensions simplify the task of migrating existing C++ applications to the new .NET Framework. Managed C++ is used to interface with the Subversion C client API and present it to the managed world.

2.1.7 EA

Enterprise Architect is a project oriented UML modelling tool for the Windows platform. This tool is used to generate the use case diagrams and class diagrams.

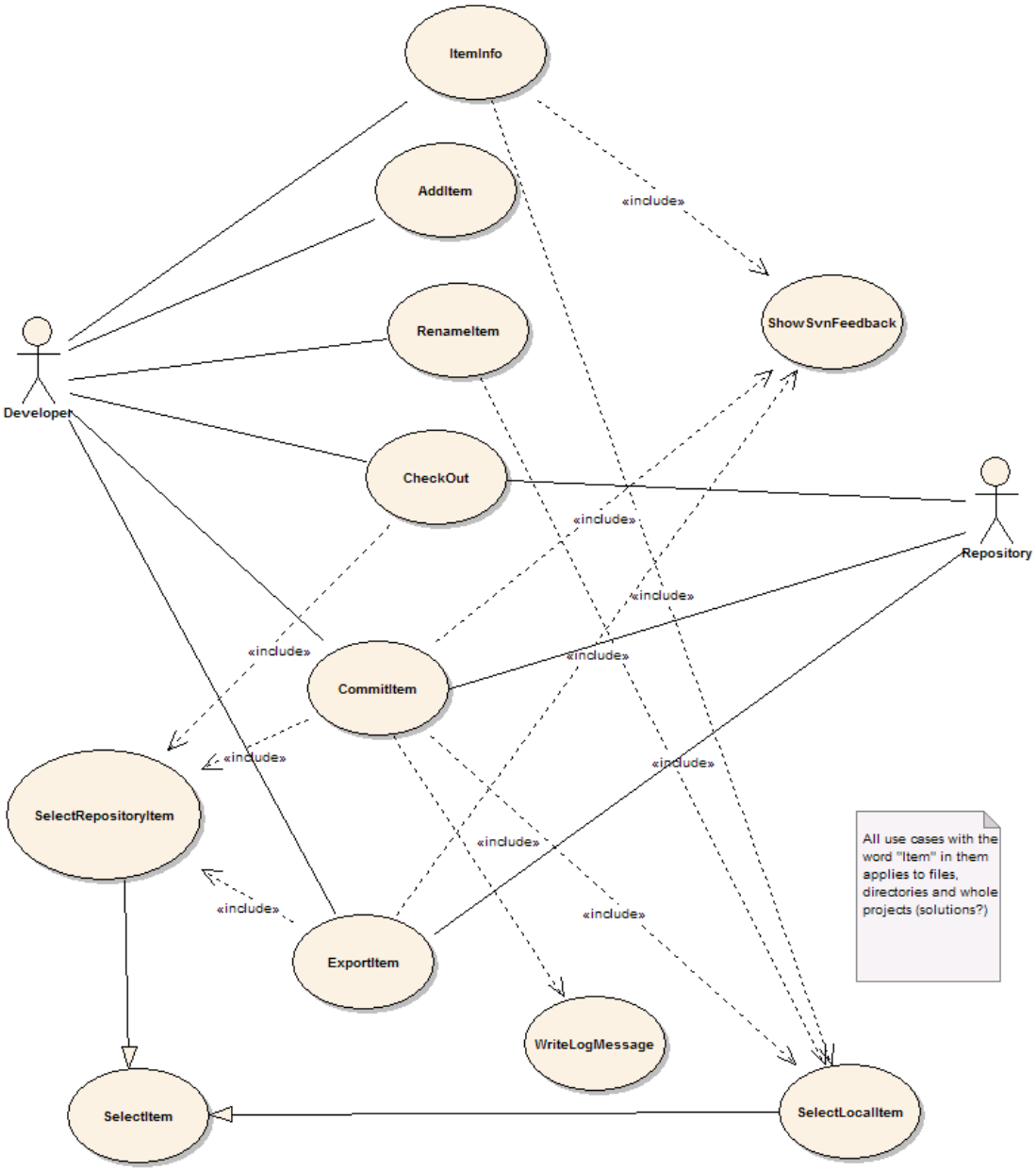
2.2 System architecture

The system is divided into two main layers, a GUI/addin layer and a .NET wrapper layer. The GUI/addin layer contains the code for the addin itself and it’s graphical user interface.This layer is called Ankh, and is written in C#. The wrapper is divided into a lower level part for the SVN's client API written in Managed C++ and a more object-oriented high level part coded in C#. The .NET wrapper layer is called NSvn.



2.3 Conceptual model

The system was analysed from the developer's point of view and a conceptual model is summed up in appendix 6.2 as a complete collection of use case diagrams. One example of the use case diagrams is shown below to better illustrate the text. Two actors were identified, the potential user of the system and the repository. The bubbles represent the most important use cases identified and the lines indicate the relations. Most of the use cases are reflected as menu options in Ankh.


ItemInfo
AddItem
«include»
RenameItem
ShowSvnFeedback
Developer
CheckOut
Repository
«include»
«include»
«include»
«include»
CommitItem
«include»
SelectRepositoryItem
«include»
All use cases with the word "Item" in them applies to files, directories and whole projects (solutions?)
«include»
«include»
«include»
ExportItem
WriteLogMessage
SelectItem
SelectLocalItem

2.4 Use cases

Each object identified in the conceptual model was analysed as a use case. One example of a use case is listed below. Actors, preconditions, scenario and post conditions were identified and analysed. The complete set of use cases can be found in appendix 6.3.

Use case	Add item
Actors	1 Developer
Preconditions	
1 The item is not currently in the workspace 2 The item is in a directory that is currently in the workspace	
Scenario	
1 The user invokes the VS.NET command File.AddNewItem or File.AddExistingItem 2 The item is added to the working copy 3 <Include 'ShowSvnFeedback'>	
Postconditions	
1 The item is in the workspace	

The use cases currently implemented in Ankh are :

- Add Item
- Commit Item
- Diff Local Item
- Remove Item
- Revert Item
- Select Item
- Select Local Item
- Select Repository Item
- Show Local Changes
- Show SVN feedback
- Update Item
- View Item Status
- Write Log Message

2.5 Programming guidelines

This subchapter describe guidelines used while developing Ankh.

2.5.1 Naming guidelines

Ankh follows the Microsoft .NET class library naming guidelines. A summary of these guidelines is listed below:

2.5.1.1 General

Hungarian notation, eg lpszFooBar is not used

Underscores are not used

2.5.1.2 Classes

Class names use Pascal case, like:

```
public class FooBar{...}
```

2.5.1.3 Interfaces

Interface names use Pascal Case

The name of an interface starts with the letter I

2.5.1.4 Methods

Methods are named using Pascal Case

2.5.1.5 Properties

Properties are named using Pascal Case

It is permitted (and even encouraged) to name a property the same as the underlying type:

```
public Color Color
{
   get{ return this.color; }
}
```

2.5.1.6 Attributes

Attributes are named with the Attribute-suffix added, as in FooBarAttribute.

Attributes suffix are not used when **applying** an attribute(in C# this suffix is optional):

```
//yes
[FooBar]
public void Foo()
{...}

//no
[FooBarAttribute]
public void Foo()
{...}
```

2.5.1.7 Method parameters

Parameters are named in camel case

2.5.1.8 Fields

Private fields are named using camel case

2.5.2 Formatting

Tab characters are not used for indentation.

Each nested block is indented 4 spaces relative to the enclosing block

The 'normal' indentation style are used:

```
//yes
public void FooBar()
{
    //...
}

//no
public void FooBar(){
    //...
}

//no
public void FooBar()
    {
    //...
    }

//no
public void FooBar()
    {
        //...
    }
```

All use of member fields are prefixed with `this.`:

```
public class Foo
{
    //...
```

```
    public void Bar()
    {
        this.baz = 42; //yes
        baz = 13; //no
    }

    private int baz;
}
```

Members of a class are ordered according to their protection level, in the following order from the top:

- public member methods
- public properties
- public fields(should be avoided as a general rule)
- protected member methods
- protected properties
- protected fields(should be avoided)
- private member methods
- private properties
- private fields

2.5.3 Comments

Comments in the Ankh code should in general only add information that is not obvious from reading the code alone. We have tried to choose variable and method names that are as self explanatory as possible to eliminate redundant commenting. In general, end-of-line comments are avoided unless absolutely necessary. The only exception is while making method/function calls that takes a large number of arguments. In that case it is accepted to put each argument on a separate line and add a comment describing the parameter at the end of the line.

All classes and methods that are part of a public API have an extensive documentation comment. These comments are used to generate documentation about parameters, return value, potential exceptions that are thrown by the method and a verbose summary.

For entities that are not a part of a public API (protected/private methods, classes for internal use etc.) the comments are less stringent or formal. The main objective of the comments has been to provide enough information so that others can discern what a method does and what its parameters are for.

2.5.4 Namespaces

The source tree contains more than 100 files. Class diagrams of the different namespaces used are attached in appendix 6.4. The following sections describe the purpose of each namespace.

2.5.4.1 Ankh

This namespace contains the bulk of the code relating to Visual Studio .NET integration.

2.5.4.2 Ankh.Commands

This namespace contains classes that implement the ICommand interface, which contains two important methods: QueryStatus and Execute. QueryStatus is called by VS.NET to check whether the command should be enabled or disabled. If the command is executed, the Execute method is called. In general, each ICommand implementation corresponds to a specific use case.

2.5.4.3 Ankh.EventSinks

Contains the code that handles the events that are fired by the VS.NET automation model. Examples include files being added or removed.

2.5.4.4 Ankh.Solution

Contains code pertaining to the integration with the VS.NET Solution Explorer.

2.5.4.5 Ankh.UI

The purpose of the Ankh.UI namespace is to gather all the classes relating to the graphical user interface. The classes mainly implement dialog boxes and the user controls that are used to build these.

2.5.4.6 NSvn

The NSvn namespace contains the high-level, object-oriented interface to the Subversion API.

2.5.4.7 NSvn.Common

NSvn.Common contains common functionality that code in both the NSvn.Core and NSvn namespaces rely on.

2.5.4.8 NSvn.Core

NSvn.Core contains the lower level part of the wrapper, written in Managed C++.

2.5.4.9 NSvn.Core.Tests

NSvn.Core.Tests contains unit tests for NSvn.Core, written in C#

2.5.4.10 NSvn.Core.Tests.MCpp

NSvn.Core.Tests also contains unit tests for NSvn.Core, but written in Managed C++. The need for two sets of tests for this assembly is due to the fact that some aspects of the functionality of NSvn.Core cannot be tested from a managed language.

2.5.4.11 Utils

The Utils namespace contains general helper classes that didn't fit in anywhere else. It also contains declarations for Win32 API functions.

2.5.5 Short system description

This subchapter gives a rough description of the system. The code is documented in detail in an html representation of the source code available on the enclosed CD and in the enclosed class diagrams.

2.5.6 NSvn wrapper layer

The wrapper contains a complete translation of the Subversion Client API coded in C. The wrapper is divided in 2 levels. A lower level part written in Managed C++ and a more object-oriented higher-level part written in C#. The NSvn::Core::Client class contains static methods that map directly to the svn_client_* functions in the Subversion C API. The NUnit test suite extensively tests all these methods. An example of a unit-test is described in the test report.
Above NSvn.Core resides a higher level API that attempts to abstract the idea of a working copy or a repository into classes. Examples of these classes include WorkingCopyDirectory, WorkingCopyFile, RepositoryFile and RepositoryDirectory.

2.5.7 Addin

The GUI/addin layer represents a complete example of how to hook up to menus, make explorer panes, output panes, how to make dialog boxes and user controls coded in C# in VS.NET.

2.6 Testing

Ankh was unit-tested with NUnit and built outside VS.NET with NAnt continuously during the development phase. Ankh was tested functionally by using Ankh while developing Ankh. Feedback from these tests gave useful information of what to be improved and ideas for future development. An early paper prototype testing of the GUI confirmed that our navigation model was logical and intuitive and gave us useful ideas of how to improve the GUI. In the end Ankh was tested functionally with a number of tests (described in the test report) to give an indication of the robustness of Ankh. Error messages are divided into 2 categories; Ankh errors and SVN errors, that reflect two different tools under development. More details regarding testing can be found in the test report.

2.6.1 Weaknesses in Ankh

Using Ankh while developing Ankh gave us an amazing overview of the weaknesses in Ankh. Below is a list of the most important ones we found:

- The Ankh menu entries sometimes don't get registered on the first run on a clean system. Restarting VS.NET will usually fix it.
- Support for VC++ projects is weaker than support for VB and VC# projects. If a new file is added to the project through the IDE, you may need to run the Ankh Refresh option in the item menu in Solution Explorer on the project to detect the new file.
- If you delete a file, the status on the project node will not change. To be able to commit a delete, you need to change something else, then run commit on the project or solution.
- Adding a (web) form will not automatically add the corresponding .resx or code behind file. To add these files, click the Show All Files button in the solution explorer, and then add them the usual way.
- There is currently no support for checkout and import – you will need to create your repository and import your source to it using the command line tools. For details on how to use these, see http://svnbook.red-bean.com
- There is currently no support for svn rename through the IDE – Ankh might get very confused if you do so. The preferred way is to exclude the file in question from the project, run svn rename on it from the command line, and then re-add it to the project. The same goes for move operations.
- There is currently no support for editing SVN properties – this still needs to be done from the command line.
- The messages you get when updating a single file are misleading – they seem to indicate that you updated the whole directory. This is only a flaw in the message code – update itself works as expected.

2.7 Future developments

Fix Ankh weaknesses defined as issues in issue tracker at http://ankhsvn.tigris.org.

Due to limited time schedule and unforeseen challenges the following Use Cases are not implemented: Cleanup, Copy Item, Diff Repository Item, Edit Properties, Export Item, Import Solution, Item Info, Make Directory, Make patch, Merge Item, Move Item, Rename Item, Resolve File Conflict, Switch Item, View Log, View Repository File.

Deployment and integration of our user manual into VS.NET was not completed due to a limited time schedule and unforeseen challenges.

