



文档原始地址:

http://help.collab.net/index.jsp?topic=/com.collabnet.doc.anksvn_001/action/ankh_getting_started.html

翻译整理: 守护者 2010/5/23

<http://blog.csdn.net/shouhuzhe>

本版本为初级版本, 可能有很多错误。

如果有发现问题, 请邮件通知我: wuyaping_xp@126.com 以便于更新

引用请注明出处

目 录

1	AnkhSVN 概述.....	4
2	背景知识.....	5
2.1	什么是 AnkhSVN?.....	5
2.2	什么是版本库浏览器?	5
2.3	什么是工作拷贝浏览器?	6
2.4	什么是未提交更改视图 (the Pending Changes view)	7
2.4.1	什么是本地文件更改视图(Local File Changes view)	8
2.4.2	什么是问题视图 (the Issues view)	8
2.4.3	什么是最近修改视图?	9
2.4.4	什么是冲突合并视图.....	10
3	开始使用 AnkhSvn.....	10
3.1	安装 AnkhSVN.....	10
3.2	在 Visual Studio 中启用 AnkhSvn.....	10
3.3	将项目连接到 AnkhSVN.....	11
3.4	增加解决方案到版本库.....	12
3.5	浏览版本库.....	13
3.6	增加工作拷贝到工作拷贝浏览器.....	14
4	版本控制操作.....	14
4.1	签出解决方案.....	14
4.1.1	从 Subversion 打开解决方案.....	15
4.1.2	从版本库浏览器中签出.....	15
4.2	更新你的工作拷贝.....	16
4.3	在 Pending Changes 中执行 Subversion 操作.....	16
4.4	提交修改.....	17
4.4.1	提交你的修改.....	17
4.4.2	签入一个新项.....	18
4.5	获得和释放锁.....	18
4.5.1	锁定一个项.....	19
4.5.2	释放锁.....	19
4.6	修改取消.....	20
4.7	回滚一个项到特定的版本.....	20
4.8	查找什么人对其中的一行进行了修改.....	21
4.9	分支、标记和合并.....	22
4.9.1	创建一个分支或者标记.....	22
4.9.2	切换到分支.....	23
4.9.3	合并更改.....	24
4.10	比较不同.....	25
4.11	查看版本历史.....	27
4.12	创建和应用补丁 (patches)	28
4.12.1	创建一个补丁.....	28
4.12.2	应用补丁.....	29
4.13	将项增加到更改列表中.....	29

4.14	增加一个 Subversion 属性.....	30
4.15	清理工作拷贝.....	31
4.16	集成 issue tracker.....	32
4.16.1	连接到 issue tracker.....	32
4.16.2	将问题关联到一个 Commit.....	32
4.16.3	当提交提交时更新问题的状态.....	33
4.16.4	在查看历史时打开一个问题.....	33
4.17	配置和外部工具.....	34
附件 1、AnkhSVN 图标样例表.....		35
附件 2、Overview of CollabNet Merge Client.....		36
附件 3、Creating and Applying Patches.....		47
附件 4、Changelists.....		50

1 AnkhSVN 概述

AnkhSVN 为 Visual Studio 提供的一个 Subversion 源码控制插件。通过该 IDE 工具，你可以在 Visual Studio 中直接进行大部分版本控制操作。



你可以在 [AnkhSVN project on open.collab.net](http://AnkhSVN.project.on.open.collab.net) 下载 AnkhSvn 的 Release 版本以及日常开发版。



提示：本文档针对于 AnkhSvn2.0及更新的版本，支持 vs2005、2008、2010。

本文档包括 Richard LeBreton 提供的部分。

2 背景知识

本章对于使用软件需要用到的一些概念、定义以及最优方法进行了描述。

2.1 什么是 AnkSVN?

AnkhSVN 为 Visual Studio 提供源码控制。它允许你再 Visual Studio 环境中直接进行版本控制操作。

通过 AnkhSvn，你可以：

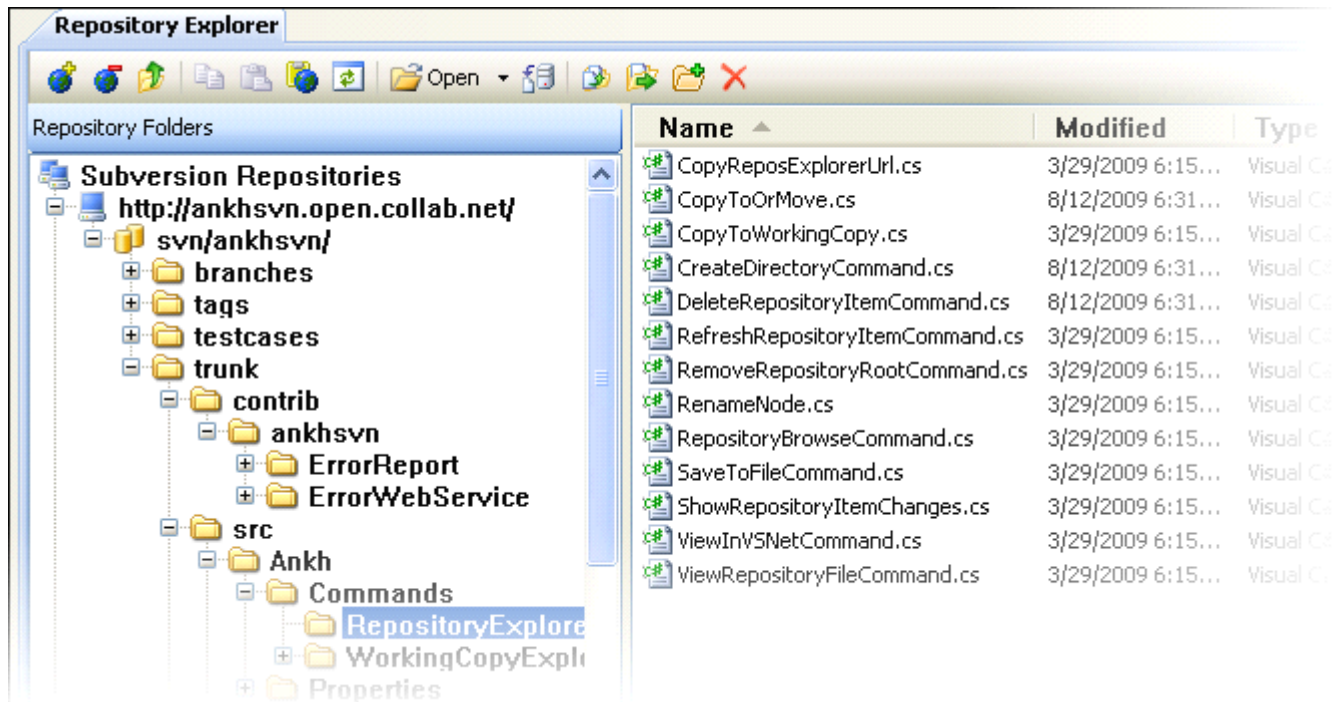
- 查看你项目或解决方案中所有文件和文件夹的源码控制状态
- 通过解决方案浏览器直接从 Subversion 版本库中导入新的解决方案
- 使用工作拷贝浏览器（the Working Copy Explorer）：
 - 与非项目或解决方案文件协同工作
 - 使用与解决方案浏览器相同的命令行
- 使用版本库浏览器（the Repository Explorer）：
 - 非常容易的浏览当前版本库的内容
 - Check out 或 查看一个文件夹
 - 在 Visual Studio 打开一个文件，通过浏览器查看或者拷贝该文件
- 通过 the Pending Changes window 实时查看文件的状态
- 提供 Subversion1.5 合并操作（Merge changesets and track merges based on the functionality provided by Subversion 1.5）
- 可自行选择 Diff 和 Merge 工具

2.2 什么是版本库浏览器？

版本库浏览器是一个显示你解决方案在 Subversion 版本库中文件和文件夹的操作窗口。

当你增加 Subversion 版本库的 URL 到版本库浏览器时，你可以：

- 非常容易的浏览版本库中的内容
- 签出活浏览一个文件夹
- 在 Visual Studio 打开一个文件，在浏览器中查看该文件，或者拷贝该文件
- 在 Visual Studio 属性窗口查看远程文件的扩展信息以及路径信息



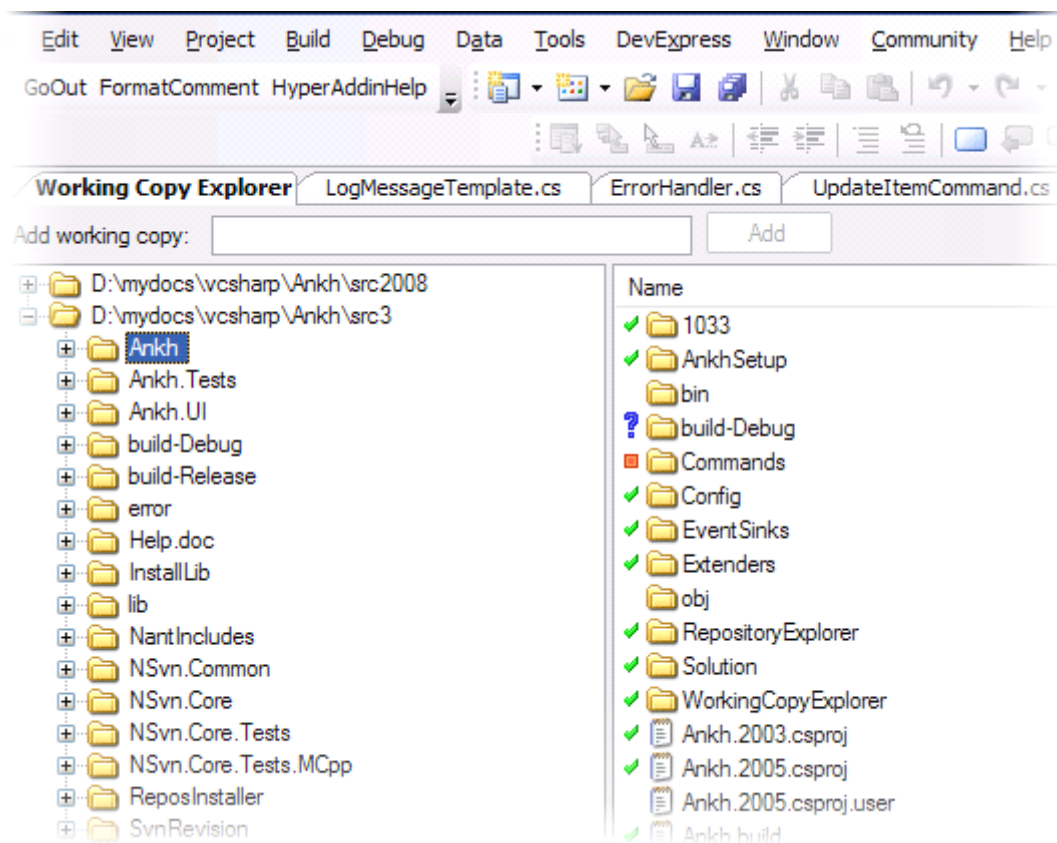
2.3 什么是工作拷贝浏览器？

工作拷贝浏览器允许你在 Visual Studio 中同时使用多个版本库的本地拷贝。甚至包括不属于项目或解决方案的文件和文件夹。

在工作拷贝浏览器，你可以：

- 查看文件或文件夹的版本控制状态
- 双击某个项目可以在工作盘中打开它（the task pane）
- 右键单击项目可以获得与解决方案浏览器中操作同样的源码控制操作菜单

以下是一张工作拷贝浏览器的样例图：



2.4 什么是未提交更改视图（the Pending Changes view）

The Pending Changes view provides a single window to handle your normal workflow. It gives you an overview of all project changes in real time, and easy access to most Subversion operations.

In Pending Changes, clicking an icon in the left panel displays one of these four views:

- Local File Changes
- Issues
- Recent Changes
- Conflicts and Merges

The top toolbar in Pending Changes lets you do the following:

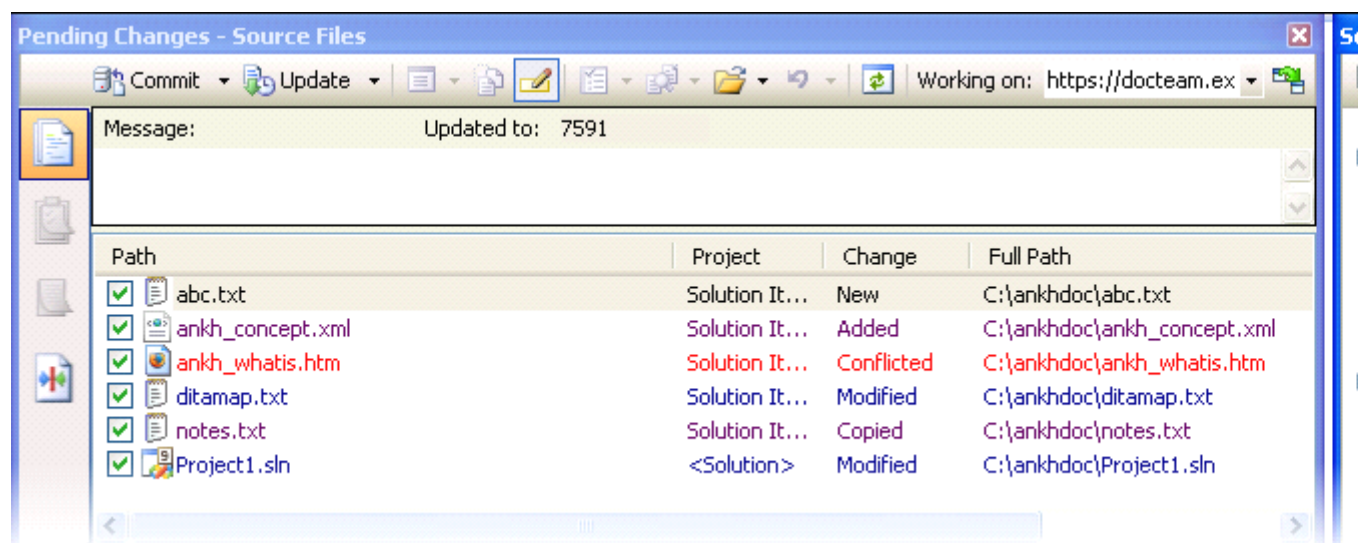
- Select individual files or all eligible ones and commit them in a single operation
- Branch, merge or switch a solution
- Update a file to a specific revision
- Compare the current version of a file to some other version
- In the Flat Changes view, see the changed files in your working copy at various hierarchical levels

- See the repository path you are currently working with, and switch to a different one if needed

2.4.1 什么是本地文件更改视图(Local File Changes view)

本地文件更改时显示待提交的所有源代码的状态。如果本地文件与版本库中版本相匹配，那么该文件将不会在这个视图显示。

以下例子图片，本图片表明了解决方案视图与本地文件更改视图一致。



文件状态:

新建 (New)

显示新增加但是还没有计划放入到版本库中的文件。

已增加 (Added)

显示已经增加，但是还没有提交到版本库的文件。

被编辑 (Edited)

显示正在编辑但是还没有保存的文件。

被编辑 (Modified)

显示已经编辑并前保存到本地的文件。

冲突 (Conflicted)

显示你从版本库更新产生冲突的文件。

删除、拷贝 (Deleted, Copied)

当你重命名一个文件，文件将拷贝一个新文件，而旧文件将标记为删除 (Deleted)。

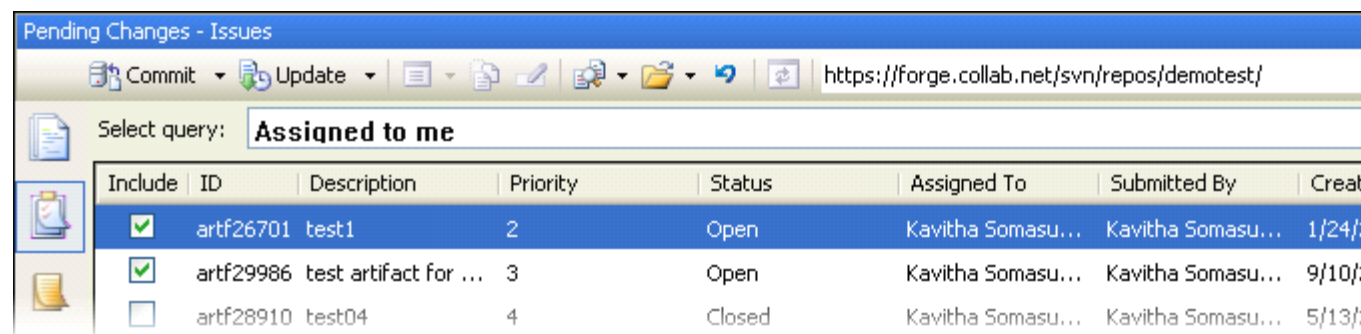
删除显示的是原始名称，而拷贝显示的是新的名称。

2.4.2 什么是问题视图 (the Issues view)

如果你为你的解决方案安装了 issue tracker 那么你可以在问题视图中查看提交的问题。

AnkhSVN 完全支持允许任何 issue tracker 进行插入。

以下是 CollabNet issue tracker 集成的例子：




提示：集成 CollabNet issue tracker 你需要进行以下操作：

- CollabNet Desktop - Visual Studio Edition version 1.5.9 or later
(<http://desktop-vs.open.collab.net/>)
- AnkhSVN version 2.1.7250 or later

For help on this feature, see [CollabNet Desktop for Visual Studio - Integrate an issue tracker](#).

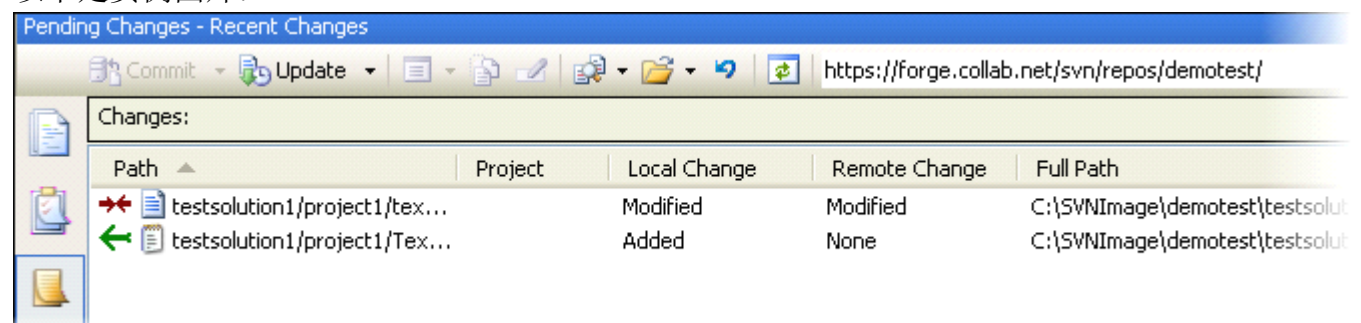
2.4.3 什么是最近修改视图？

最近修改视图显示了当前即将发生活刚刚发生的改变，相当于 `svn st -u` 操作

 提示：你需要点击"Refresh"按钮来刷新视图。

包括其他用户提交的变更（Incoming）和自己通过编辑、删除、增加所作的提交（Outgoing）。

以下是实例图片：



2.4.4 什么是冲突合并视图

当前，该功能正在完成中。。。

3 开始使用 AnkhSvn

本章将为你使用 AnkhSvn 提供一些最基本的议题。

3.1 安装 AnkhSVN

首先下载并运行 AnkhSVN 的安装包。

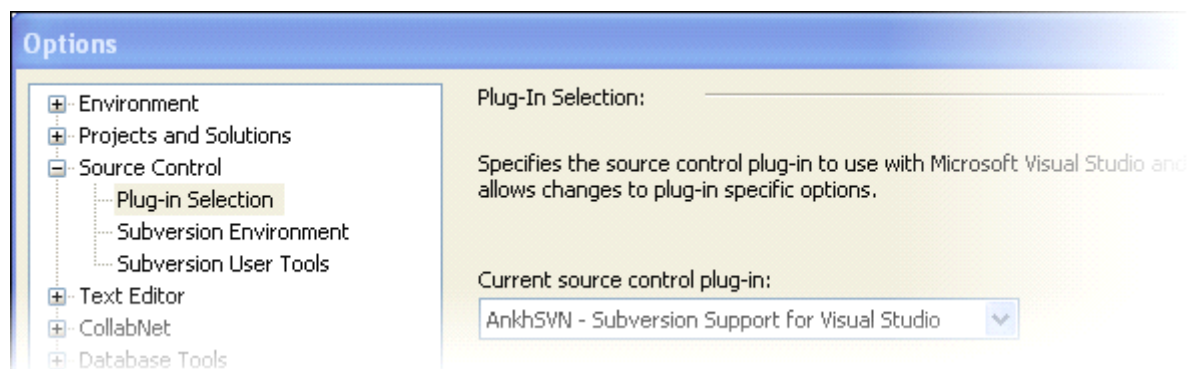
1. 从 <http://ankhsvn.open.collab.net> 下载 AnkhSVN 2.x 版本。
2. 运行安装包。

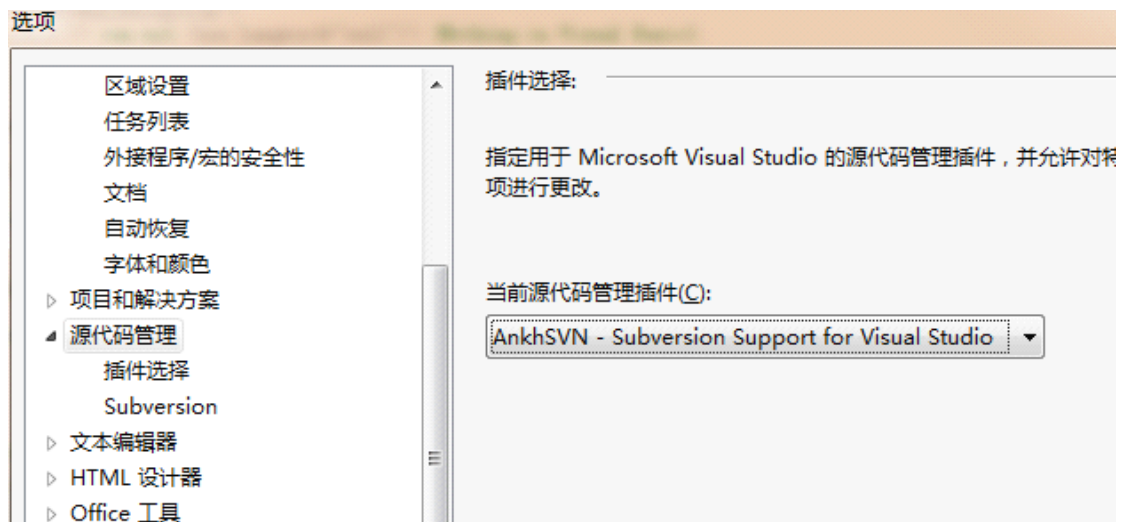
AnkhSVN 安装包是一 Visual Studio 使用 Subversion 源码控制的插件。