In the middle of March 2003 Microsoft announced a potential serious bug in VS.NET that affected C++ developers. The bug can occur while creating Managed C++ Class Libraries that contain a mixture of native code and intermediate language (IL).
As we are using mixed DLLs this bug can potentially affect our addin. As we had limited time available and did not experience a big impact of this bug we have not

implemented the workaround. Guidelines for how to avoid problems caused by this bug are described in the article “The mixed DLL Loading Problem”.

We initially considered whether to use a tool called SWIG instead of writing the wrapper manually. This would have allowed us to completely sidestep the DLL bug. Unfortunately, theC# module had only recently been added to SWIG and was still in a state of development during our project period and we decided against using such an immature tool.

3 Installation and configuration

Ankh will install and run on any OS that is supported by VS.NET. This currently includes Windows 2000, Windows XP and Windows Server 2003. Visual Studio .NET 2002 was used to develop Ankh. The current release of the Ankh source code builds against Subversion 0.22.2.

Ankh does not at this time run very well on VS.NET 2003. This is due to the fact that VS.NET 2003 was released late in our development cycle and we didn’t have access to a copy during development. We expect the next release of Ankh to fully support both VS.NET 2002 and VS.NET 2003

Ankh Setup.exe and the source code can be downloaded from http://ankhsvn.tigris.org or the enclosed CD.

The installer includes a test repository that is optionally extracted to the computers local hard drive during installation. The users who download Ankh are encouraged to utilize this repository to test Ankh.

4 How to build Ankh from source code

In order to build Ankh, you will also need to build Subversion itself. Start the process by building your own copy of the Subversion commandline client.

Note: The current source tree only builds against the 0.22.2 release of SVN.

Start by getting the 0.22.2 source tarball from http://subversion.tigris.org. You will also need the VS.NET version of the Berkeley DB binaries (4.0.14 version). Make sure you don't get the VC6.0 versions, since they will cause spurious access violations at runtime.

Extract the subversion tarball (using Winzip or similar). Then, extract the Berkeley zip into a directory called db4-win32 at the root of the tree (alongside the doc and build directories).

APR

Subversion relies on the Apache Portable Runtime (APR) library. Most of APR is included in the tarball, but to get Subversion to build on Win32 you will also need the apr-iconv module. You can check this out from the Apache CVS repository using a CVS client:

cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic login
(password: 'anoncvs')
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic co apr-iconv

You may run into a problem with the supplied .dsp files in the tarball. If the person packaging it was using a UNIX machine, the line endings will probably be wrong and VS.NET will refuse to convert them. You can use an utility like unix2dos to fix this, or you can check out those modules from CVS using the approach described above(just substitute apr and apr-util for apr-iconv). All the modules need to be in the top-level directory of the Subversion source tree.

Building Neon

Open a VS.NET command shell (Start->Programs->Microsoft Visual Studio.NET → Tools → VS-NET Command Prompt), go into the neon directory at the root of the Subversion tree, and type the following:

cmd /c ..\build\win32\build_neon.bat debug
cmd /c ..\build\win32\build_neon.bat release

Building Subversion

Building Subversion requires you to have Python installed.

1. Run gen-make.py -t dsp from the root of the Subversion tree. This will generate the VC++ project files.
2. Open the solution by clicking on subversion_msvc.dsw at the root. VS.NET will prompt you to convert the workspace to the new VS.NET format. Answer yes to this.
3. Disable the neon project in the VS.NET solution, as well as the tests_* projects and mod_dav_svn. If you chose to use the 4.1.25 version of the Berkeley binaries, you will need to change the svn, svnadmin, svnlook, svnserve and svnversion projects to link against libdb41(d).lib instead of libdb40(d).lib. The linker settings can be accessed through project properties, Linker → Input → Additional Dependencies. The whole solution should build now.
4. Build both debug and release versions. Put the generated binaries somewhere in your PATH.
5. Create a new system environment variable, SVNSRC, to point to the Subversion source tree. This is necessary to get the Ankh build to locate the Subversion libraries and headers.

Building Ankh itself

1. Unzip the Ankh.zip file from the enclosed CD.
2. Open and build src.sln. Run the ReCreateCommands.reg file in the Ankh directory.
3. To get VS.NET to recognise the custom menu icons, you need to edit SatelliteDll.reg in the same directory. Set the path for SatelliteDllPath to the correct one for your system and then run the .reg file.
4. Go into the Project Properties of the Ankh project. Choose Configuration Properties->Debug. Set Debug Mode to Program, and set Start Application to the path to devenv.exe on your system.

Starting the Ankh project should now create a new instance of VS.NET with Ankh loaded.

5 Conclusion

Improvements due to the development of Ankh:

- By creating a GUI, SVN is made available to a large number of Windows programmers. Today the audience of SVN is mainly Unix users. We hope to convert this trend.
- Increased productivity since revision control is directly accessible in VS.NET.
- We can offer the Windows users a FREE open source revision control tool integrated in VS.NET
- Ankh is a part of the open source community at ankhsvntigris.org.
- VS .NET integration coded in C# is poorly documented. The Ankh project represents a complete example of how to hook up to menus, make explorer panes, output panes and how to make dialog boxes and user controls coded in C#. Since Ankh is open source code we hope to transfer our knowledge to others.

Ankh covers the most frequently used functionality in a daily session with Subversion.

The .NET wrapper layer is available as a standalone component so that it easily can be used in other applications in the future. The wrapper covers a complete translation of the Subversions Client API coded in C to Managed C++ and C# code. NSvn.Core covers all of Subversion’s client functionality.

6 References

1. “NUnit” http://nunit.org/
2. “VS.NET help” integrated in VS.NET
3. “NAnt” http://nant.sourceforge.net/
4. ”Ant” http://ant.apache.org/manual/

5. "Creating Office Managed COM Add-Ins with Visual Studio .NET" by Paul Cornell, Microsoft Corporation, June 6, 2002
6. "Enterprise Architect" at http://www.sparxsystems.com.au/
7. "The Mixed DLL Loading Problem Using Visual C++ .NET", March 16, 2003 http://www.codeguru.com/dll/kmg15.html
8. "15 SWIG and C#" http://www.swig.org/Doc1.3/Csharp.html
9. "Subversion: The Definitive Guide", Draft: Revision 5113, 26 February, 2003, by Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato, http://subversion.tigris.org
10. "Inside Microsoft Visual Studio .NET" by Brian Johnson, Craigh Skibo and Marc Young
11. Draco.NET http://draconet.sourceforge.net/
12. Doxygen http://www.stack.nl/~dimitri/doxygen/