3.2 在 Visual Studio 中启用 AnkhSvn

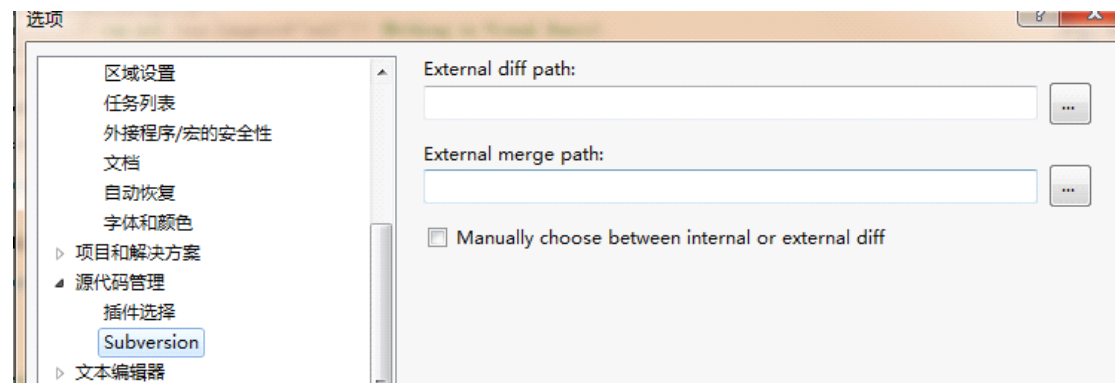
当你在 VS 开发环境中启用 AnkhSVN 后你就可以在 VS 环境中直接使用 IDE 界面进行源码管理操作。

1. 在 Visual Studio 中选择工具>选项（Tools > Options ）。
2. 在选项窗口中找到源代码管理。
3. 选择 AnkhSVN - Subversion Support for Visual Studio 作为当前的版本控制工具。





现在还可以配置 AnkhSVN 的不同选项

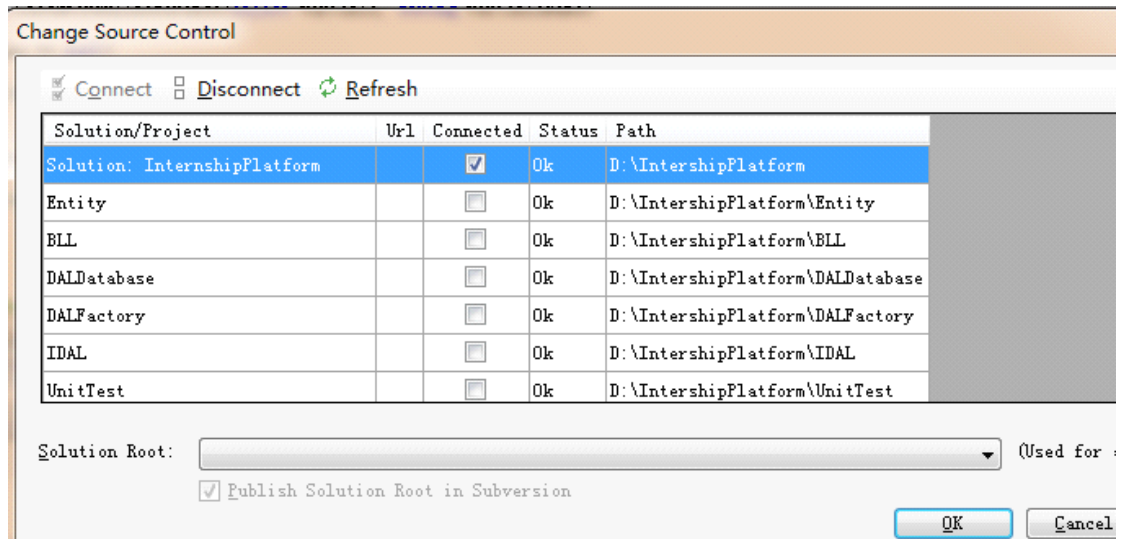


3.3 将项目连接到 AnkhSVN

当你的 VS 有多个版本控制系统并且你已经为你的项目选择了版本控制工具时，如果需要使用 AnkhSvn，第一步你需要将项目连接到 AnkhSvn。

1. 在解决方案浏览器中打开项目。
2. 在 VS 的文件菜单选择 Subversion > Change Source Control 。
3. 在 Change Source Control 窗口包含了你的项目和解决方案，选择，然后点击连接。
4. 点击 OK

例子图片：



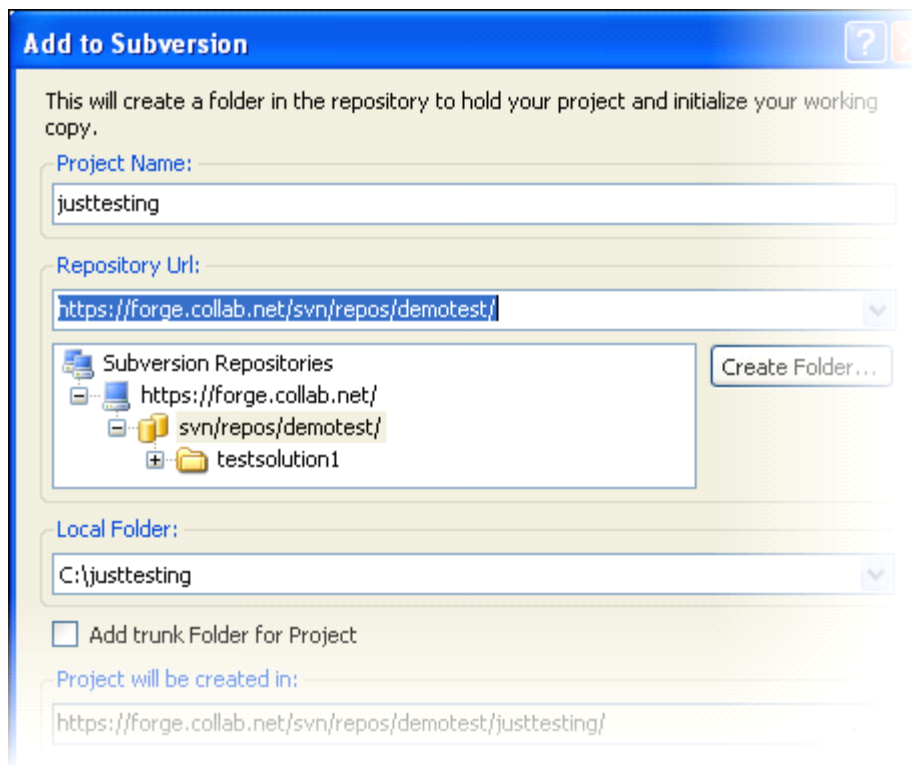
3.4 增加解决方案到版本库

开始版本控制之前，将新创建或已存在的项目或解决方案加入到 Subversion 版本库中。



重点提示：你首先需要在 VS 环境之外创建一个 Subversion 版本库，你可以使用类似 TortoiseSVN 这样的 Subversion 客户端工具或者通过命令 "svnadmin create <repository_path>" 来创建版本库。

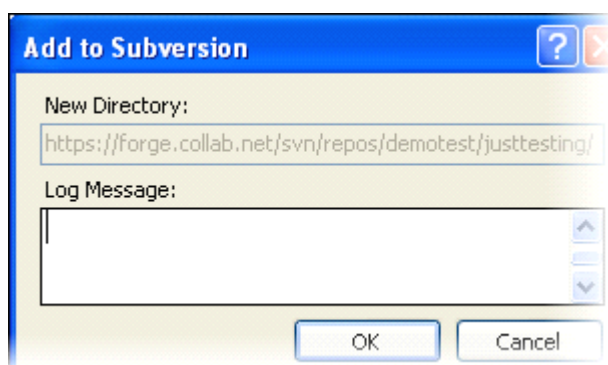
1. 在解决方案浏览器打开你的解决方案。
 - 如果你是新创建的项目或解决方案，将默认在解决方案浏览器中打开。
 - 如果是已经存在的窗口，通过文件>打开>项目/解决方案菜单打开。
2. 右击你的解决方案，选择 Add Solution to Subversion。如果增加特定的项目而不是解决方案，那么选择 Add Selected Projects to Subversion。点击后将出现 Add to Subversion 操作界面。你需要填入项目名称以及你的工作拷贝。



3. 填写相关细节：

- 制定你 Subversion 版本库的 URL。在下拉列表中选择或者输入，例子见：
`https://demo.collab.net/svn/repos`.
- 如果需要创建子目录，那么选择 **Create Folder**，然后输入目录名和日志信息。
- 如果你想要创建一个主线的分支，你可能需要创建一个 **trunk** 目录来保存主线，首先选择 **Add trunk folder for Project** 选项

点击 OK 按钮，点击后将出现如下窗口，提示你输入日志信息。

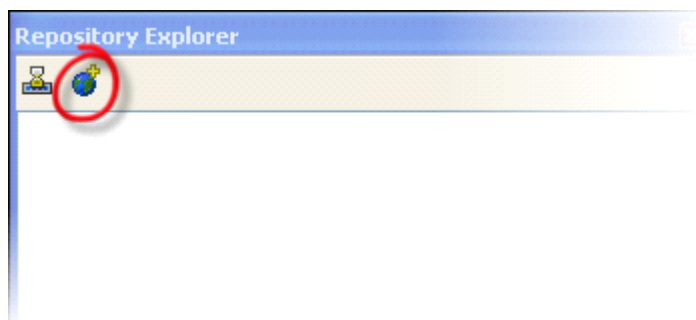


4. 输入日志信息并点 OK。

3.5 浏览版本库

如果想要以树形菜单查看版本库，你需要在版本库浏览器中增加 URL。

1. 在视图菜单中选择 Repository Explorer。
2. 在版本库浏览器中点击增加 URL 图标。



3. 在增加 URL 窗口中写入 Subversion 版本库的 URL 例如：
<https://Project1.collabnetdemo.net/svn/Project1>。
4. 制定版本库的一个版本，并点击增加。

增加后你可以展开树形菜单查看版本库中相应的内容。

3.6 增加工作拷贝到工作拷贝浏览器

为了方便的管理版本库的工作拷贝（包括不在版本库的文件），你可以将工作拷贝的路径增加到工作拷贝浏览器中。

1. 在视图菜单中选择 Working Copy Explorer。
2. 在工作拷贝浏览器中右击左边操作区，选择 Add new root 图标。
3. 输入本地工作拷贝的路径，点击 OK。

在工作拷贝浏览器中，你可以右击本地文件和文件夹进行与解决方案浏览器中相同的版本控制操作。

4 版本控制操作

对于普通的 Subversion 操作，右击解决方案浏览器中的项，选择 Subversion 并选择相应的操作命令即可。

4.1 签出解决方案

你可以使用文件菜单或者版本库浏览器来签出项目或解决方案到你的本地拷贝。

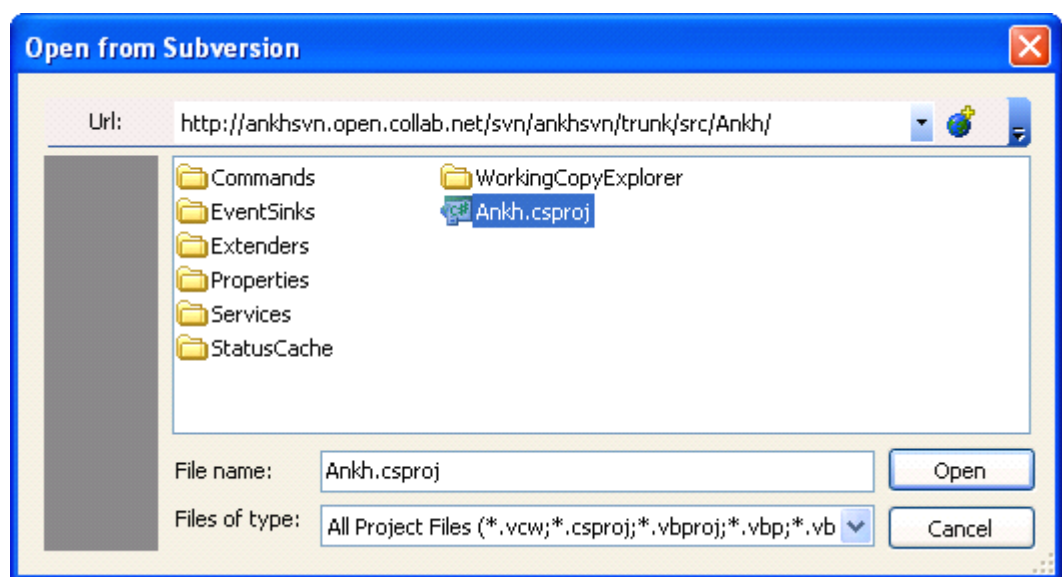
4.1.1 从 Subversion 打开解决方案

使用 Open from Subversion 菜单来签出受 Subversion 控制的项目或解决方案。

1. 从 VS 的文件菜单，选择 Subversion > Open from Subversion .

文件 > 打开 > Subversion Project 提供了同样的功能。

2. 在 Open from Subversion 窗口中指定 Subversion 版本库。
 - 在下拉列表中选择版本库的 URL。
 - 选择地球图标来增加一个新的 URL。



3. 选择项目或解决方案，点击 Open。将会出现 Open Project from Subversion 窗口。
4. 在 Local Directory 处写入你希望本地拷贝存放的位置，例如： C:/project1 。
5. 在 Version 处选择你需要签出的版本号。默认签出最新的版本。
6. 点击 OK。

之后项目或解决方案将在解决方案浏览器中打开。

4.1.2 从版本库浏览器中签出

从版本库浏览器中可以直接签出你的项目或解决方案并在解决方案浏览器中打开。

1. 在 visual Studio 中选择 视图 > Repository Explorer。
2. 点击地球图标并增加。
3. 选中你需要签出的项目点击鼠标右键并选择打开。
4. 在 Local Directory 处写入你希望本地拷贝存放的位置，例如： C:/project1 。

5. 在 Version 处选择你需要签出的版本号。默认签出最新的版本。
6. 点击 OK。

之后项目或解决方案将在解决方案浏览器中打开。

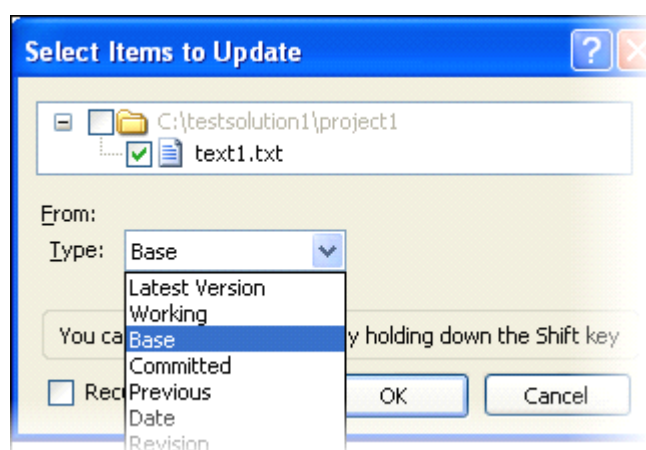
4.2 更新你的工作拷贝

通过更新操作是你的工作拷贝中的项目于版本库中同步。

当你在解决方案中的项上进行工作式，其他用户可能对这些项进行了更改，你需要通过更新来查看这些修改。

在结局方案浏览器，你可以更新到最新或者指定的版本：

- 右击选择 Update to Latest Version 更新到最新版本。
- 更新到特定版本，右键点击需要更新的项，选择 Subversion > Update to Specific Version 并指定一个版本。



当其他用户和你共同修改了同一个文件，并且对方已经修改的时候，Subversion 将尝试对修改进行合并。如果你们修改的部分没有重叠的，那么将自动合并，例如两个用户更改了同一文件的不同行。如果你们的修改产生了冲突，那么工作拷贝将显示唯一一个惊叹号图标来指示产生了冲突，例如两个用户修改了同一个文件的相同行。

4.3 在 Pending Changes 中执行 Subversion 操作

Pending Changes 视图时 Subversion 操作的一个中心。对于 Pending Changes 中的文件或文件夹，你可以像在解决方案浏览器中一样执行所有的 Subversion 操作。

1. 通过以下两种方式打开 pending changes 视图：
 - 文件 > Subversion > Pending Changes .
 - 视图 > Pending Changes.

2. 在 Local File Changes 或 Recent Changes 视图中对一个项执行 Subversion 操作，右击项并选择相应的选项。

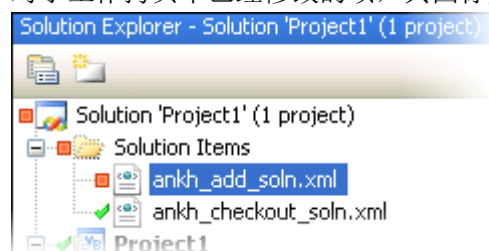
4.4 提交修改

为了正式保存你的修改，你需要将其提交到版本库中。

4.4.1 提交你的修改

在解决方案浏览器中点击鼠标右键并选择 Commit。

对于工作拷贝中已经修改的项，其图标为一个红色的狂，如下：



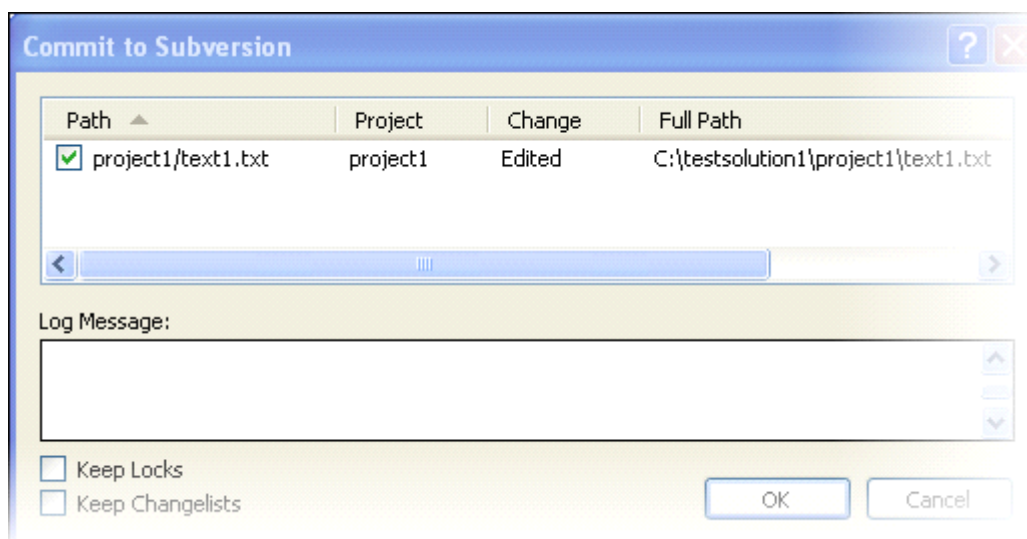
提示：当你完成对一个项的工作后，建议你在提交变更前首先更新，这样，其他用户的提交将会自动合并到你的提交中。

1. 在解决方案浏览器中右击你修改的项并选择 Commit。



建议：在你提交之前，你可能想要你工作拷贝与基线版本之间的差异，你可以右击项然后选择 Show Changes 来进行比较。

2. 在 Commit to Subversion 窗口，输入你本次修改的日志细节，点击 OK。



提示:



- 当你提交修改到版本库, 你的锁将自动释放, 你也可以选择 **Keep Locks** 选项。
- 如果你使用修改列表来组织你的文件, 提交将自动从修改列表中移除文件, 你可以选择 **Keep changelists** 来避免。

当你的修改提交之后, 解决方案浏览器中一个蓝色的勾将被标记 (比较老的版本是绿色的)。

4.4.2 签入一个新项

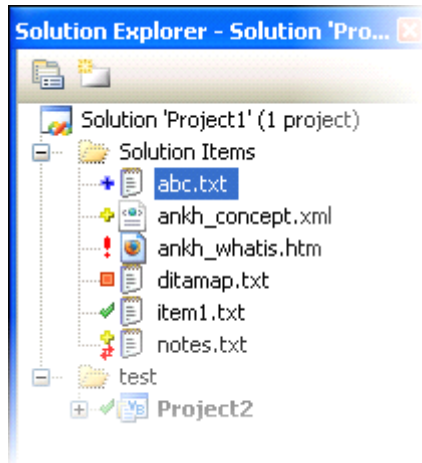
如果想增加一个新项到源码管理中, 你需要首先 Add 然后 Commit。

1. 在解决方案浏览器中右击项目, 选择 **Subversion > Add**。



提示: 新项用一个蓝色的加号图标标示。

2. 在 **Select items to add** 窗口, 确保你需要增加的文件已经选择, 然后点击 **OK** 按钮, 之后增加的项将会出现一个黄色的加号图标, 这表示图标已经在版本库的增加序列中了。