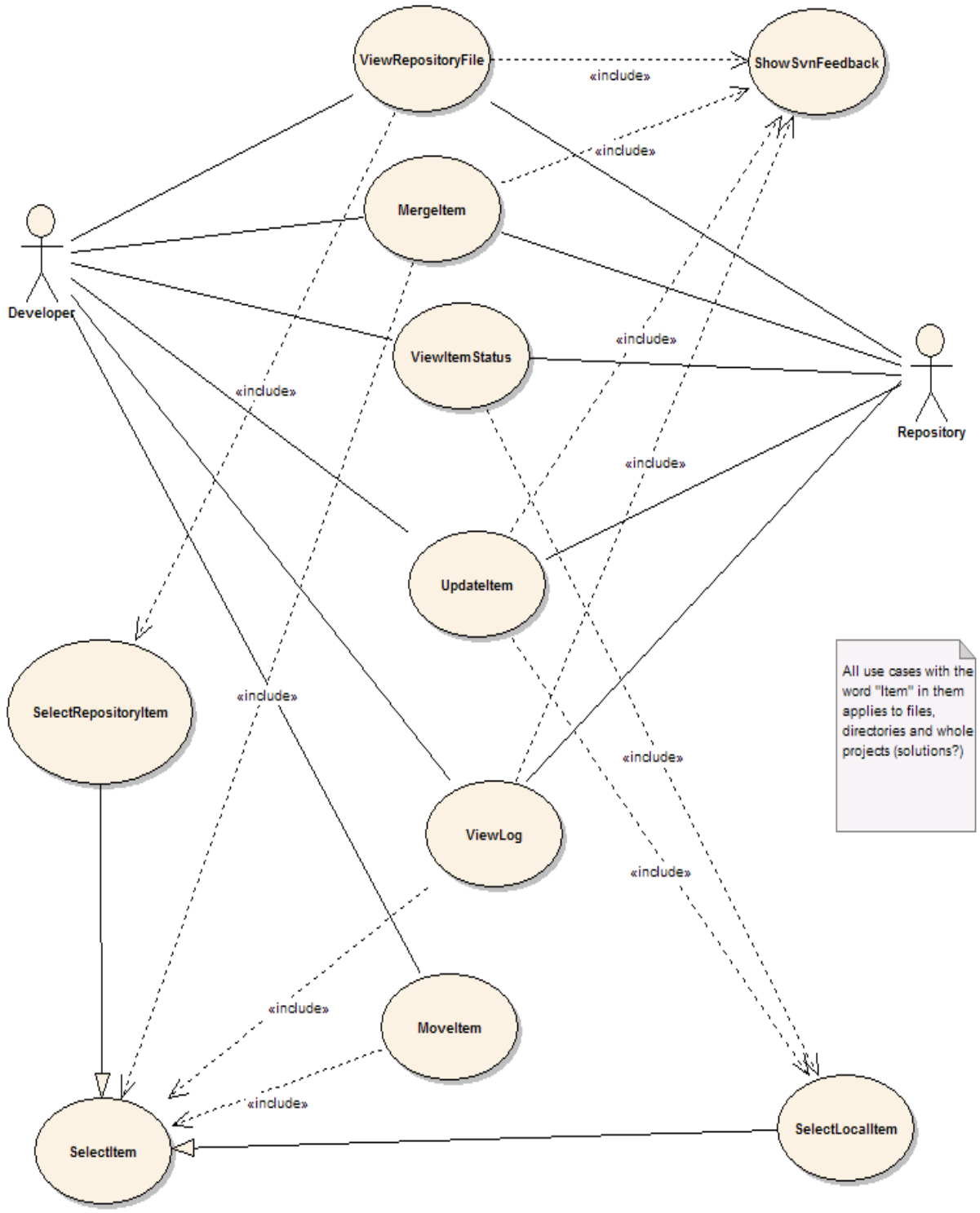
7 Appendix

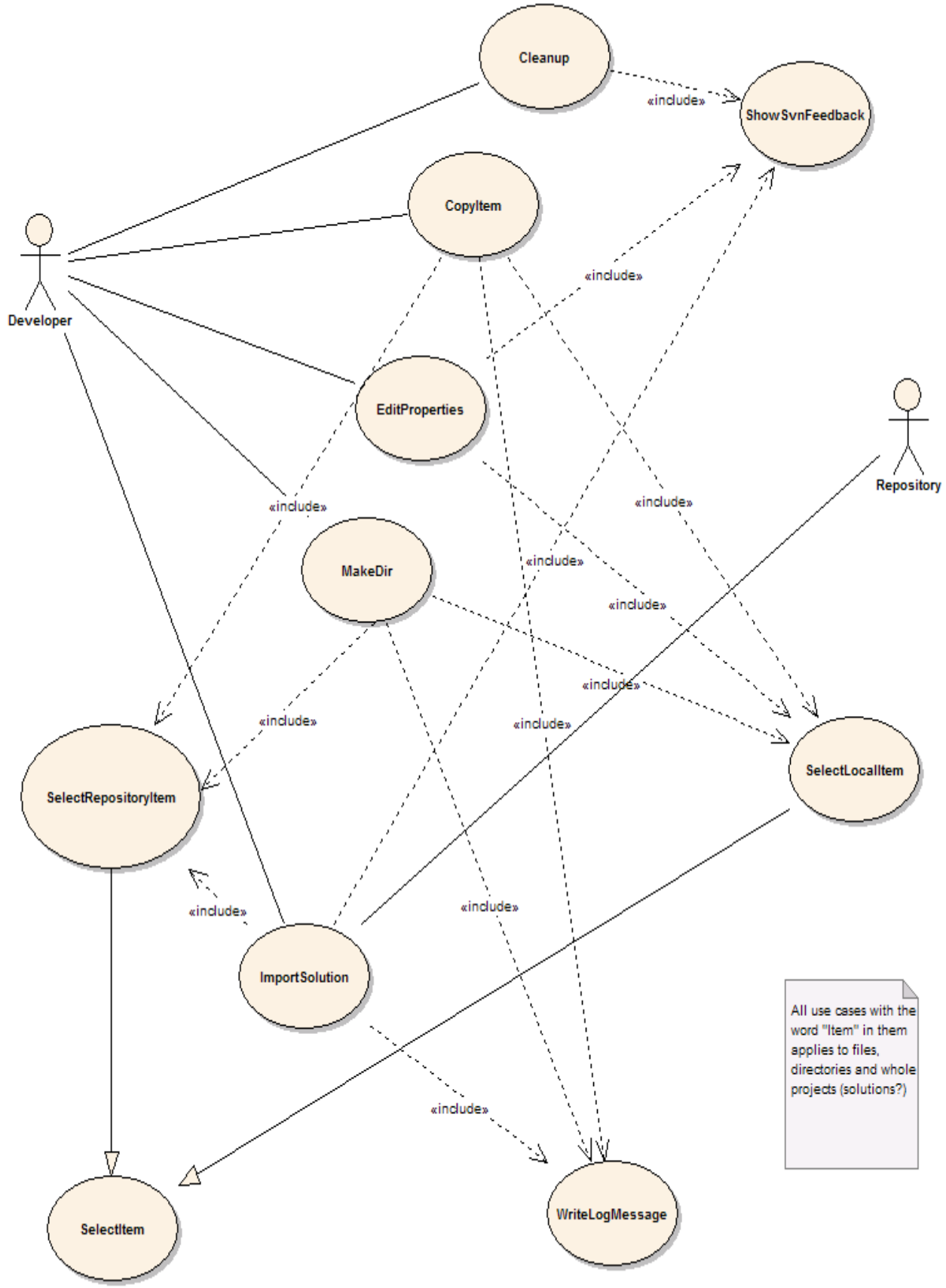
7.1 Dictionary

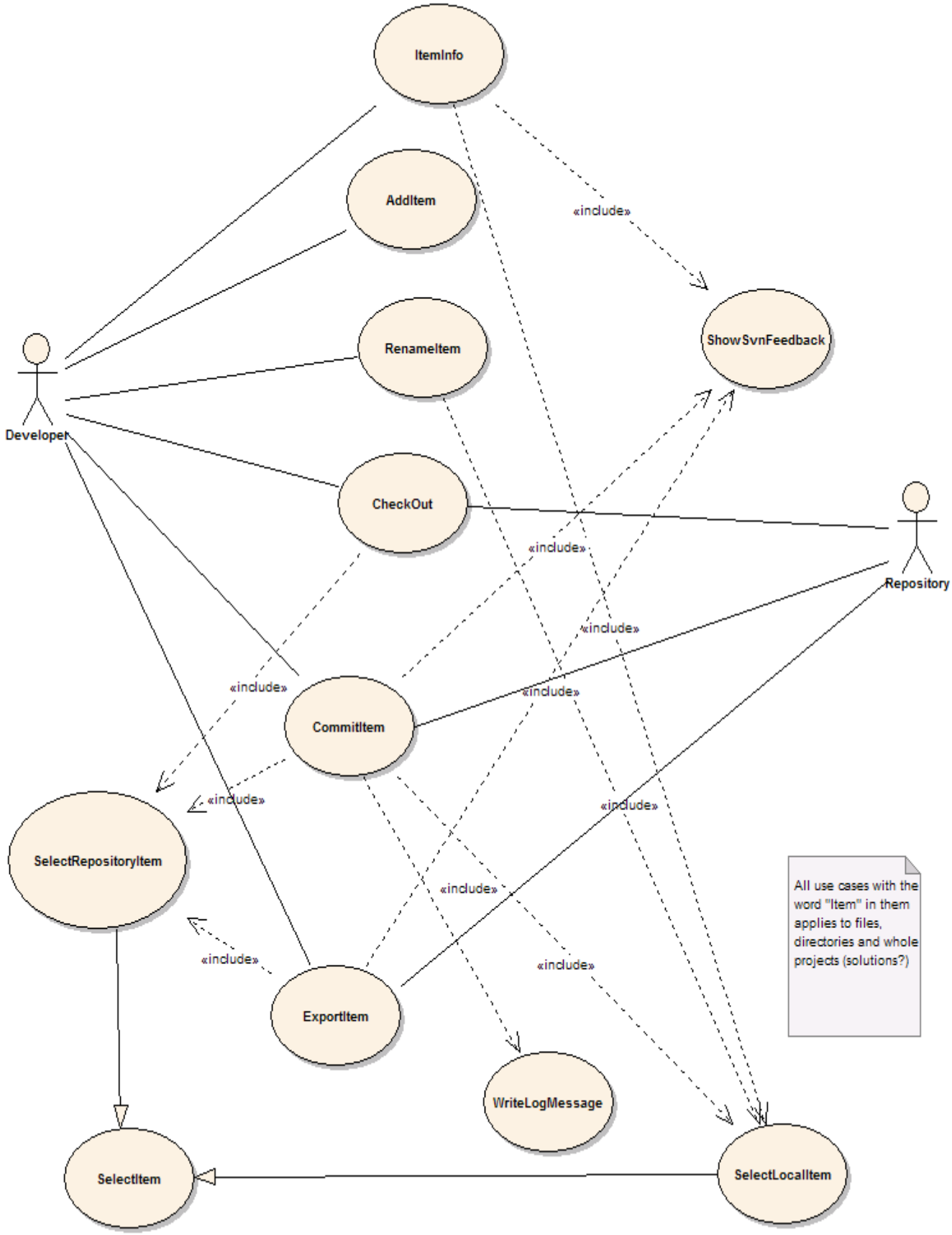
Addin	A software component that extends the functionality of another product.
Alpha release	Alpha is the first release to the general public where basic functionality exists and a reasonable amount of stability is present. Some features may still be absent and there are probably major defects still in the code
API	Application Program Interface
Base revision	Last committed or updated revision
Beta release	A beta release is significantly more advanced than the alpha release. While still not completely stable because of lingering bugs, all features planned for the final release should be present and reasonably reliable. This is when the more elusive bugs will be become apparent.
COM	Component Object Model (COM), a software architecture that allows the components made by different software vendors to be combined into a variety of applications. COM defines a standard for component interoperability. It is not dependent on any particular programming language, is available on multiple platforms, and is extensible.
Commit	Send changes from your working copy to the repository.
Conflicted file	If two users modify the same file at the same line a conflict will occur if the file is under revision control. The conflict occurs because the repository simply doesn't know whether both or only one of the lines should be added.
CVS	Concurrent Versions System -A free open source version control system, the predecessor to Subversion.
DLL	Dynamic Link Library
Doxygen	Doxygen is a documentation system for C++, C, Java, IDL (Corba, Microsoft, and KDE-DCOP flavours), and to some extent PHP and C#. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual from a set of documented source files.

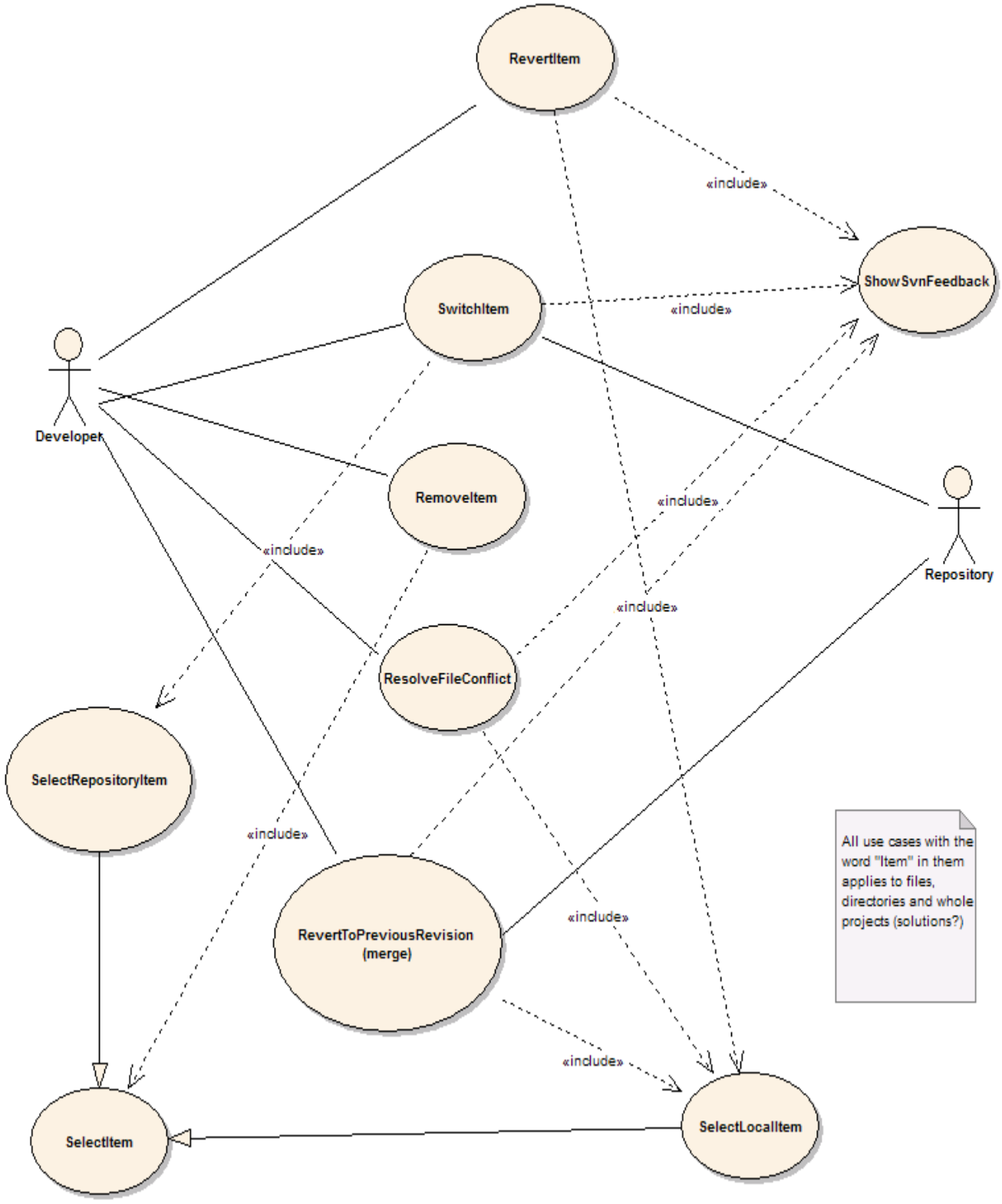
Draco.NET	Draco.NET is an automated build system. It monitors your source repository, automatically rebuilds your project when changes are detected and e-mails you the build result along with a list of changes since the last build. Draco.NET version 1.4 supports the NAnt build tool. It now also supports Subversion as we contributed code to their repository.
Folder	Directory
GUI	Graphical User Interface
Head revision	Last revision in repository
Item	File, folder, project or solution
IDE	Integrated Development Environment
IIS	Internet Information Server
IL	Intermediate Language, the bytecode to which all languages in the .NET framework compile.
NAnt	A free .NET build tool.
NUnit	A unit testing framework, based on JUnit and part of the xUnit family. .
Repository	The server component of the Subversion version control system; this is where your files get stored.
Repository Explorer	An Ankh tool used to browse a Subversion repository.
Revision control system	In a typical software development environment, many developers will be engaged in work on one code base. Revision control makes it easier for several people to share the same collection of program source code. A revision control system is a centralised system for sharing information where the core is a repository.
Solution Explorer	A window in VS.NET used to manage files in a project/solution
SVN	Short name for Subversion, which is a free open source revision control system.
SWIG	Simplified Wrapper and Interface Generator
UI	User Interface
UML	Unified Modelling Language
Unversioned file	File not under version control
VB	Visual Basic .NET
VC++	Visual C++ .NET
VC#	Visual C# .NET
Versioned file	File under version control
VS .NET	Visual Studio .NET. A programming platform that includes environments for developing C#, C++, Managed C++ and Visual Basic programs.
WC	Short for Working Copy.
Working Copy	Collection of files and directories under revision control.

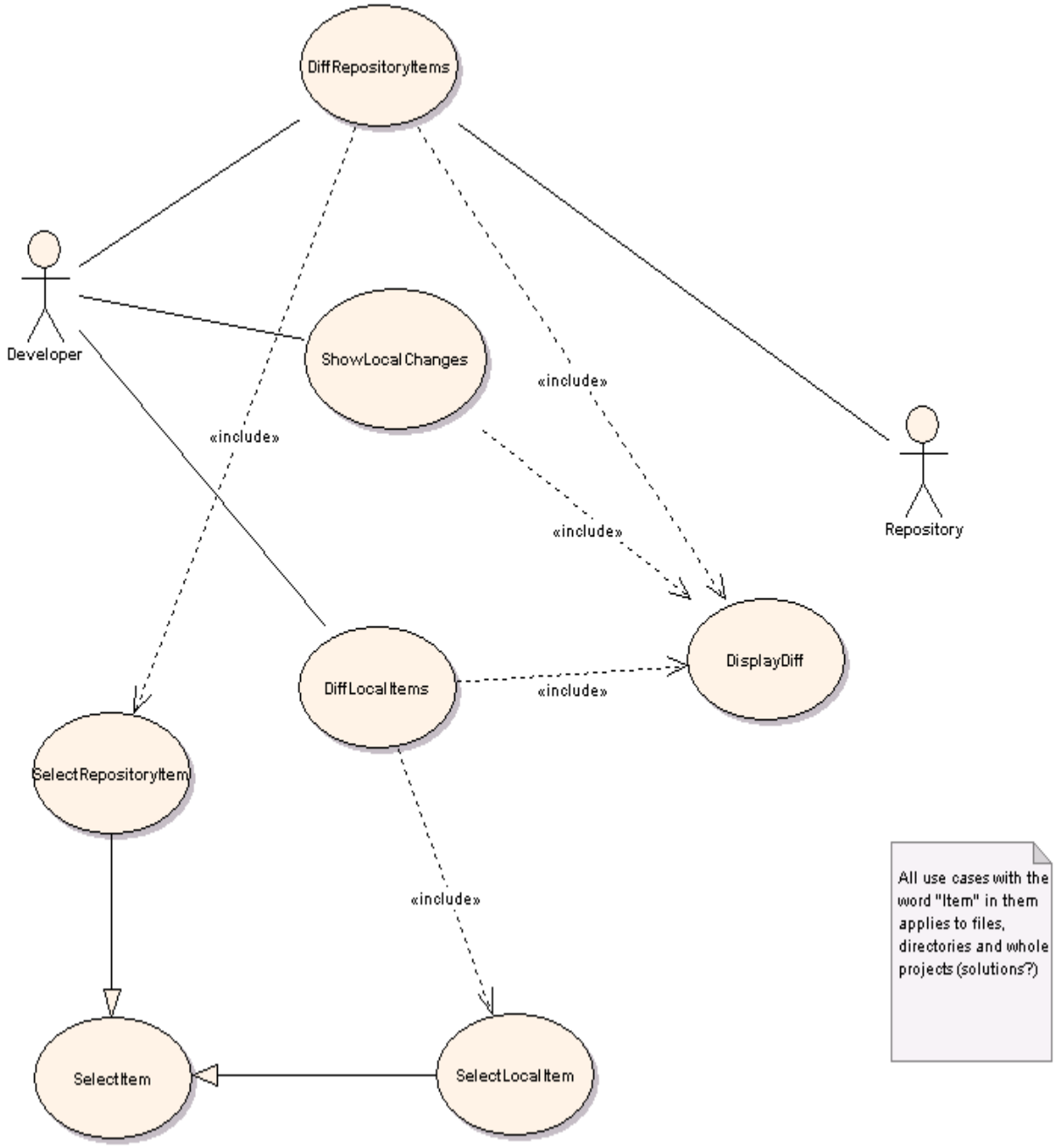
7.2 Use case diagrams




Cleanup
ShowSvnFeedback
CopyItem
Developer
EditProperties
Repository
MakeDir
SelectRepositoryItem
SelectLocalItem
ImportSolution
SelectItem
WriteLogMessage
«include»
All use cases with the word "Item" in them applies to files, directories and whole projects (solutions?)


ItemInfo
AddItem
RenameItem
CheckOut
CommitItem
SelectRepositoryItem
ExportItem
WriteLogMessage
SelectItem
SelectLocalItem
ShowSvnFeedback
Developer
Repository
«include»
All use cases with the word "Item" in them applies to files, directories and whole projects (solutions?)


RevertItem
«include»
ShowSvnFeedback
SwitchItem
«include»
Developer
RemoveItem
«include»
«include»
Repository
«include»
ResolveFileConflict
SelectRepositoryItem
«include»
All use cases with the word "Item" in them applies to files, directories and whole projects (solutions?)
RevertToPreviousRevision (merge)
«include»
«include»
SelectItem
SelectLocalItem


DiffRepositoryItems
Developer
ShowLocalChanges
«include»
«include»
«include»
Repository
DiffLocalItems
DisplayDiff
«include»
SelectRepositoryItem
«include»
All use cases with the word "Item" in them applies to files, directories and whole projects (solutions?)
SelectItem
SelectLocalItem

7.3 Use cases

The following diagrams describe the use cases in the Ankh project. Each individual use case is derived from the objects in the conceptual model listed in the previous section.

Use case	Add item
Actors	1 Developer
Preconditions	
1 The item is not currently in the workspace 2 The item is in a directory that is currently in the workspace	
Scenario	
1 The user invokes the VS.NET command File.AddNewItem or File.AddExistingItem 2 The item is added to the working copy 3 <Include 'ShowSvnFeedback'>	
Postconditions	
1 The item is in the workspace	

Use case	Cleanup
Actors	1 Developer
Preconditions	
1 The working copy is locked 2 An SVN operation has failed due to the locked wc	
Scenario	
1 Developer is asked whether a cleanup is desired 2 Cleanup is run on the working copy area 3 <Include 'ShowSvnFeedback'>	
Postconditions	
1 The working copy is in an unlocked state 2 OR: Cleanup failed	

Use case	Commit item
Actors	1 Developer 2 Repository
Preconditions	
1 The item has local changes 2 The item is in the working copy	
Scenario	
1 <Include 'SelectLocalItem'> 2 The user selects Commit item from a menu 3 <Include 'WriteLogMessage'> 4 Commit the item to the repository 5 <Include 'ShowSvnFeedback'>	
Postconditions	
1 The item is committed 2 OR: The commit is rejected because of other committed changes	

Use case	Copy Item
Actors	1 Developer 2 Repository
Preconditions	
1 Item doesn't exist at destination path 2 Item is under revision control	
Scenario	
1 <Include 'SelectItem'> 2 Developer selects "Copy" from menu 3 <Include 'SelectItem'> 4 <Include 'WriteLogMessage'> 5 Developer invokes "Paste" 6 svn copies item	
Postconditions	
1 Item is copied to new destination	

Use case	Diff Local Item
Actors	1 Developer 2 Repository
Preconditions	
1 The item is under revision control	
Scenario	
1 <Include 'SelectLocalItem'> 2 Developer selects "Diff" from menu 3 Developer selects revisions 4 svn performs a diff 5 <Include 'DisplayDiff'>	
Postconditions	
1 The diff is displayed	

Use case	Diff Repository Item
Actors	1 Developer 2 Repository
Preconditions	
1 The item exists in the repository	
Scenario	
1 <Include 'SelectRepositoryItem'> 2 Developer selects "Diff" from menu 3 Developer selects revisions 4 If developer wants to compare two different paths select another repository 5 svn performs a diff 6 <Include 'DisplayDiff'>	
Postconditions	
1 The diff is displayed	

Use case	Edit Properties
Actors	1 Developer
Preconditions	
1 The item is under revision control	
Scenario	
1 <Include 'SelectItem'> 2 User selects "edit properties" on an item 3 Dialogbox pops up and the developer can edit the property list 4 Developer close the dialog box 5 The properties are applied onto the item 6 <Include 'ShowSvnFeedback'>	
Postconditions	
1 Properties are set on the item	

Use case	Export Item
Actors	1 Developer 2 Repository
Preconditions	
1 The item exists in the repository	
Scenario	
1 <Include 'SelectRepositoryItem'> 2 Developer selects "Export" 3 Developer selects local folder to export to 4 SVN exports the item 5 <Include 'ShowSvnFeedback'>	
Postconditions	
1 The item is exported to a local directory	

Use case	Import Solution
Actors	1 Developer 2 Repository
Preconditions	
1 The solution is not currently an SVN working copy	
Scenario	
1 Developer selects a VS.NET solution 2 Developer selects "Import solution" 3 <Include 'SelectRepositoryItem'> 4 <Include 'WriteLogMessage'> 5 SVN imports all source files contained in the solution 6 Close solution 7 Inform developer that backup copies will be made, and where 8 Make backup copies of solution directories 9 Delete solution directories 10 Check out solution from repository to original location 11 Reopen solution	
Postconditions	
1 Solution is under version control	

Use case	Item info
Actors	1 Developer
Preconditions	
1 The item is under revision control	
Scenario	
1 <Include 'SelectLocalItem'> 2 Retrieve info from svn 3 Display info in VS:NET property pane	
Postconditions	

Use case	Make directory
Actors	1 Developer

Preconditions

1 The parent path exists
2 The parent path is under version control

Scenario

1 <Include 'SelectItem'>
2 Developer selects "Make directory"
3 Developer enters new directory name
4 <Include 'WriteLogMessage'>
5 SVN creates new directory
6 <Include 'ShowSvnFeedback'>

Postconditions

1 New directory is created

Use case	Make patch
Actors	1 Developer
Preconditions	
1 Project is under revision control	
Scenario	
1 <Include 'SelectLocalItem'> 2 Developer invokes the MakePatch command 3 Run SVN diff on the item 4 Present developer with the diff 5 Allow user to copy diff to clipboard 6 OR: create new mail message with diff as attachment	
Postconditions	
1 A patch is created	

Use case	Merge Item
Actors	1 Developer 2 Repository
Preconditions	
1 Paths exist and are in version control	
Scenario	
1 <Include 'SelectItem'> 2 Developer selects "Merge" 3 <Include 'SelectItem'> 4 Developer confirms operation 5 SVN performs the merge 6 <Include 'ShowSvnFeedback'>	
Postconditions	
1 The working copy contains the merged differences	

Use case	Move Ite m
Actors	1 Developer 2 Repository
Preconditions	
1 Item doesn't exist at destination path 2 Item is under revision control	
Scenario	
1 <Include 'SelectItem'> 2 Developer choose "Cut" from menu 3 <Include 'SelectItem'> 4 Developer selects "Paste" from menu 5 <Include 'WriteLogMessage'> 6 svn moves item	
Postconditions	
1 Item is moved to new destination	

Use case	Remove Item
Actors	1 Developer 2 Repository
Preconditions	
1 The item is under revision control	
Scenario	
1 <Include 'SelectItem'> 2 Developer choose "Remove" from menu 3 Developer is asked to confirme the delete 4 svn removes the item	
Postconditions	
1 The item is removed	

Use case	Rename Item
Actors	1 Developer
Preconditions	
1 The item is under version control	
Scenario	
1 <Include 'SelectLocalItem'> 2 Developer invokes the VS.NET Rename command on the item 3 SVN renames the item	
Postconditions	
1 The item has a new name	

Use case	Resolve File Conflict
Actors	1 Developer

Preconditions

1 The files must be under revision control
2 A conflict must have occured

Scenario

1 <Include 'SelectLocalItem'>
2 Developer selects which of the revisions he wants to retain
3 svn deletes the remaining files
4 Status is updated

Postconditions

1 Conflict is resolved

Use case	Revert Item
Actors	1 Developer

Preconditions

1 The item has local changes
2 The item is under revision control

Scenario

1 <Include 'SelectLocalItem'>
2 Developer selects "Revert" from menu
3 Ask developer to confirm the revert
4 svn reverts item to "pristine" state

Postconditions

1 The item is reverted

Use case	Select Item
Actors	1 Developer 2 Repository
Preconditions	
1 The item exists 2 The item is under version control	
Scenario	
1 Developer selects item from a tree view 2 OPTIONAL: Developer selects revision	
Postconditions	
1 The item is selected	

Use case	Select Local Item
Actors	1 Developer
Preconditions	
1 Item under revision control	
Scenario	
1 The developer clicks on an item in Solution Explorer	
Postconditions	
1 The item is selected	

Use case	Select Repository Item
Actors	1 Developer 2 Repository
Preconditions	
1 Item under revision control	
Scenario	
1 The developer specifies an URL to the repository 2 Developer provides login credentials if required 3 SVN connects to repository and requests a list of files 4 The repository files are displayed in a tree-view 5 Developer selects an item from the tree-view	
Postconditions	
1 The item is selected	

Use case	Show Local Changes
Actors	1 Developer
Preconditions	
1 The item is under revision control	
Scenario	
1 <Include 'SelectLocalItem'> 2 Developer choose "Show Local Changes" from menu 3 svn performs a diff 4 <Include 'DisplayDiff'>	
Postconditions	
1 The diff is displayed	

Use case	Show SVN feedback
Actors	1 Developer
Preconditions	
1 The user has initiated an operation that needs to provide feedback	
Scenario	
1 A new Output window is created in VS.NET 2 The window receives focus 3 Output messages are written to the window	
Postconditions	
1 The developer is enlightened	

Use case	Switch Item
Actors	1 Developer 2 Repository
Preconditions	
1 Working copy item exists 2 Repository path exists	
Scenario	
1 <Include 'SelectLocalItem'> 2 Developer selects "Switch" 3 <Include 'SelectRepositoryItem'> 4 Developer confirms the operation 5 SVN switches to the specified path	
Postconditions	
1 The working copy is updated to the specified path	

Use case	Update Item
Actors	1 Developer 2 Repository

Preconditions

1 The item is under revision control

Scenario

1 <Include 'SelectLocalItem'>
2 Developer selects "Update" from the menu
3 Optional: selects revision to be updated to
4 svn updates
5 <Include 'ShowSvnFeedback'>

Postconditions

1 The item is updated

Use case	View Item Status
Actors	1 Developer

Preconditions

1 The item is under revision control

Scenario

1 An icon is displayed in the Solution Explorer next to the item signifying its status

Postconditions

1 Developer is enlightened

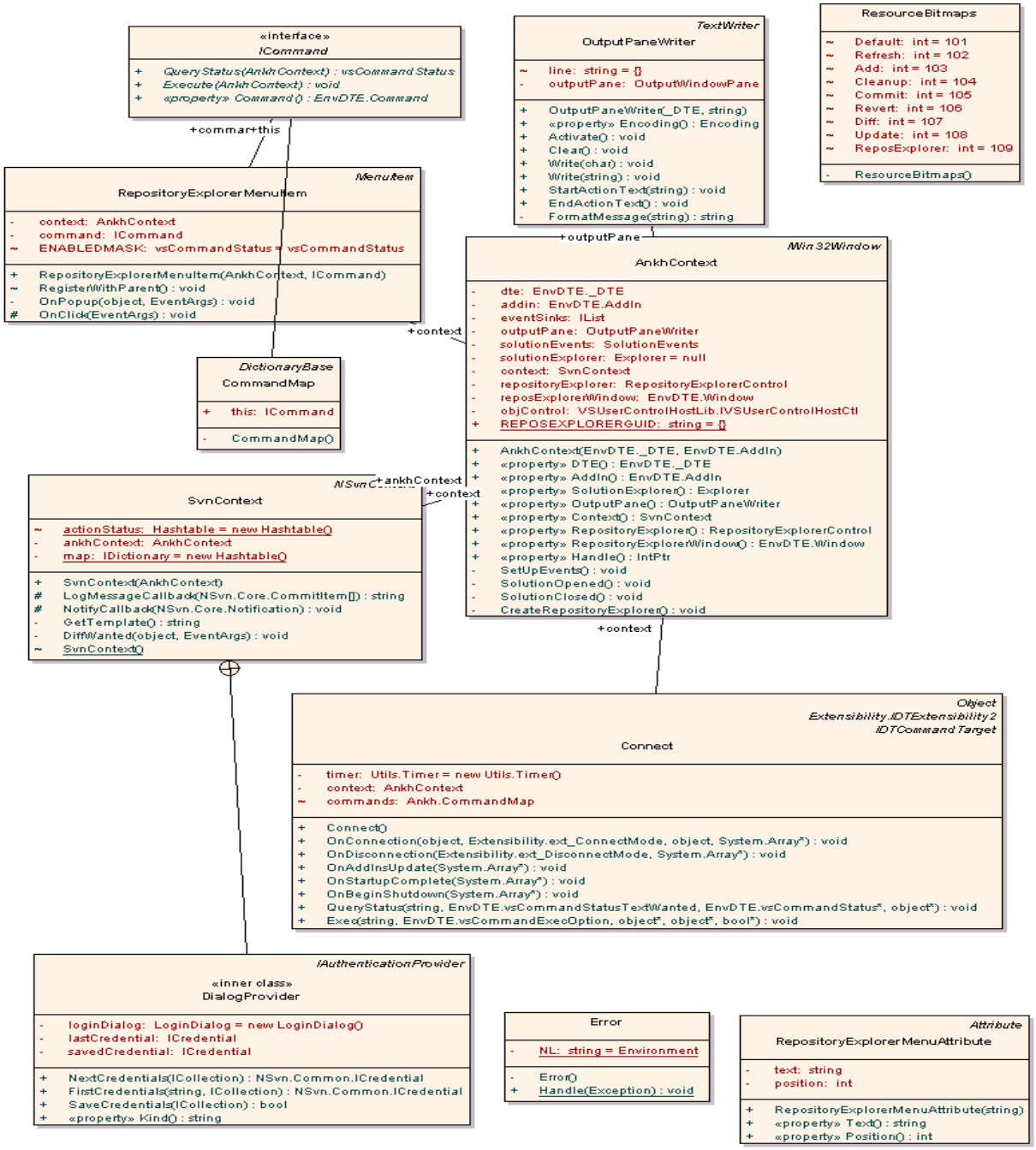
Use case	View Log
Actors	1 Developer 2 Repository
Preconditions	
1 The item must be under revision control	
Scenario	
1 <Include 'SelectItem'> 2 Developer selects "Log" from menu 3 Optional: Developer choose revision range 4 svn fetches log from repository 5 Display log	
Postconditions	
1 Log messages are displayed	