3. 选择该项, 右击并选择 **Commit** 菜单。

成功增加到版本库后, 图标将变成一个蓝色的对号 (原来是绿色)。

4.5 获得和释放锁

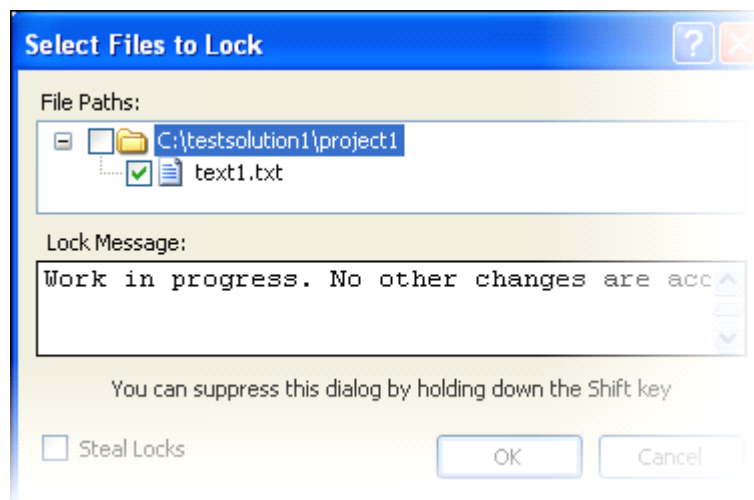
如果 Subversion 采用 copy-modify-merge 模式进行版本控制的话, 这可能需要用户锁定一个文件, 并且只有该文件能够提交该文件的修改。

4.5.1 锁定一个项


当你需要确保只有你自己能够修改和提交一个项时，你可以先锁定它。

对于这种情形的一个例子是比如你正在处理的文件不能够被合并，例如图片或二进制文件。直到你释放锁，其他用户才能提交他们的修改。

1. 在解决方案浏览器中右击一个文件，然后选择 Subversion > Lock。
2. 在 Select Files to Lock 窗口输入你锁定文件的理由。
3. 点击 OK。



4. 如果需要获得其他用户锁定的文件，请选择 Steal Locks 选项。

当该操作结束，解决方案浏览器中该文件上将出现一个小锁。  item1.txt



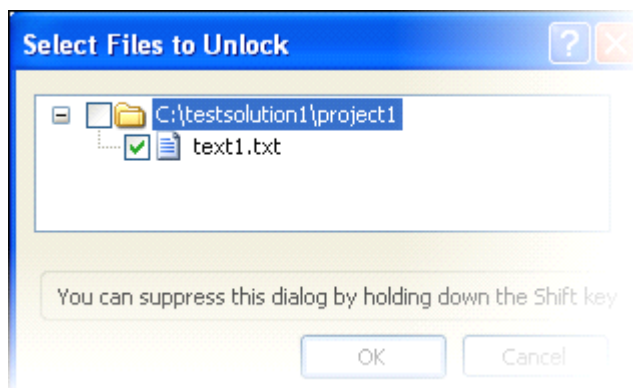
提示：当你提交锁定项后，锁将自动释放，如果你需要保持锁，请在提交的时候选择 Keep locks 选项。

4.5.2 释放锁

如果需要允许其他用户修改和提交你锁定的项，那么首先通过 Unlock 来释放你的锁。

当你提交锁定项的时候，通常锁将自动释放。但是如果你提交时选择了 Keep Locks 选项，那么锁将不会被释放。这种情况下你需要手动释放锁。

在解决方案浏览器中右击锁定的项，然后选择 Subversion > Unlock。



当操作结束后，锁图标将本移除。

4.6 修改取消

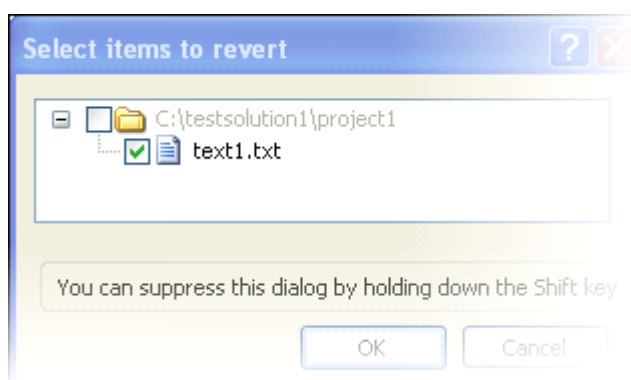
如果你需要将你本地拷贝恢复到最后更新状态，选择 Revert 操作。

但是这个操作将对你提交的操作产生任何影响。这个操作只会回滚你本地的修改，并且回复文件到最后更新时的状态。如果需要回滚到一个特定版本，你需要使用 Revert to Revision 选项。

1. 在解决方案浏览器中右击一个项，然后选择 Revert 将出现 Revert 窗口，窗口中显示了你最后更新之后所做的所有修改。
2. 在 Select items to revert 窗口中选择你需要回滚的项。
3. 点击 OK.



提示：由于 Revert 操作时无法回复的，你最好在 Revert 之前对你的本地拷贝进行备份。

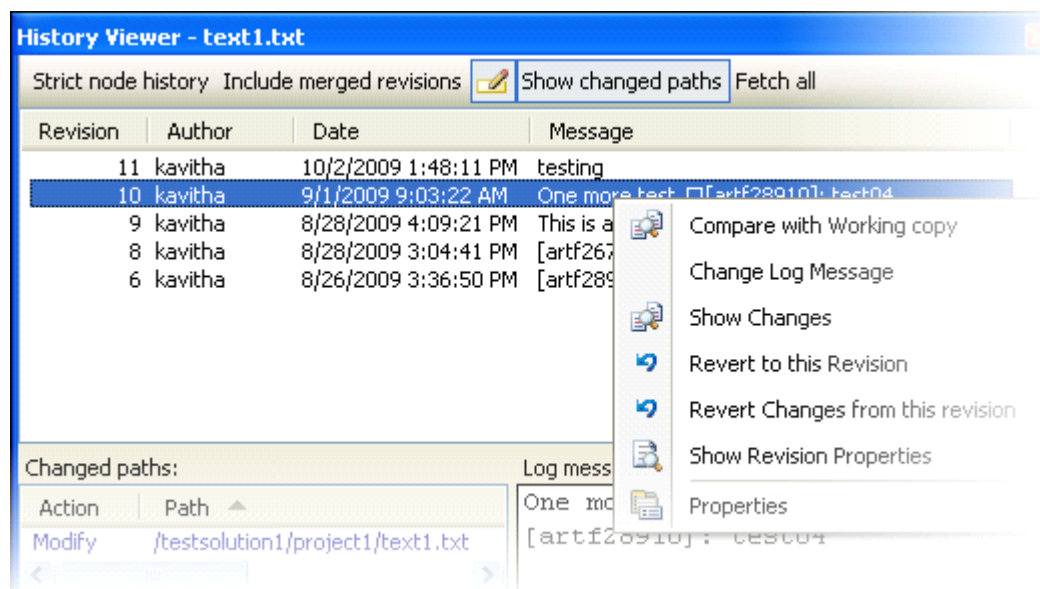


4.7 回滚一个项到特定的版本

如果需要回滚一个项到特定的版本，你需要使用在 History Viewer 使用 Revert to this revision 选项。

如果回滚你的本地拷贝到最后更新的状态，使用 **Revert** 操作即可。

1. 在解决方案浏览器中右击项，选择 **View History**。
2. 在 **History Viewer** 中右击你需要回滚到的版本，选择 **Revert to this Revision**。



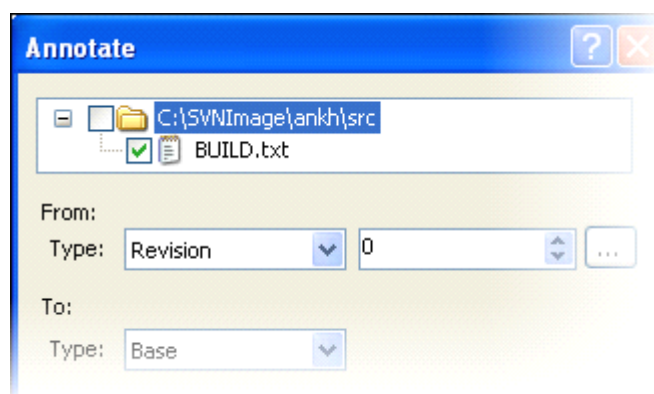
项将回滚到你选择的版本。

4.8 查找什么人对其中的一行进行了修改

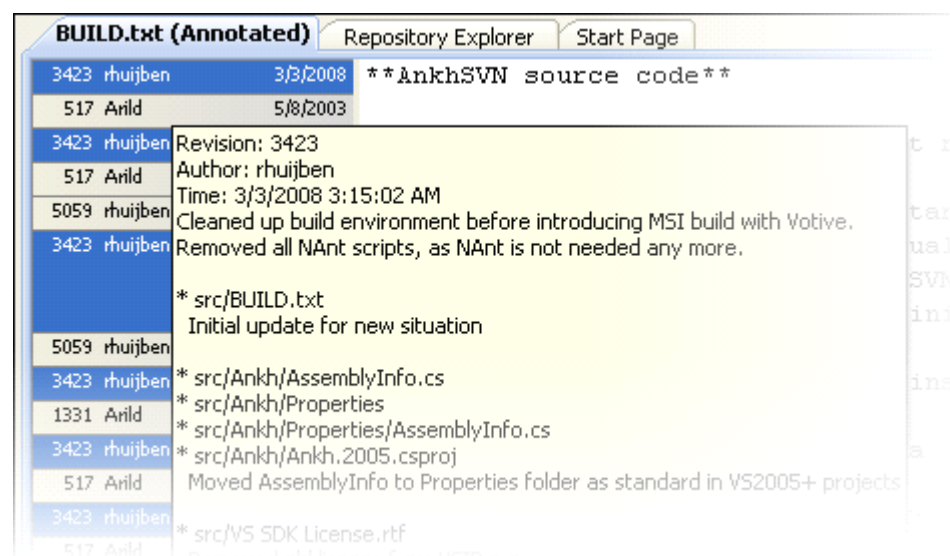
如果你要知道是什么人对文件中的某一行代码进行了修改以及谁创建的，使用 **Annotate**（注解）操作。

在 AnkhSvn 的早期版本，该操作称为 **Blame**（责任）。

1. 在解决方案浏览器右击项，选择 **Subversion > Annotate**。
2. 在 **Annotate** 窗口，制定你要查看的起始和终止版本。
3. 点击 **OK**。



将自动创建一个缓存文件包含作者、版本号、时间戳以及其他细节。例如：



4.9 分支、标记和合并

Subversion 提供项目中单独行的支持，这些行在版本库中式独立保存的，但是你可以进行比较、拷贝以及合并。

分支（Branching）

分支起始的时候是主线的一个拷贝，但是其后他将保持独立性，拥有自己的变化。有时你需要创建一个分支，例如你需要开发一个新功能，但是你不希望新功能开发完成之前影响主线的开发。

标记（Tagging）

标记是版本库中一个特定版本的快照。通常当项目进展到里程碑或者你预备发布 release 版本的时候你需要创建一个标签。虽然也可以对标记进行 Commit，但是一般都不这么做。