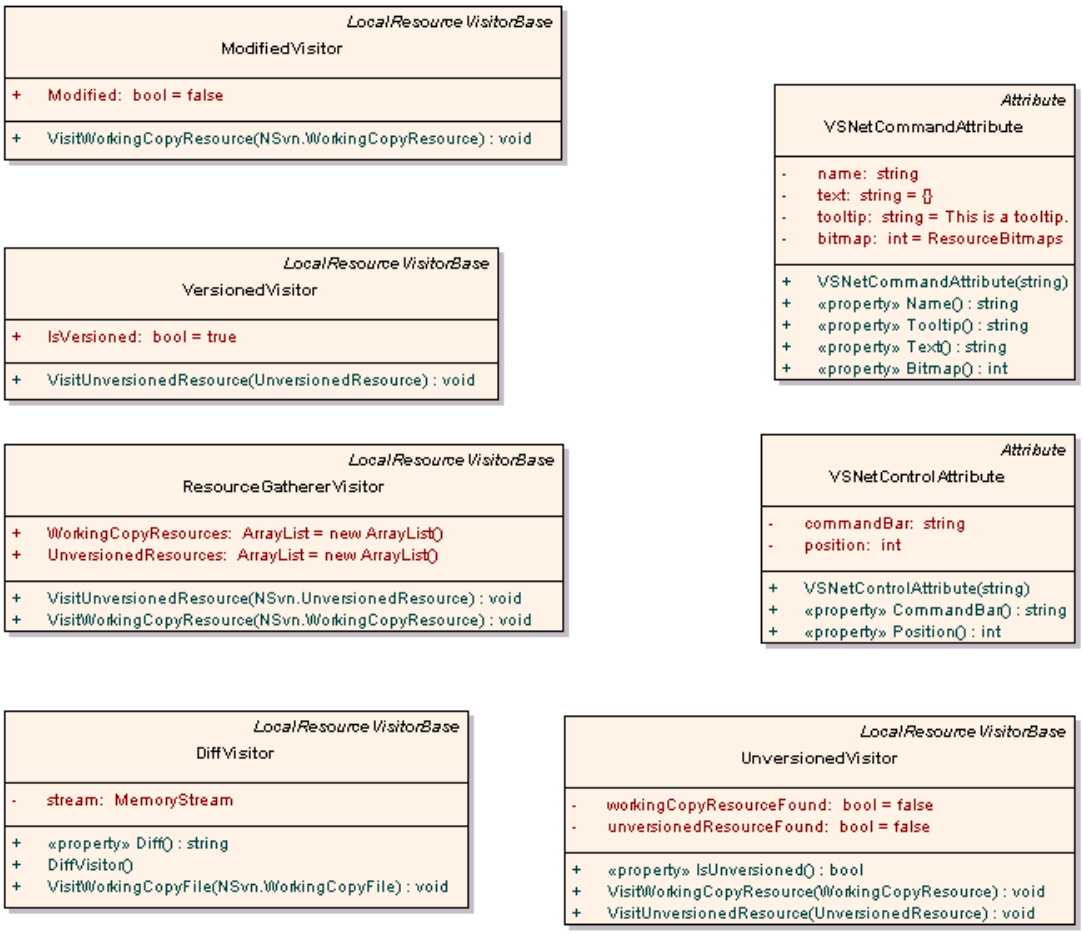
Use case	View Repository File
Actors	1 Repository 2 Developer
Preconditions	
1 The file exists in the repository	
Scenario	
1 <Include 'SelectRepositoryItem'> 2 Developer selects "View" 3 SVN retrieves file contents from repository 4 If file is text: Show in VS.NET editor window 5 else: Invoke associated application	
Postconditions	
1 The selected file is displayed	

Use case	Write Log Message
Actors	1 Developer
Preconditions	
1 An operation requiring log message has been invoked	
Scenario	
1 Developer selects whether he wants to see diff 2 Developer writes log message 3 Developer clicks "ok"	
Postconditions	

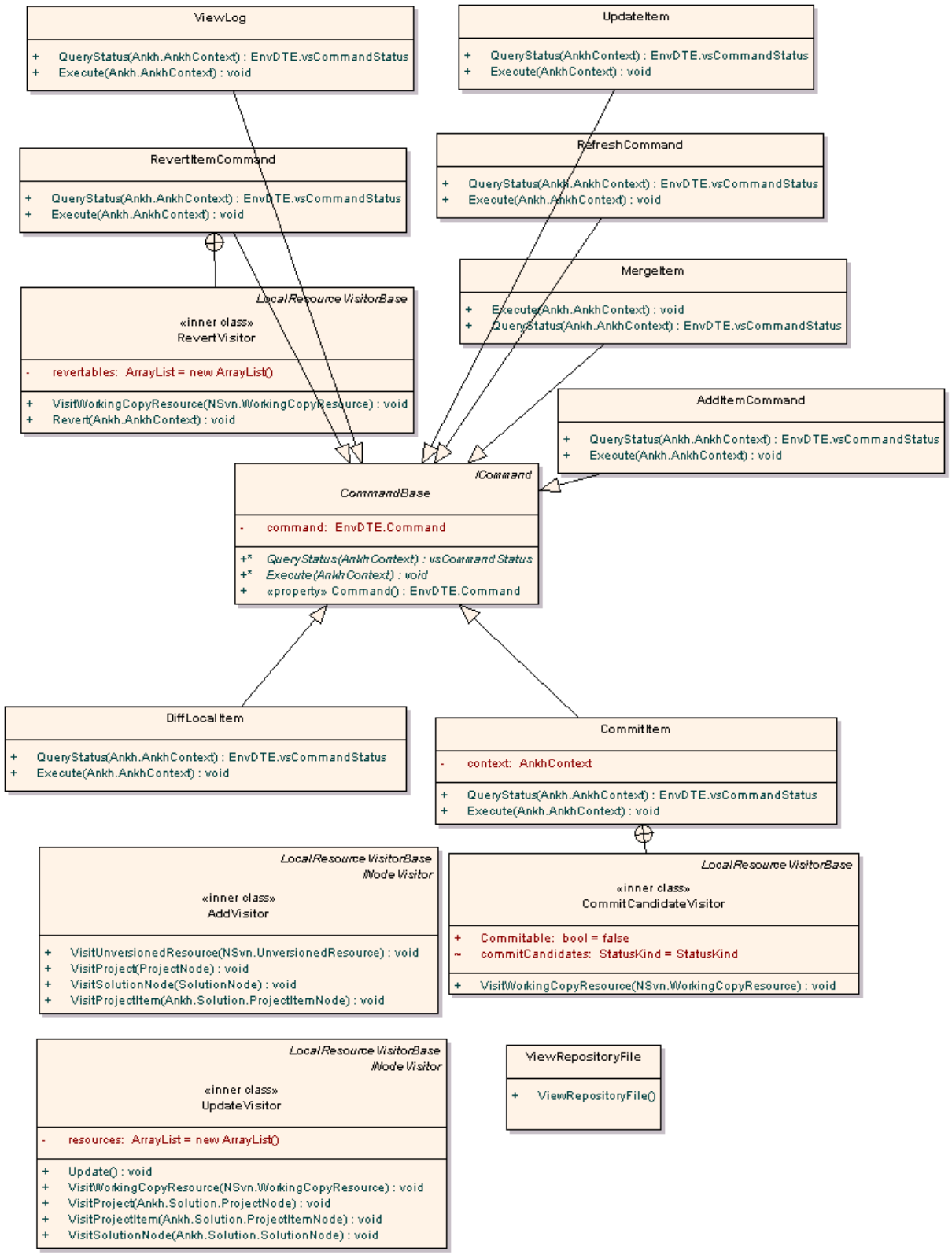
7.4 Class diagrams

7.4.1 Class diagram of Ankh

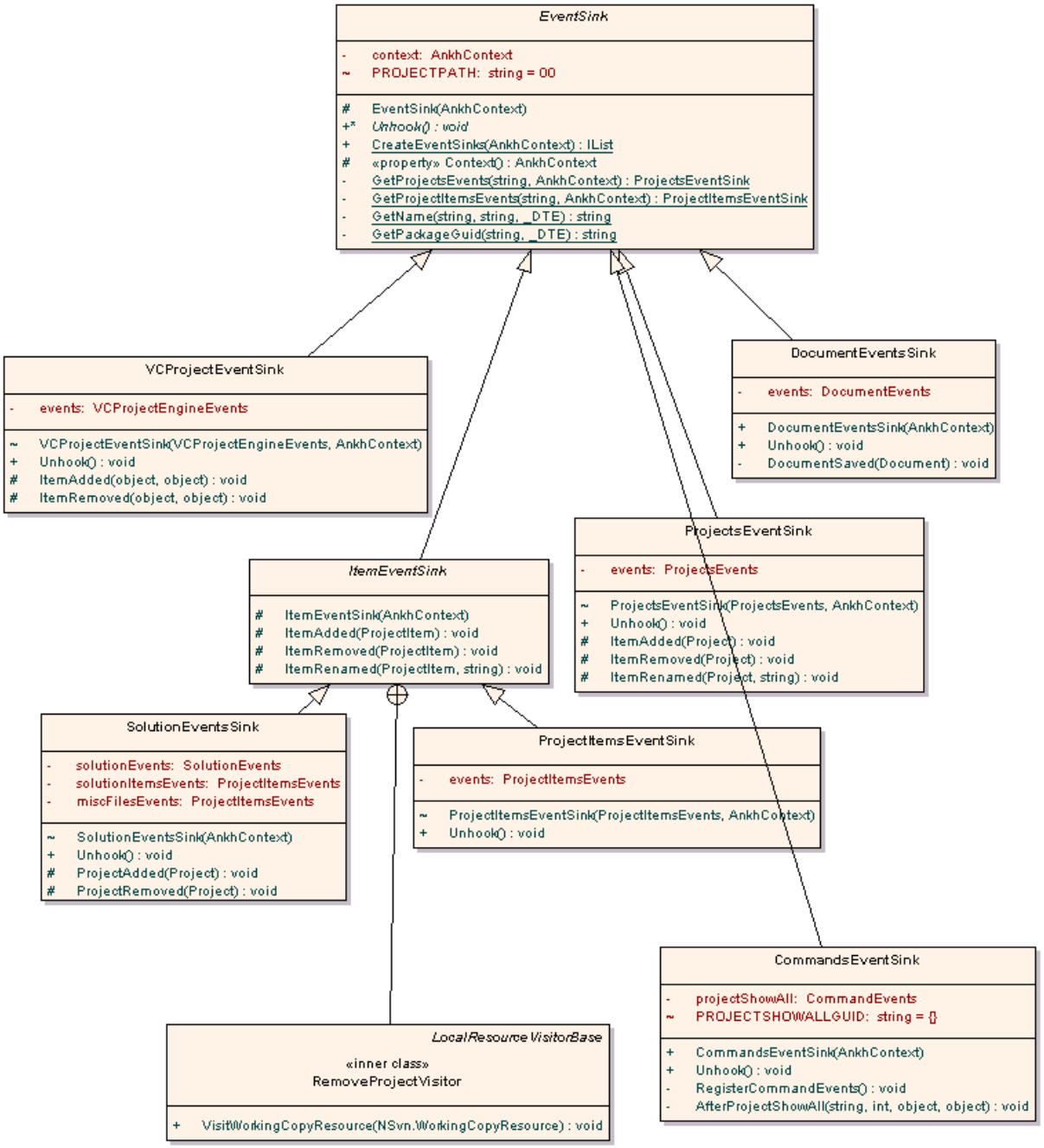



LocalResourceVisitorBase
ModifiedVisitor
+ Modified: bool = false
+ VisitWorkingCopyResource(NSvn.WorkingCopyResource) : void
Attribute
VSNetCommandAttribute
- name: string
- text: string = {}
- tooltip: string = This is a tooltip.
- bitmap: int = ResourceBitmaps
+ VSNetCommandAttribute(string)
+ «property» Name() : string
+ «property» Tooltip() : string
+ «property» Text() : string
+ «property» Bitmap() : int
LocalResourceVisitorBase
VersionedVisitor
+ IsVersioned: bool = true
+ VisitUnversionedResource(UnversionedResource) : void
LocalResourceVisitorBase
ResourceGathererVisitor
+ WorkingCopyResources: ArrayList = new ArrayList()
+ UnversionedResources: ArrayList = new ArrayList()
+ VisitUnversionedResource(NSvn.UnversionedResource) : void
+ VisitWorkingCopyResource(NSvn.WorkingCopyResource) : void
Attribute
VSNetControlAttribute
- commandBar: string
- position: int
+ VSNetControlAttribute(string)
+ «property» CommandBar() : string
+ «property» Position() : int
LocalResourceVisitorBase
DiffVisitor
- stream: MemoryStream
+ «property» Diff() : string
+ DiffVisitor()
+ VisitWorkingCopyFile(NSvn.WorkingCopyFile) : void
LocalResourceVisitorBase
UnversionedVisitor
- workingCopyResourceFound: bool = false
- unversionedResourceFound: bool = false
+ «property» IsUnversioned() : bool
+ VisitWorkingCopyResource(WorkingCopyResource) : void
+ VisitUnversionedResource(UnversionedResource) : void

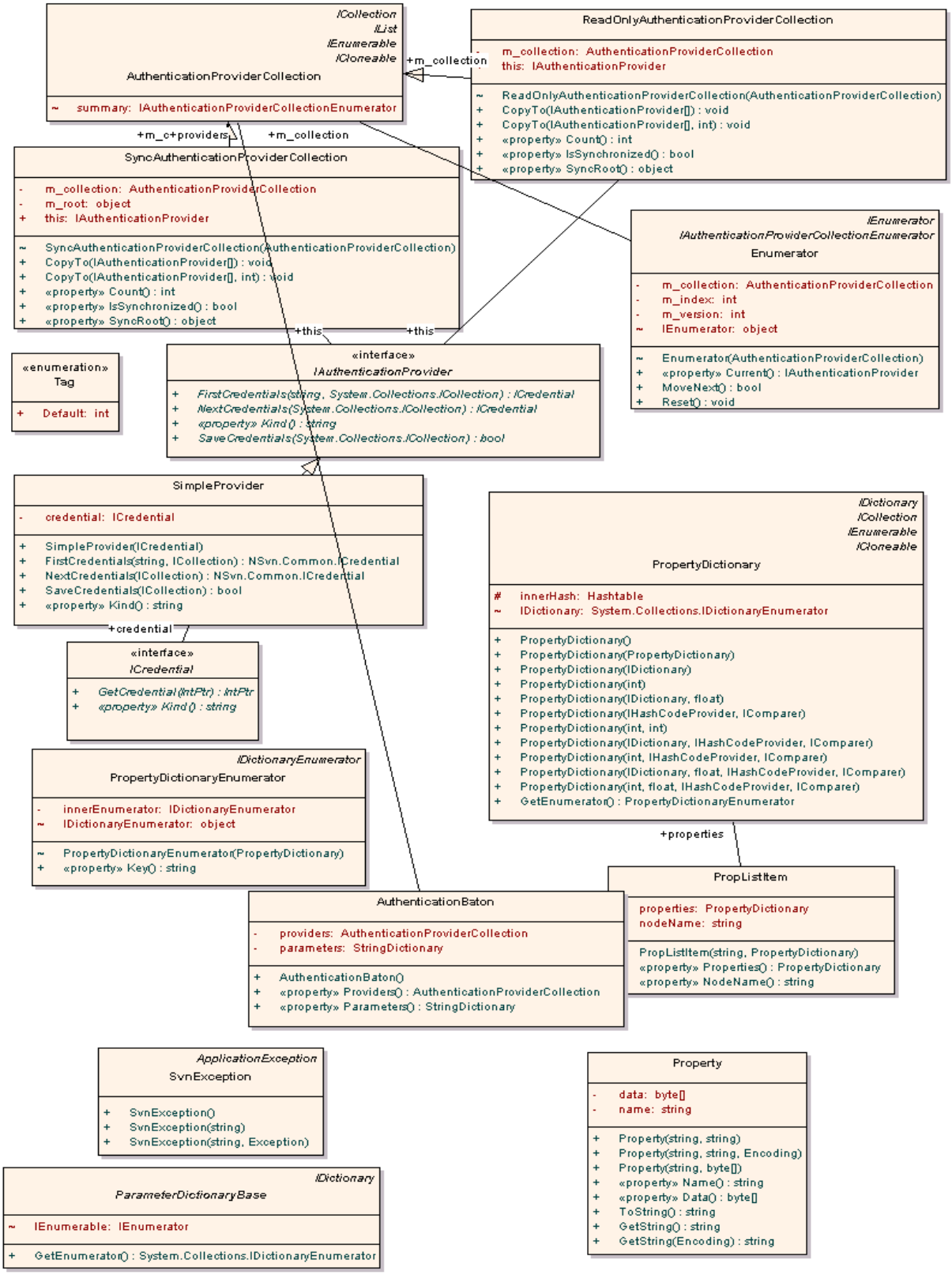
7.4.2 Class diagram of Ankh.Commands



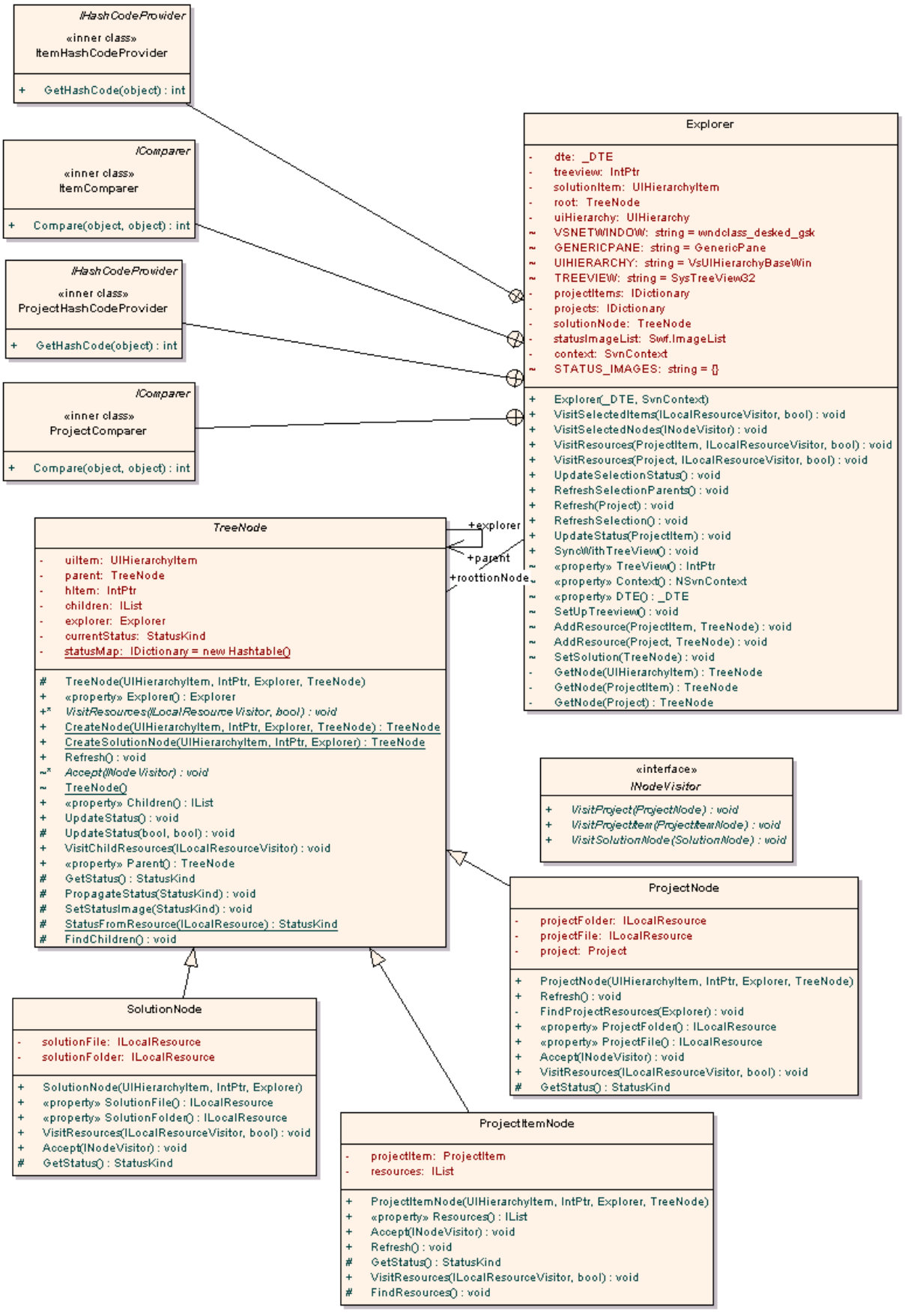
7.4.3 Class diagram of Ankh.Eventsinks



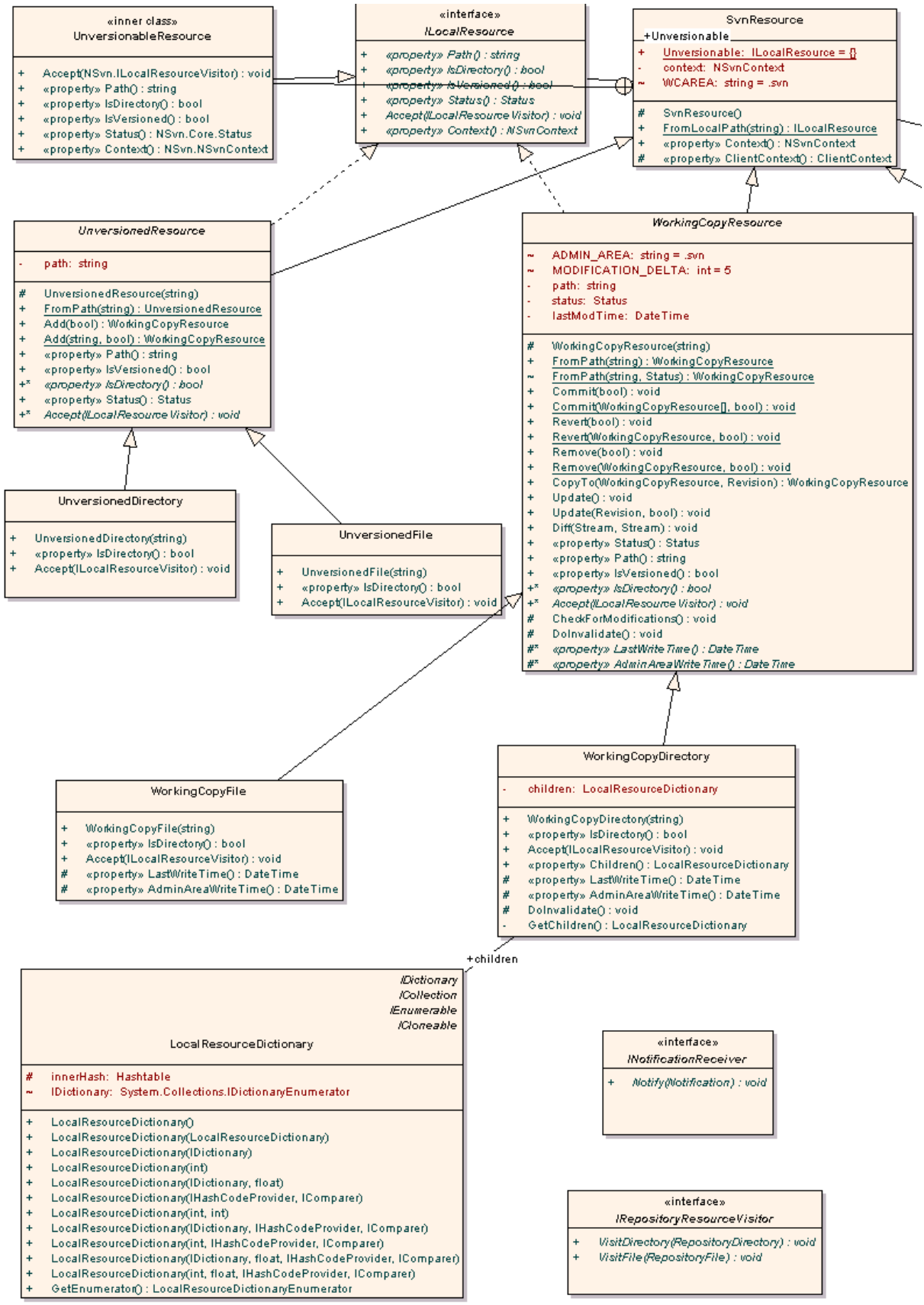
7.4.4 Class diagram of NSvn.Common

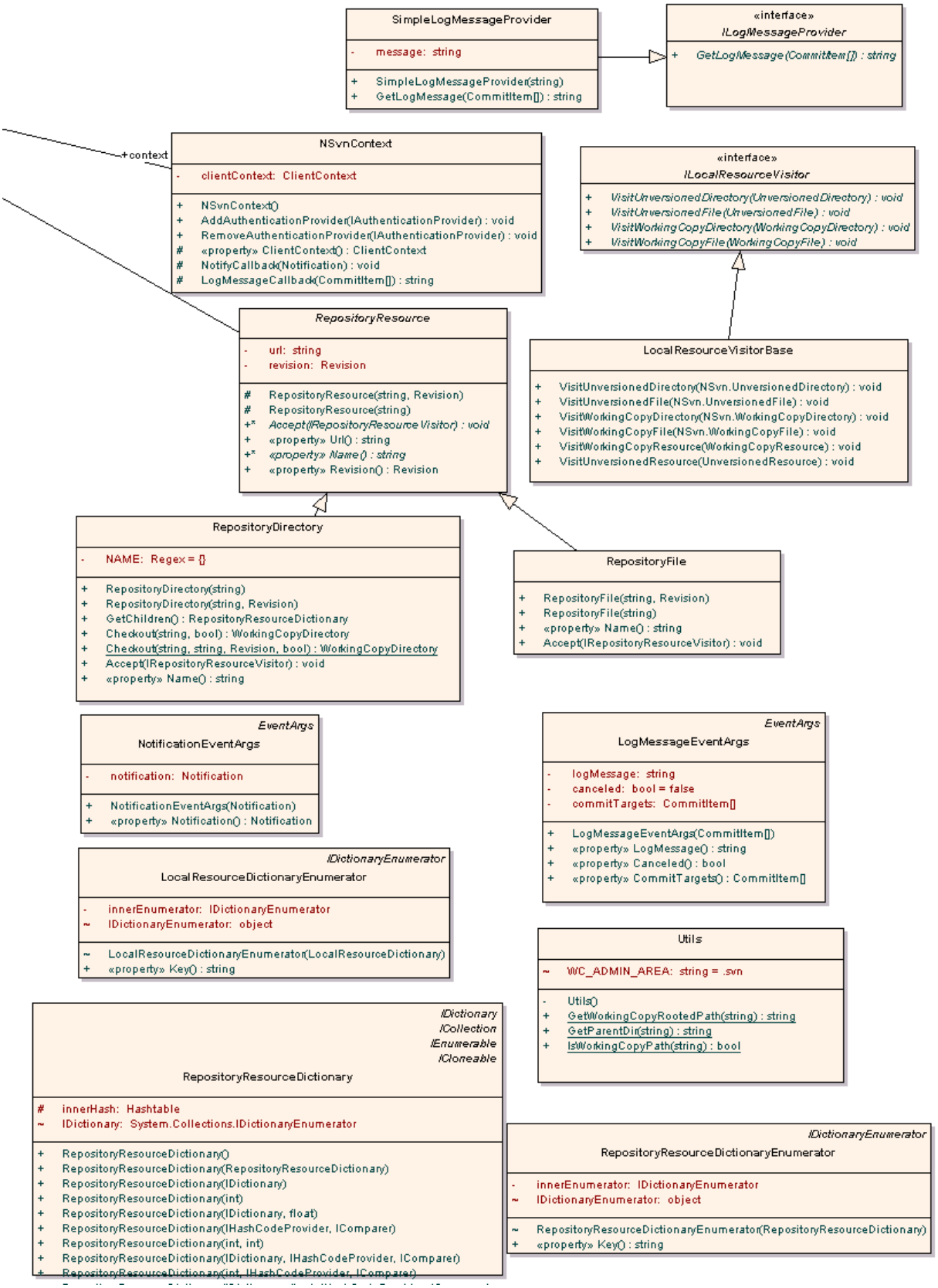


7.4.5 Class diagram of Ankh.Solution

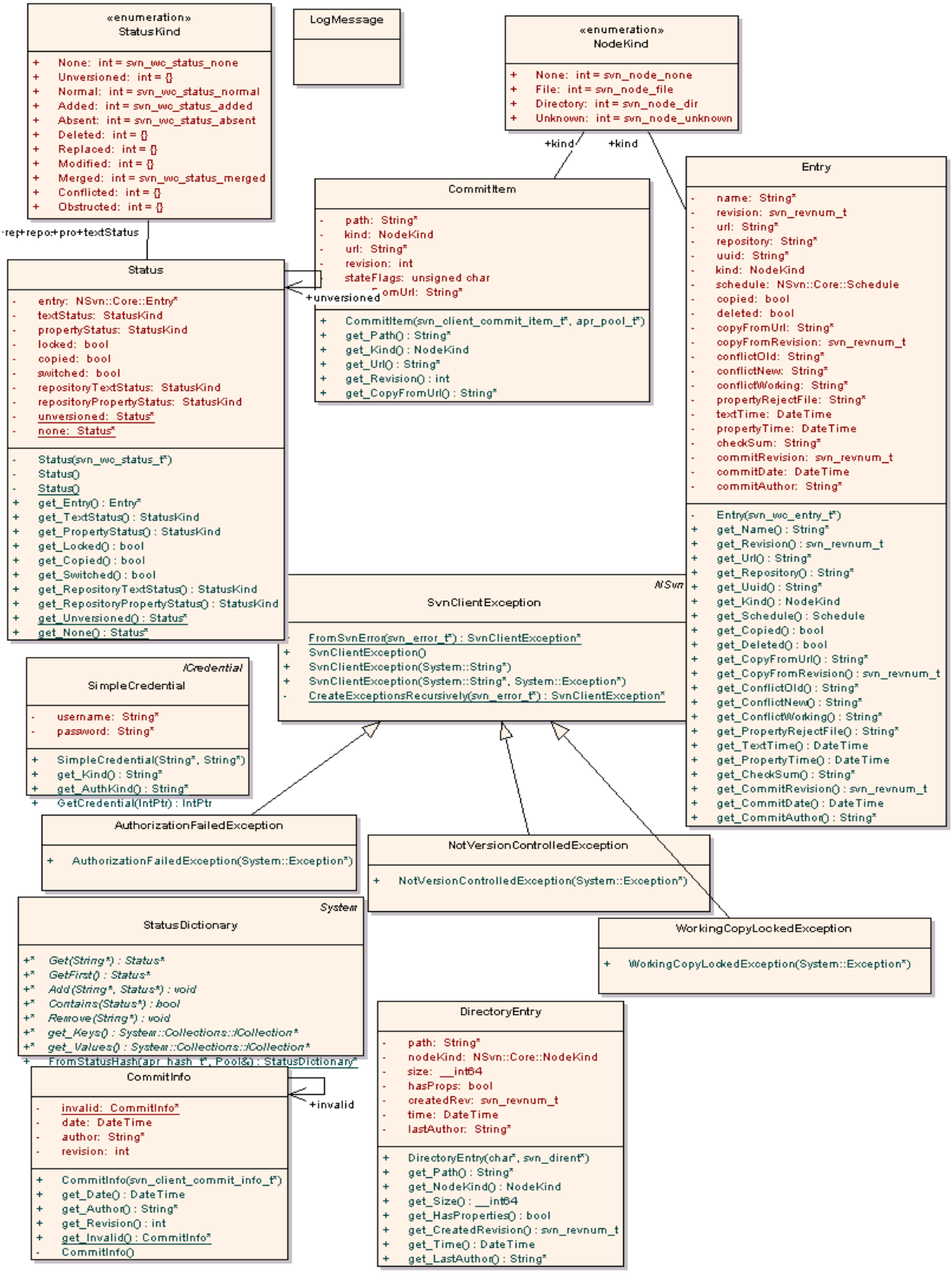


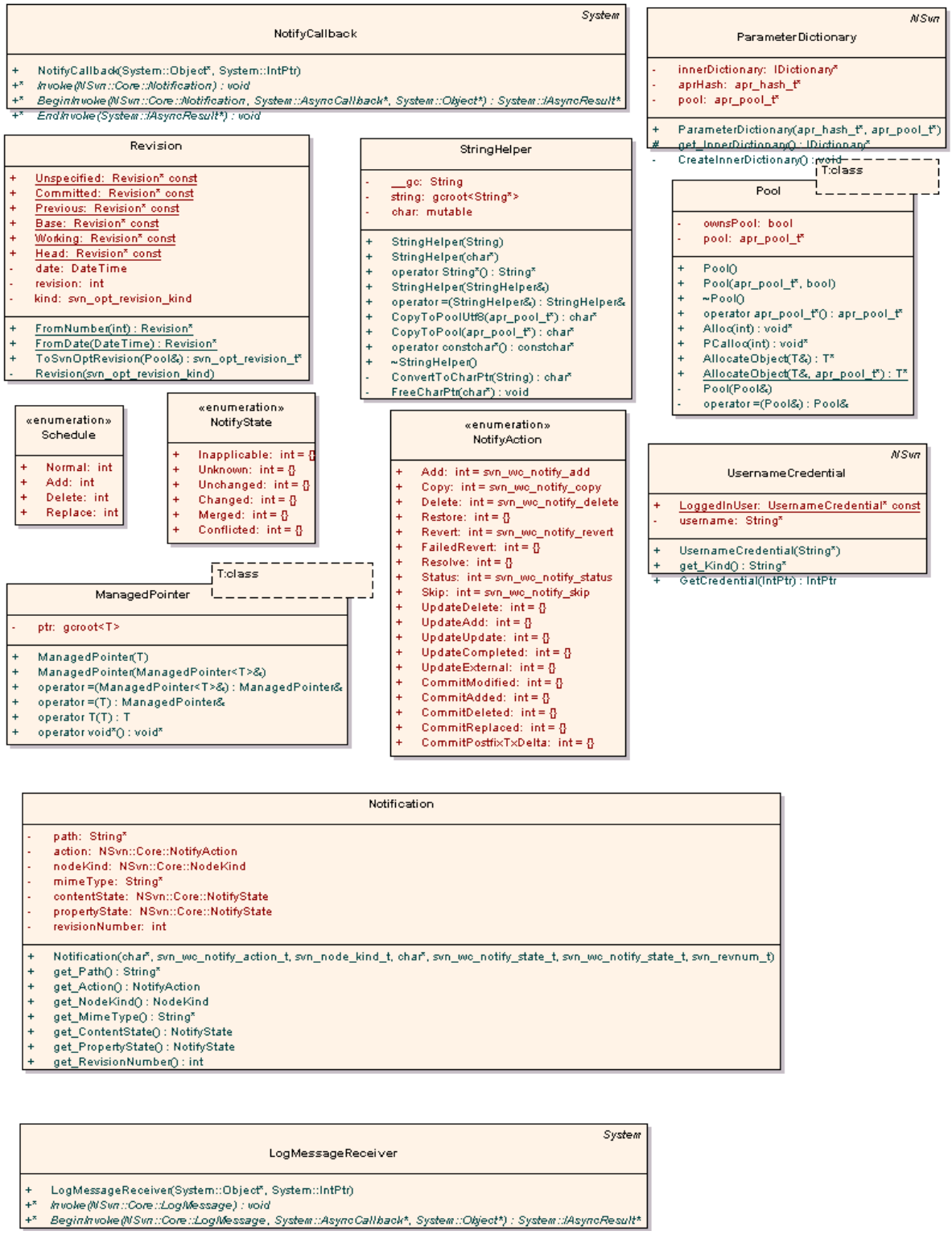
7.4.6 Class diagram of NSvn



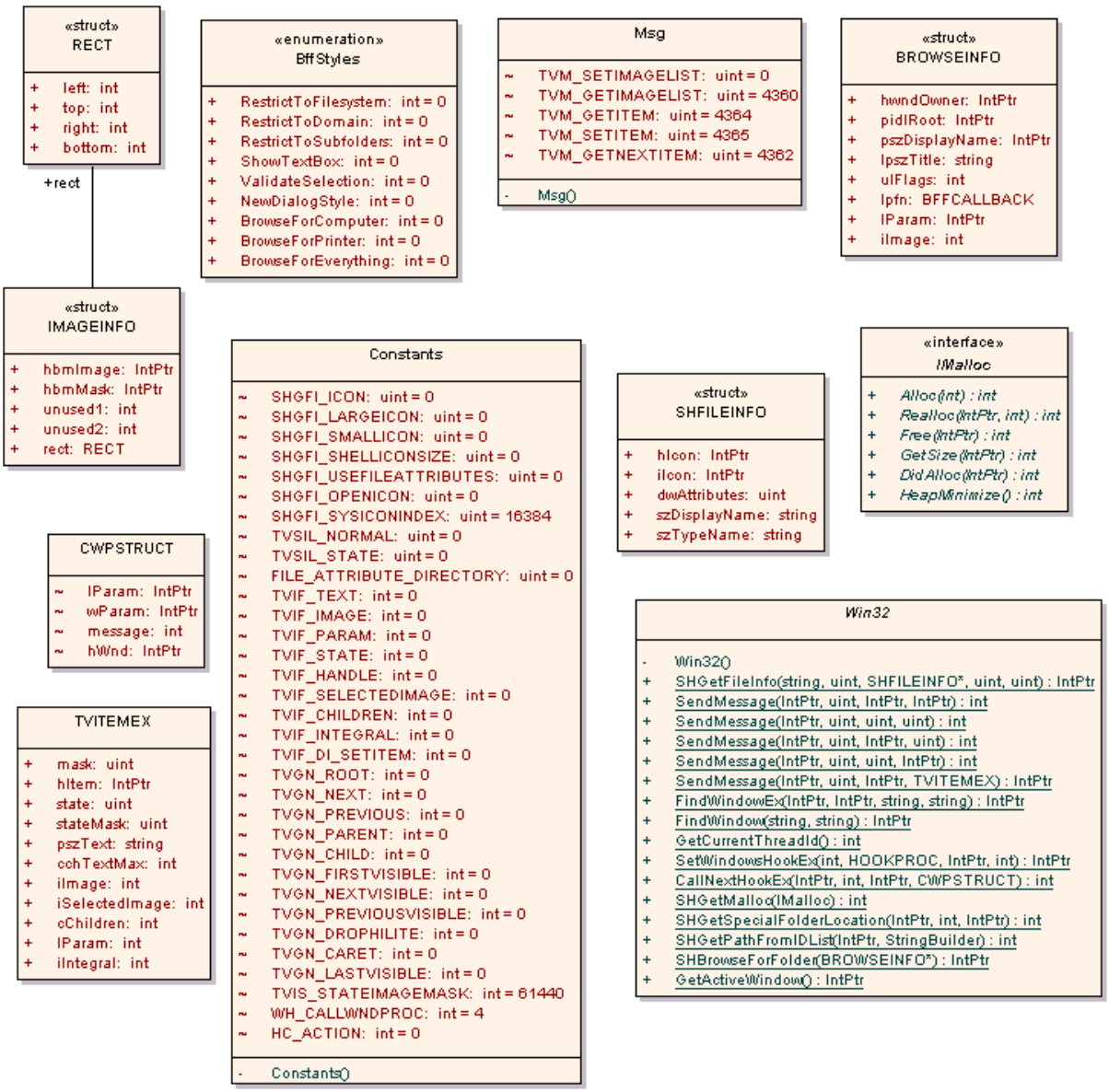

SimpleLogMessageProvider
- message: string
+ SimpleLogMessageProvider(string)
+ GetLogMessage(CommitItem[]) : string
«interface»
ILogMessageProvider
+ GetLogMessage(CommitItem[]) : string
+context
NSvnContext
- clientContext: ClientContext
+ NSvnContext()
+ AddAuthenticationProvider(IAuthenticationProvider) : void
+ RemoveAuthenticationProvider(IAuthenticationProvider) : void
# «property» ClientContext() : ClientContext
# NotifyCallback(Notification) : void
# LogMessageCallback(CommitItem[]) : string
«interface»
ILocalResourceVisitor
+ VisitUnversionedDirectory(UnversionedDirectory) : void
+ VisitUnversionedFile(UnversionedFile) : void
+ VisitWorkingCopyDirectory(WorkingCopyDirectory) : void
+ VisitWorkingCopyFile(WorkingCopyFile) : void
RepositoryResource
- url: string
- revision: Revision
RepositoryResource(string, Revision)
RepositoryResource(string)
+* Accept(IRepositoryResourceVisitor) : void
+ «property» Url() : string
+* «property» Name() : string
+ «property» Revision() : Revision
LocalResourceVisitorBase
+ VisitUnversionedDirectory(NSvn.UnversionedDirectory) : void
+ VisitUnversionedFile(NSvn.UnversionedFile) : void
+ VisitWorkingCopyDirectory(NSvn.WorkingCopyDirectory) : void
+ VisitWorkingCopyFile(NSvn.WorkingCopyFile) : void
+ VisitWorkingCopyResource(WorkingCopyResource) : void
+ VisitUnversionedResource(UnversionedResource) : void
RepositoryDirectory
- NAME: Regex = {}
+ RepositoryDirectory(string)
+ RepositoryDirectory(string, Revision)