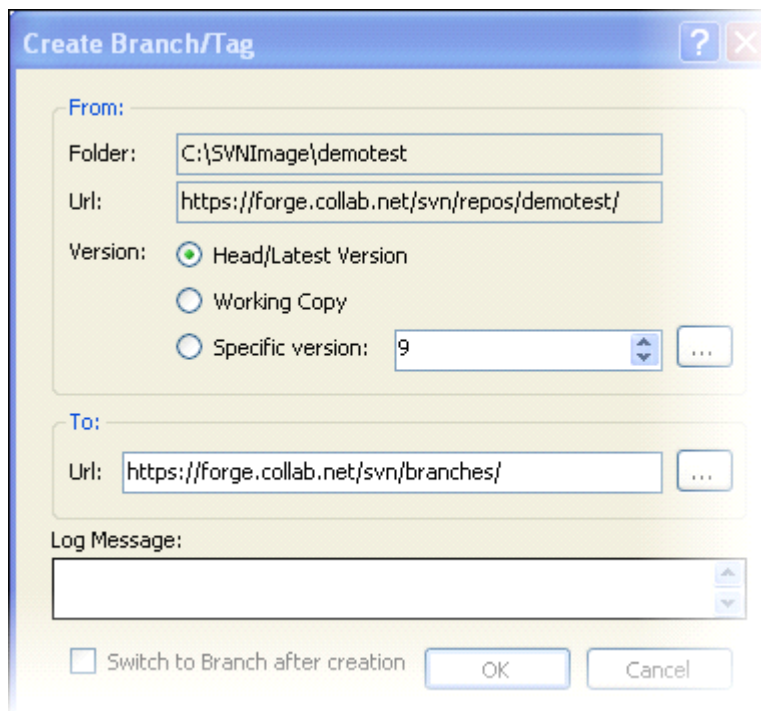
合并（Merging）

当分支开发进行到某个特定场景的时候，通常需要将更改合并到主线中。例如，如果开发的心功能已经稳定，那么分支就可以合并到主干了。

4.9.1 创建一个分支或者标记

创建标记的步骤与创建分支的步骤一致。Subversion 对于两者的处理都是使用“cheap copy”或者内部链接的方式指定到特定版本。两者之间的区别在于如何使用它们。

1. 在解决方案浏览器中，右击一个项，然后选择 Subversion > Branch Solution.。



2. 在 Create Branch/Tag 窗口中为你的分支或标记选择一个源作。

- 版本库中的”HEAD or latest revision”。
- 版本库中一个指定的较老版本
- 你的本地拷贝



提示：如果你不知道具体的版本号，点击右边的按钮来启动日志查看窗口，然后选择合适的版本号。

3. 如果需要自动切换你的工作拷贝到新的分支，选择 Switch to Branch after creation.。



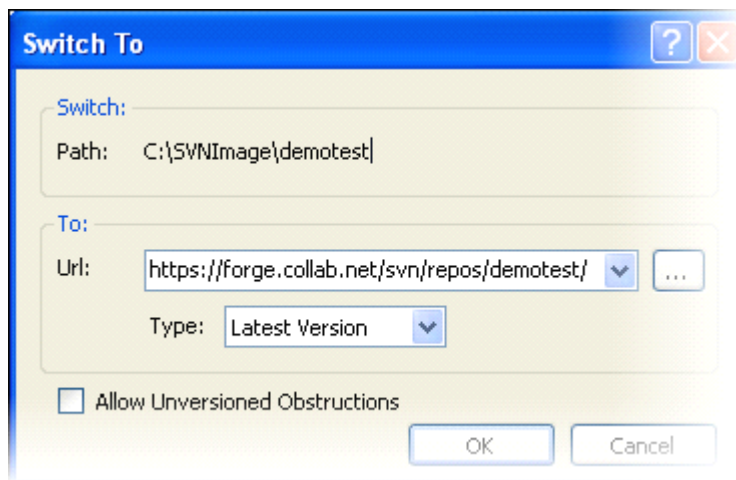
提示：如果你的工作拷贝已经修改，那么当你切换的时候这些修改将合并到工作拷贝中。

4. 输入你为什么创建分支或标签的日志。
5. 点击 OK.

4.9.2 切换到分支

如果需要从你当前的工作拷贝切换到分支或标记，使用 Switch Solution 菜单。

1. 在解决方案浏览器中右击一个项，选择 Subversion > Switch Solution。



2. 在 Switch to 窗口中输入分支的 URL 或者你需要切换的版本号。
3. 点击 OK.

操作之后你的工作拷贝将切换到分支或标记。

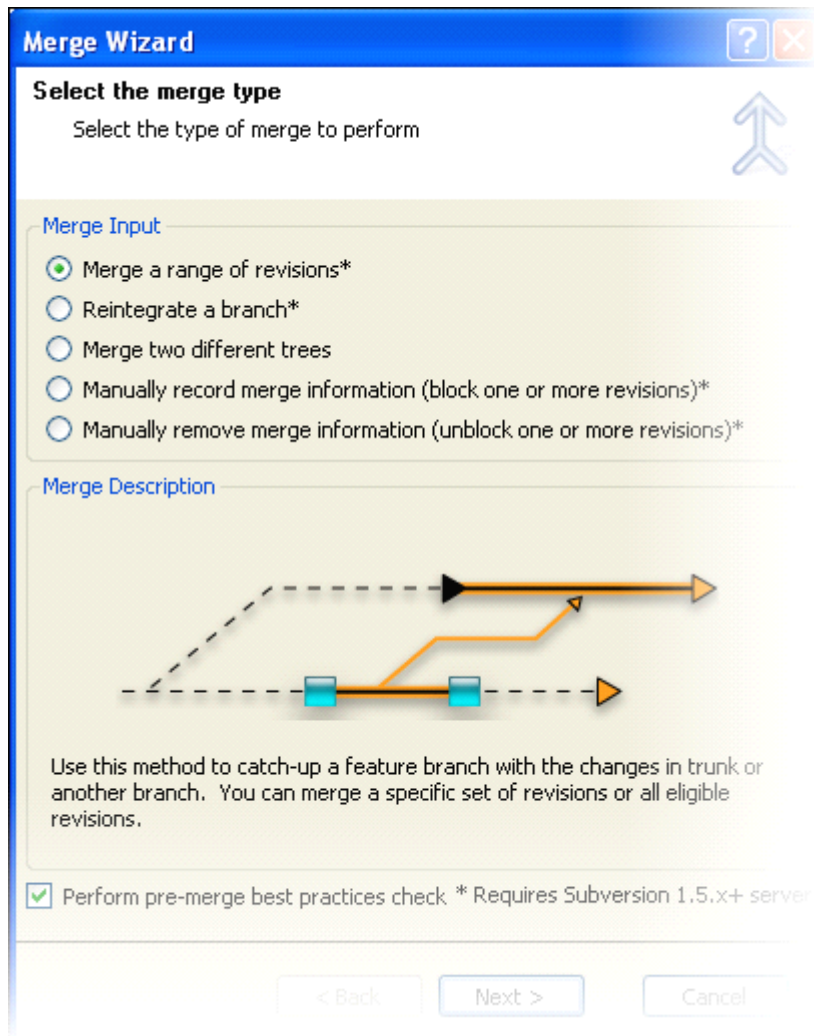


提示：如果你已经对你的工作拷贝做了修改但是还没有提交，那么这些修改将合并到你切换的工作拷贝中。

4.9.3 合并更改

如果你的项目包含多个开发线，在某个时间点，你可能需要将你的分支合并到主干或者其他分支。

- 在解决方案浏览器右击解决方案，选择 Subversion > Merge Solution。

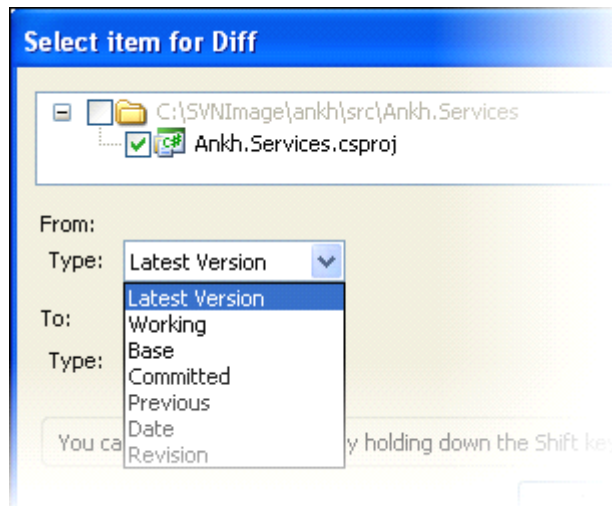


- 对于合并分支的更多介绍请参看["Performing a Merge"](#)（附件 1）。文档时针对 Eclipse 的操作，但是与 AnkhSvn 相似，只是少了 changeset-based merge 。

4.10 比较不同

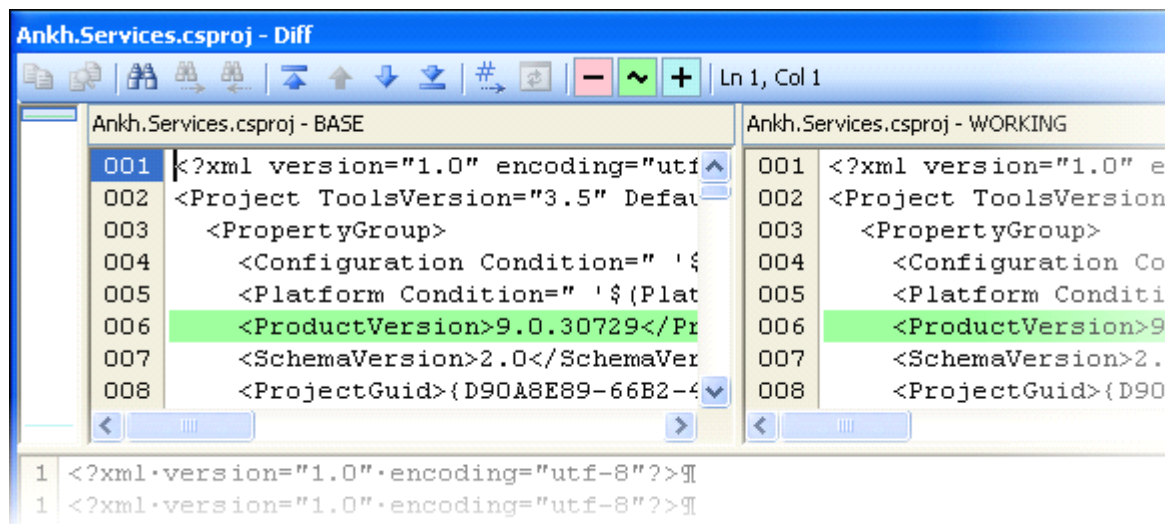
比较解决方案中某一个项在不同版本之间的区别，使用 Compare 操作。

1. 在解决方案浏览器中右击相应的项选择 Subversion > Compare。
 - a. 在 Select item for Diff 窗口，选择你需要比较的版本。

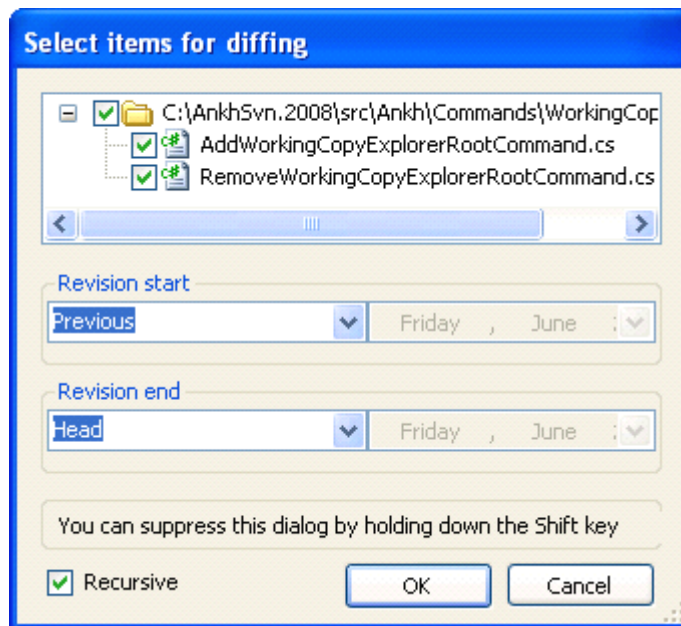


有多种不同的版本供比较，比如基线版本，工作拷贝版本，指定版本号，前一版本以及特定日期存在的版本。

- b. 点击 OK，下面是例子。



2. 如果需要查看你项目或解决方案中所有项的不同可以使用"Unified Diff"格式，在解决方案浏览器中右击，选择 Subversion > Unified Diff。
 - a. 在 Select item for Diff 窗口中选择你需要比较的版本。



有多种不同的版本供比较，比如基线版本，工作拷贝版本，指定版本号，前一版本以及特定日期存在的版本。

- b. 点击 OK，下面是例子：

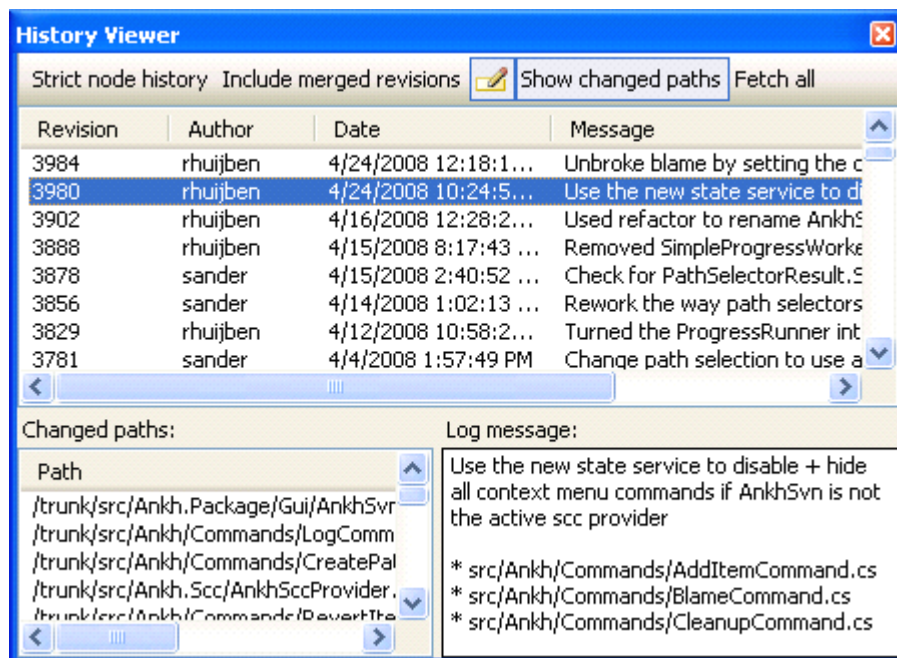


4.11 查看版本历史

使用 View History option 来查看一个项目文件或文件夹在版本库中的版本信息。

1. 在解决方案浏览器中右击一个项目文件或者文件夹，选择 **View History**。历史查看器从版本库中提取相关的日志信息并显示出来。上面的县市区显示的是版本的编号、布的时间和日期、作者以及提交的第一行日志。如果一个版本是从另一个版本拷贝的（例如分支、合并）他将以蓝色自己显示。
2. 查看特定的日志信息。
 - 修改路径去显示了关联到提交的所有文件和文件夹。文件或文件夹你需要操作这线功能的，修改将显示为蓝色字体，删除为红色，增加为红色。
 - 日志显示区显示提交时的完整日志。
3. Click Strict node history to see the equivalent of `svn log with the --stop-on-copy` option. In the History Viewer, history is normally reported up to the revision where the copy is created. For example, suppose File A is copied from trunk (rev5) to branch (rev6). File A is edited and is now at rev10. History normally shows all the revisions including the revisions in the trunk. With Strict node history, history only shows up to rev6.

Here's an example.



4.12 创建和应用补丁（patches）

为了控制大量开发者对于项目的贡献，代码通常是以补丁的形式提交审查，然后提交到版本库中的。

4.12.1 创建一个补丁

当你在本地已经开发并测试了你的代码，并且准备提交供审查时创建一个补丁。

1. 在解决方案浏览器中右击一个项，选择 **Subversion > Create Patch**。
2. 在 **Create Patch** 窗口中选择你需要包括在补丁中的文件，点击 **OK**。

3. 保存补丁文件，如果补丁文件需要进行扩展，约定适用.patch 或.diff 扩展。

补丁文件采用 Unified Diff 格式创建。它包含了自版本库更新之后的所有修改。

4.12.2 应用补丁

在你工作拷贝对应于创建补丁的相同目录层次应用补丁文件。



重要提示：在 AnkhSVN 中，只有你配置了 TortoiseSVN 作为外部合并和补丁工具时，你才可以应用补丁。

1. 在解决方案浏览器中右击相应项，选择 Subversion > Apply Patch。
2. 打开补丁文件。TortoiseMerge 将运行来将你的补丁文件合并到你的工作拷贝中。如果你想获得更多帮助，请查看 TortoiseSvn 的帮助：[Applying a patch file](#). （附件 3）

一旦补丁应用之后，你需要提交以便允许其他用户访问这些修改。

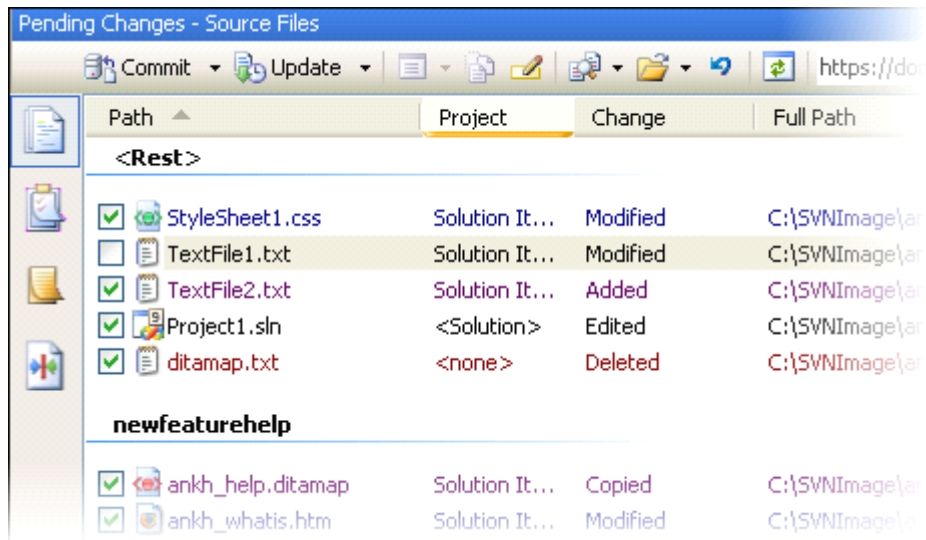
4.13 将项增加到更改列表中

如果你同时在解决好几个不同的问题，AnkhSvn 允许你将他们分组未逻辑的更改列表（Changelist）以便于更容易的跟踪。

对于 Changelist 的更多介绍，查看：[version Control with Subversion - Changelists](#). （附件 4）
在解决方案浏览器中右击相应已经修改的项。选择 Subversion > Move to Change List 并且选择以下选项之一。

- 点击<Change List>来创建一个更改列表并将相应项添加进去。
- 选择一个已经存在的更改列表。
- 选择 special ignore-on-commit changelist。这个列表中的文件将不会再提交窗口中被选择。

以下是将项放入更改列表中的例子。

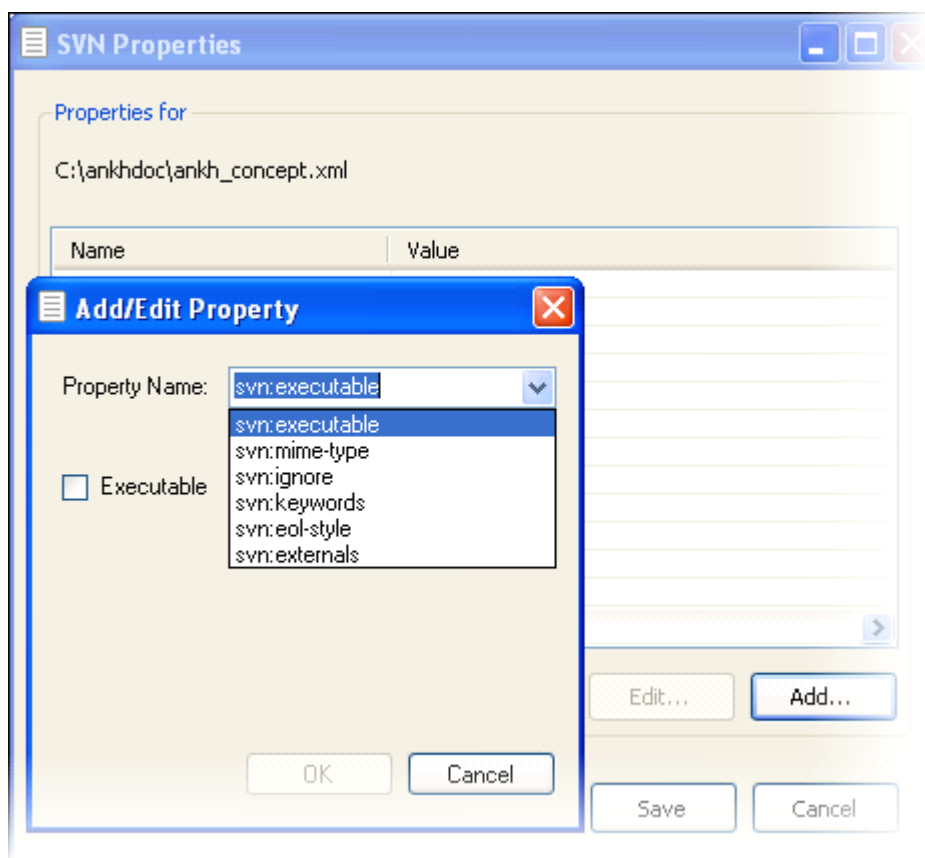


4.14 增加一个 Subversion 属性

你可以为项目文件或文件夹指定特定的 Subversion 属性。这些属性包含了 .svn 前缀。

例如,使用 svn:ignore 属性使得 Subversion 操作将忽略一个文件列表。对于 Subversion 属性的使用请参看: [Subversion Properties.](#)

1. 在解决方案浏览器中右击文件或文件夹, 选择 Subversion > Subversion Properties。
2. 在 SVN Properties 窗口中点击 Add。



3. 在 Add/Edit Property 窗口选择属性名称下来窗口中选择一个属性。
4. 基于你选择的属性，提供附加信息。例如，如果你选择了 svn:ignore 属性，指定那些行或模式 Subversion 操作将忽略。
5. 点击 OK。
6. 在 SVN Properties 窗口点击 Save。