+ GetChildren() : RepositoryResourceDictionary
+ Checkout(string, bool) : WorkingCopyDirectory
+ Checkout(string, string, Revision, bool) : WorkingCopyDirectory
+ Accept(IRepositoryResourceVisitor) : void
+ «property» Name() : string
RepositoryFile
+ RepositoryFile(string, Revision)
+ RepositoryFile(string)
+ «property» Name() : string
+ Accept(IRepositoryResourceVisitor) : void
EventArgs
NotificationEventArgs
- notification: Notification
+ NotificationEventArgs(Notification)
+ «property» Notification() : Notification
EventArgs
LogMessageEventArgs
- logMessage: string
- canceled: bool = false
- commitTargets: CommitItem[]
+ LogMessageEventArgs(CommitItem[])
+ «property» LogMessage() : string
+ «property» Canceled() : bool
+ «property» CommitTargets() : CommitItem[]
IDictionaryEnumerator
LocalResourceDictionaryEnumerator
- innerEnumerator: IDictionaryEnumerator
~ IDictionaryEnumerator: object
~ LocalResourceDictionaryEnumerator(LocalResourceDictionary)
+ «property» Key() : string
Utils
~ WC_ADMIN_AREA: string = .svn
- Utils()
+ GetWorkingCopyRootedPath(string) : string
+ GetParentDir(string) : string
+ IsWorkingCopyPath(string) : bool
IDictionary
ICollection
IEnumerable
ICloneable
RepositoryResourceDictionary
innerHash: Hashtable
~ IDictionary: System.Collections.IDictionaryEnumerator
+ RepositoryResourceDictionary()
+ RepositoryResourceDictionary(RepositoryResourceDictionary)
+ RepositoryResourceDictionary(IDictionary)
+ RepositoryResourceDictionary(int)
+ RepositoryResourceDictionary(IDictionary, float)
+ RepositoryResourceDictionary(IHashCodeProvider, IComparer)
+ RepositoryResourceDictionary(int, int)
+ RepositoryResourceDictionary(IDictionary, IHashCodeProvider, IComparer)
IDictionaryEnumerator
RepositoryResourceDictionaryEnumerator
- innerEnumerator: IDictionaryEnumerator
~ IDictionaryEnumerator: object
~ RepositoryResourceDictionaryEnumerator(RepositoryResourceDictionary)
+ «property» Key() : string

7.4.7 Class diagram NSvn.Core




System
NotifyCallback
+ NotifyCallback(System::Object*, System::IntPtr)
+* Invoke(NSvn::Core::Notification) : void
+* BeginInvoke(NSvn::Core::Notification, System::AsyncCallback*, System::Object*) : System::IAsyncResult*
+* EndInvoke(System::IAsyncResult*) : void
NSvn
ParameterDictionary
- innerDictionary: IDictionary*
- aprHash: apr_hash_t*
- pool: apr_pool_t*
+ ParameterDictionary(apr_hash_t*, apr_pool_t*)
# get_InnerDictionary() : IDictionary*
- CreateInnerDictionary() : void
Revision
+ Unspecified: Revision* const
+ Committed: Revision* const
+ Previous: Revision* const
+ Base: Revision* const
+ Working: Revision* const
+ Head: Revision* const
- date: DateTime
- revision: int
- kind: svn_opt_revision_kind
+ FromNumber(int) : Revision*
+ FromDate(DateTime) : Revision*
+ ToSvnOptRevision(Pool&) : svn_opt_revision_t*
- Revision(svn_opt_revision_kind)
StringHelper
- __gc: String
- string: gcroot<String*>
- char: mutable
+ StringHelper(String)
+ StringHelper(char*)
+ operator String*() : String*
+ StringHelper(StringHelper&)
+ operator =(StringHelper&) : StringHelper&
+ CopyToPoolUtf8(apr_pool_t*) : char*
+ CopyToPool(apr_pool_t*) : char*
+ operator constchar*() : constchar*
+ ~StringHelper()
- ConvertToCharPtr(String) : char*
- FreeCharPtr(char*) : void
T:class
Pool
- ownsPool: bool
- pool: apr_pool_t*
+ Pool()
+ Pool(apr_pool_t*, bool)
+ ~Pool()
+ operator apr_pool_t*() : apr_pool_t*
+ Alloc(int) : void*
+ PCalloc(int) : void*
+ AllocateObject(T&) : T*
+ AllocateObject(T&, apr_pool_t*) : T*
- Pool(Pool&)
- operator =(Pool&) : Pool&
«enumeration»
Schedule
+ Normal: int
+ Add: int
+ Delete: int
+ Replace: int
«enumeration»
NotifyState
+ Inapplicable: int = 0
+ Unknown: int = 0
+ Unchanged: int = 0
+ Changed: int = 0
+ Merged: int = 0
+ Conflicted: int = 0
«enumeration»
NotifyAction
+ Add: int = svn_wc_notify_add
+ Copy: int = svn_wc_notify_copy
+ Delete: int = svn_wc_notify_delete
+ Restore: int = 0
+ Revert: int = svn_wc_notify_revert
+ FailedRevert: int = 0
+ Resolve: int = 0
+ Status: int = svn_wc_notify_status
+ Skip: int = svn_wc_notify_skip
+ UpdateDelete: int = 0
+ UpdateAdd: int = 0
+ UpdateUpdate: int = 0
+ UpdateCompleted: int = 0
+ UpdateExternal: int = 0
+ CommitModified: int = 0
+ CommitAdded: int = 0
+ CommitDeleted: int = 0
+ CommitReplaced: int = 0
+ CommitPostfixTxDelta: int = 0
NSvn
UsernameCredential
+ LoggedInUser: UsernameCredential* const
- username: String*
+ UsernameCredential(String*)
+ get_Kind() : String*
+ GetCredential(IntPtr) : IntPtr
T:class
ManagedPointer
- ptr: gcroot<T>
+ ManagedPointer(T)
+ ManagedPointer(ManagedPointer<T>&)
+ operator =(ManagedPointer<T>&) : ManagedPointer&
+ operator =(T) : ManagedPointer&
+ operator T(T) : T
+ operator void*() : void*
Notification
- path: String*
- action: NSvn::Core::NotifyAction
- nodeKind: NSvn::Core::NodeKind
- mimeType: String*
- contentState: NSvn::Core::NotifyState
- propertyState: NSvn::Core::NotifyState
- revisionNumber: int
+ Notification(char*, svn_wc_notify_action_t, svn_node_kind_t, char*, svn_wc_notify_state_t, svn_wc_notify_state_t, svn_revnum_t)
+ get_Path() : String*
+ get_Action() : NotifyAction
+ get_NodeKind() : NodeKind
+ get_MimeType() : String*
+ get_ContentState() : NotifyState
+ get_PropertyState() : NotifyState
+ get_RevisionNumber() : int
System
LogMessageReceiver
+ LogMessageReceiver(System::Object*, System::IntPtr)
+* Invoke(NSvn::Core::LogMessage) : void
+* BeginInvoke(NSvn::Core::LogMessage, System::AsyncCallback*, System::Object*) : System::IAsyncResult*

7.4.8 Class diagram of Ankh.Utils



7.5 GUI not implemented in Ankh

Due to limited time available the following dialogboxes were not implemented in Ankh.

7.5.1 PropertyEditorDialog

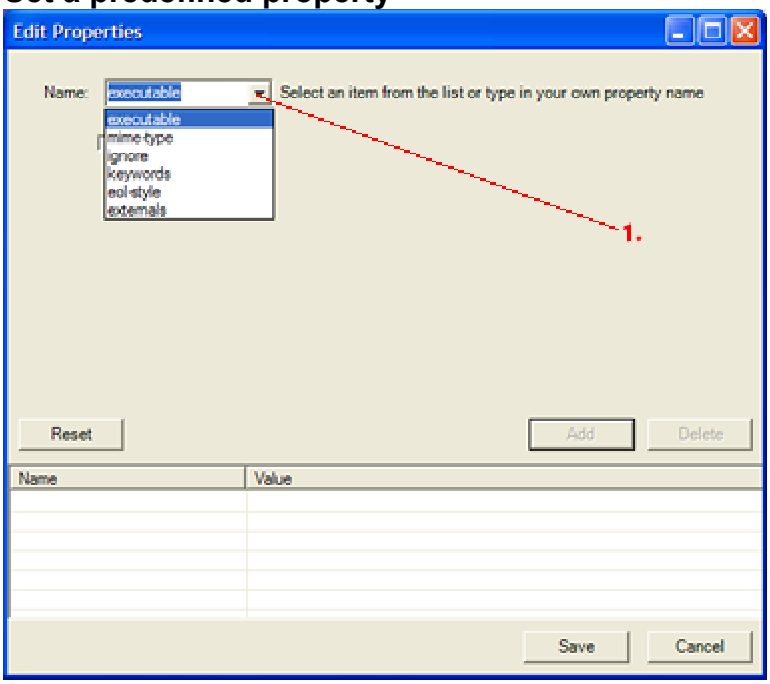
The following description was meant for Help.doc.

Setting properties
Here you will learn how to:

- set a predefined property
- create your own property
- edit and/or delete properties

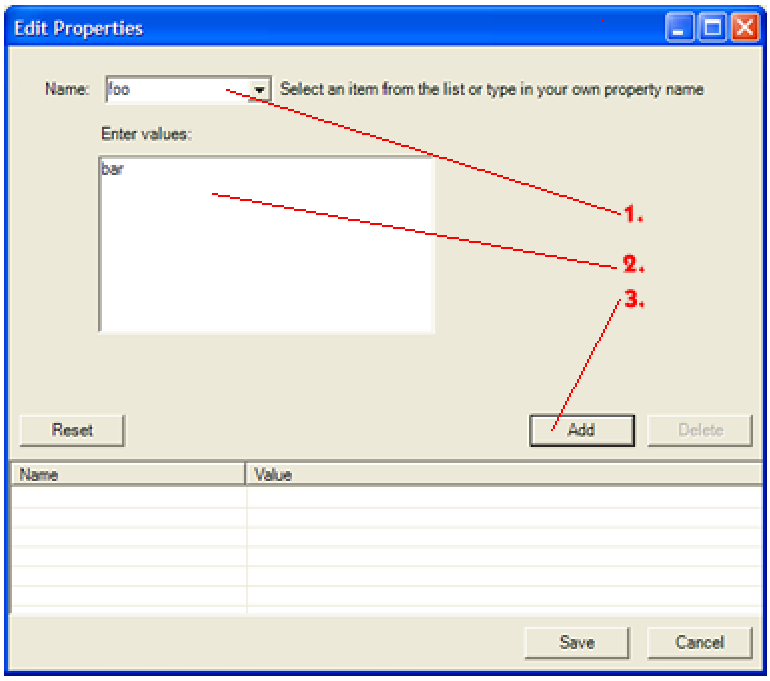
Select a file from the 'Solution Explorer', right click it and select 'edit properties'. You are now ready to set, edit or delete properties. If the file you selected already had a property, the list will populate itself. If you are uncertain what a property is, and how it works; look under the 'keywords' section under Properties

Set a predefined property



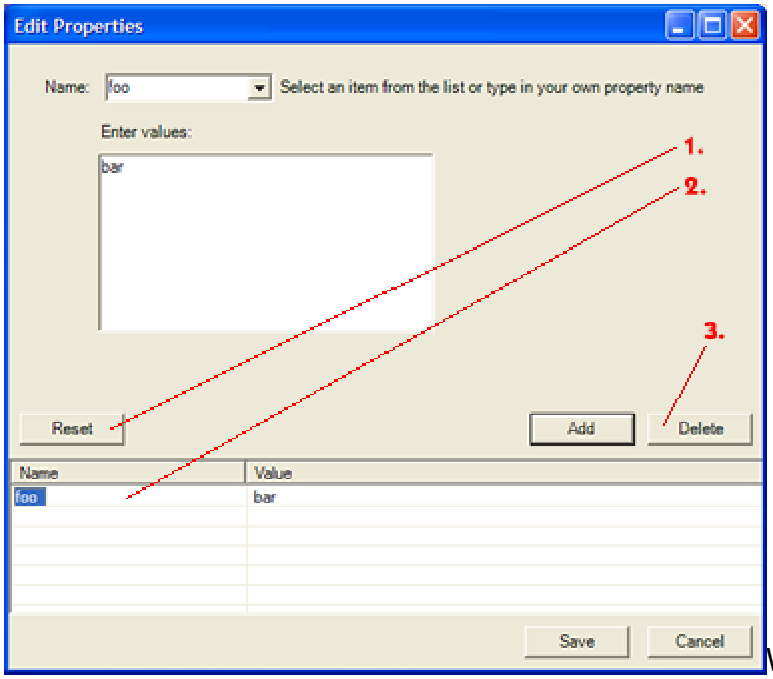
- 1. Select a item from the dropdown menu. Or you can define your own property by typing it directly in the "name" field. See next picture.

Creating your own properties



- 1. Type in the name of the property
- 2. Type in the value(s) of the property. Use a new line (ENTER) to differentiate between values.
- 3. Press 'Add' to store the property and the value(s). The property will be shown in the list at the bottom

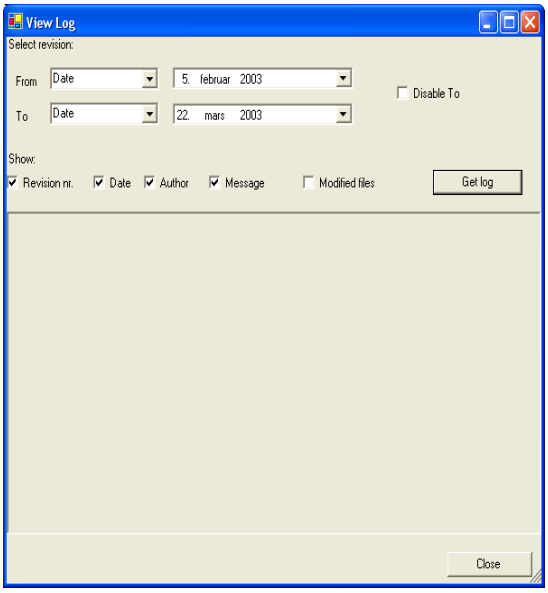
Changing and/or deleting properties



- 1. Resets all of the properties.
- 2. Select the property you want to work with from this list by double clicking it. The appropriate property screen will automatically be loaded, with values. Make your changes. And store them with 'Add'.
- 3. Deletes the property selected from the list (2.)

When you are finished defining all the properties, select 'Save'. All the properties are now stored and ready to use.

7.5.2 ViewLogDialog



7.5.3 ConflictDialog