属性将附加在一个文件上。更新或移除属性在 SVN Properties 窗口点击 Edit 或 Delete 按钮。

 提示：你增加属性之后，你必须提交你的变更。

4.15 清理工作拷贝

当你由于 Subversion 操作不完全或者提交不全而遭遇问题时，使用 Cleanup 选项清理你的工作拷贝使他可用。

在解决方案浏览器中右击相关项，选择 Subversion > Cleanup。

 建议：如果你的错误是"working copy locked"，在你工作拷贝的顶层目录执行此操作。

4.16 集成 issue tracker

当你为你的项目或解决方案安装了 issue tracker，你可以在 Pending Changes 的 Issues 视图查看和更新问题。



提示：AnkhSvn 允许并完全支持任何 issue tracker 插入，例如 CollabNet's issue tracking systems 就是为了这个功能而开发的。

需要集成 CollabNet issue tracker 你需要以下支持：

- CollabNet Desktop - Visual Studio Edition version 1.5.9 or later
(<http://desktop-vs.open.collab.net/>)
- AnkhSVN version 2.1.7250 or later

4.16.1 连接到 issue tracker

为你的解决方案配置 issue tracker，未 CollabNet tracking tool 提供细节。

1. 在解决方案浏览器右击你的解决方案，选择 Issue Tracker Setup。
2. 在 Issue Tracker Configuration 窗口，选择一个 issue tracker：
 - CollabNet TeamForge tracker
 - CollabNet Enterprise Edition Project Tracker
 - CollabNet Enterprise Edition Issue Tracker
3. 选择一个 CollabNet 网站和工程。



提示：项目强制要求连接一个 CollabNet Enterprise Edition 站点。对于一个 TeamForge 站点，你可以在站点或工程级别选择版本库。

4. 点击 OK。你的修改将在解决方案根目录被保存为一个自定义的 Subversion 属性。

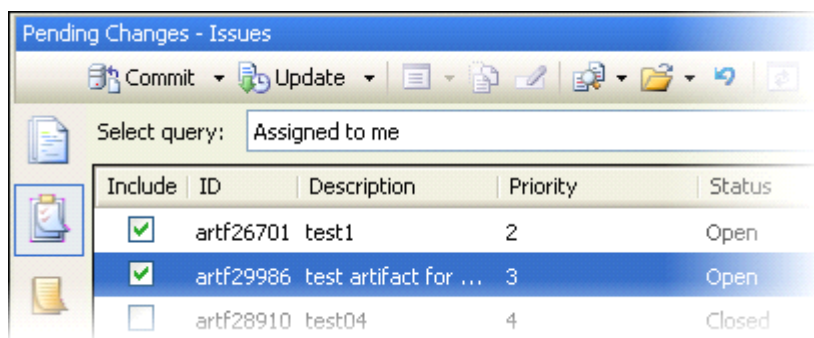


提示：如果其他开发者需要把这个配置作为他们解决方案的一部分，你需要将你的更改提交到你的 Subversion 版本库中。

4.16.2 将问题关联到一个 Commit

如果你已经为你的解决方案配置了一个 issue tracker，你可以在 Pending Changes 窗口中选择问题并关联到相应的 Commit。

1. 在 AnkhSVN 的 Pending Changes 窗口 Issues 视图，选择一个或多个问题。

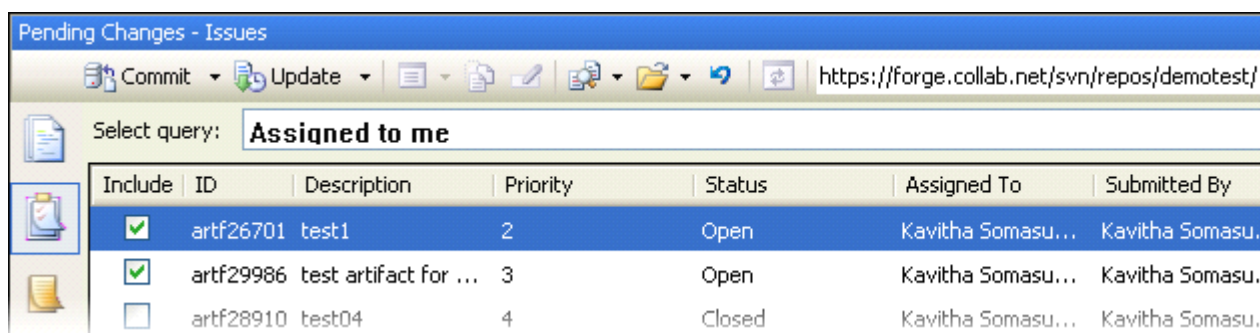


2. 在消息去输入你提交的日志信息。
3. 点击 Commit.

4.16.3 当提交提交时更新问题的状态

当将问题关联到提交时，你也可更新问题的状态。

1. 在 AnkhSVN 的 Pending Changes 窗口 Issues 视图，选择一个或多个问题。



2. 从 New Status 下拉列表中选择你希望提交之后的问题状态。
3. 输入日志并点击 Commit。

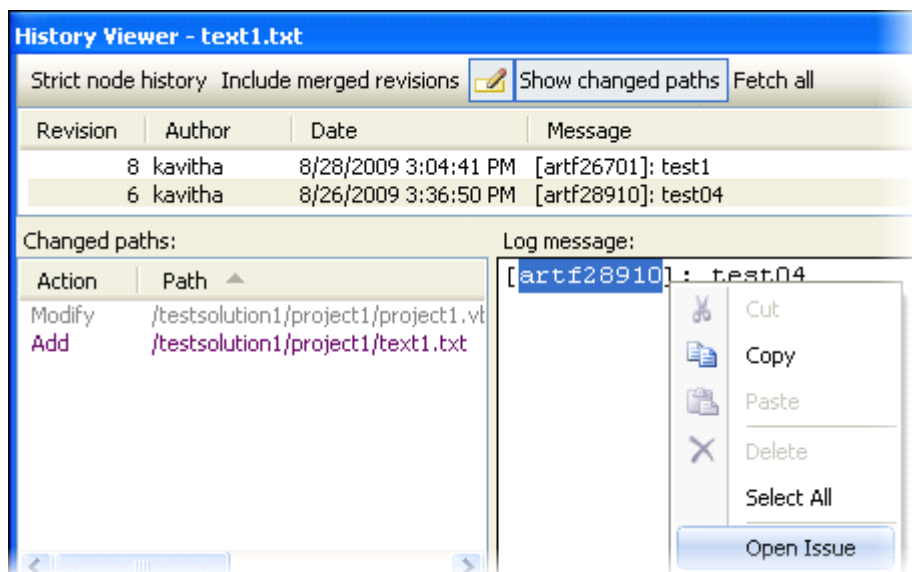
当你提交成功后，问题状态将被更新。

4.16.4 在查看历史时打开一个问题

当你查看一个关联问题的历史项时，你可以在日志区打开它。

1. 在解决方案浏览器中右击项，选择 View History。History Viewer 消息显示了作品的 ID 或者关联到该项的问题。
2. 在 History Viewer 中选择消息。

下面是一个例子：



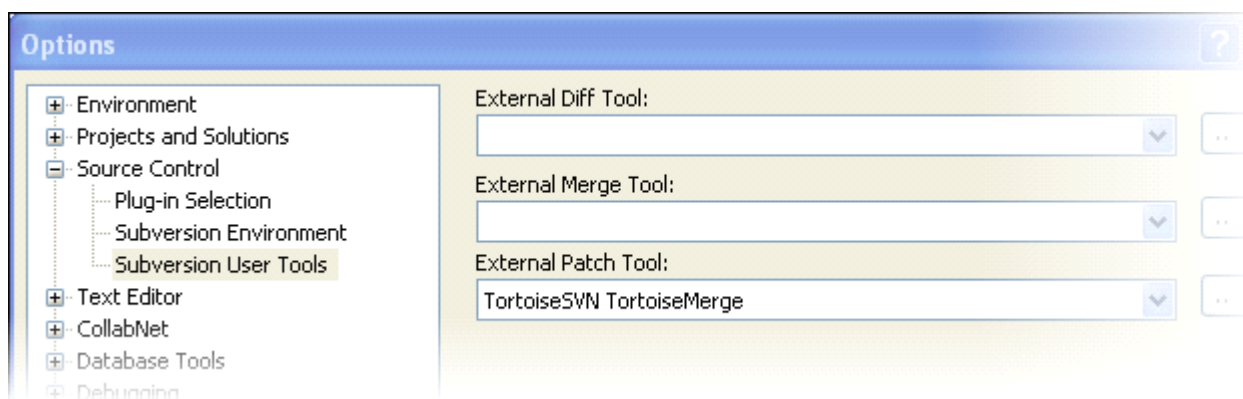
3. 在 Log message 区右击作品或者问题 ID，选择 Open Issue。

问题将会使用桌面编辑器（Desktop editor）打开。

4.17 配置和外部工具

AnkhSVN 允许你使用外部工具来进行 Merge、文件比较以及补丁操作。

1. 在 VS 中选择工具>选项。
2. 选择 Source Control > Subversion User Tools .

















3. 在下拉窗口中选择外部比较和合并工具。



提示：你可以选择一些列合并和比较工具，但是 TortoiseSVN 是你唯一可以选择的补丁工具。如果要能够应用补丁，你必须选择 TortoiseSVN。

4. 如果需要编辑命令，点击下拉窗口毗邻的按钮并且在命令编辑器中写入你的命令。
5. 在 Options 窗口点击 OK。

附件 1、AnkhSVN 图标样列表

	Must-lock
	Modified
	Deleted
	Editted
	Unmodified
	Missing
	Copied/Moved
	Locked + unmodified
	Locked + modified
	Ignored
	Added
	New
	Conflicted
	Unmodified+chg.below

附件 2、 Overview of CollabNet Merge Client

The CollabNet Merge Client has been built on top of Eclipse and Subclipse. This combination allowed us to support multiple client operating systems and also allowed us to focus on the actual merge client and merge process while leveraging the excellent Subversion capabilities that already exist in Subclipse. This gives you a powerful client that allows you to perform all Subversion operations from a single tool. Commit, History, Blame, Switch, Tagging. Everything you need is available. You do not have to be an Eclipse shop or even an Eclipse developer to use the client. Eclipse and Subclipse make it easy to checkout and access any project in your repository. You can just use the client as a general Subversion UI, or even just to perform merges. Adopting the client does not require that you adopt Eclipse as your only development tool.

The CollabNet Merge Client is also built on top of the merge tracking feature of Subversion 1.5 and requires Subversion 1.5 on the client.

The goal of the client is simple -- make merge easier. There were a couple of aspects to making merge easier. First, the Subversion 1.5 merge tracking features make certain kinds of merges much simpler as the user does not need to supply as much information to Subversion in order to perform a successful merge. So part of building this client was figuring out how to leverage this in the UI while still making it possible for a user to perform the more complicated merge scenarios. The second aspect to the development of the client was to just focus a lot of attention on the merge process itself and to provide the user with as much or as little assistance as needed to complete the merge process, which includes resolving conflicts and ultimately committing the changes to the repository.

For the purpose of this document, we will just walk through the merge process beginning to end and discuss the various features as they come up along the way. It might require reading through this a few times to catch all of the features that have been provided.

Performing a Merge

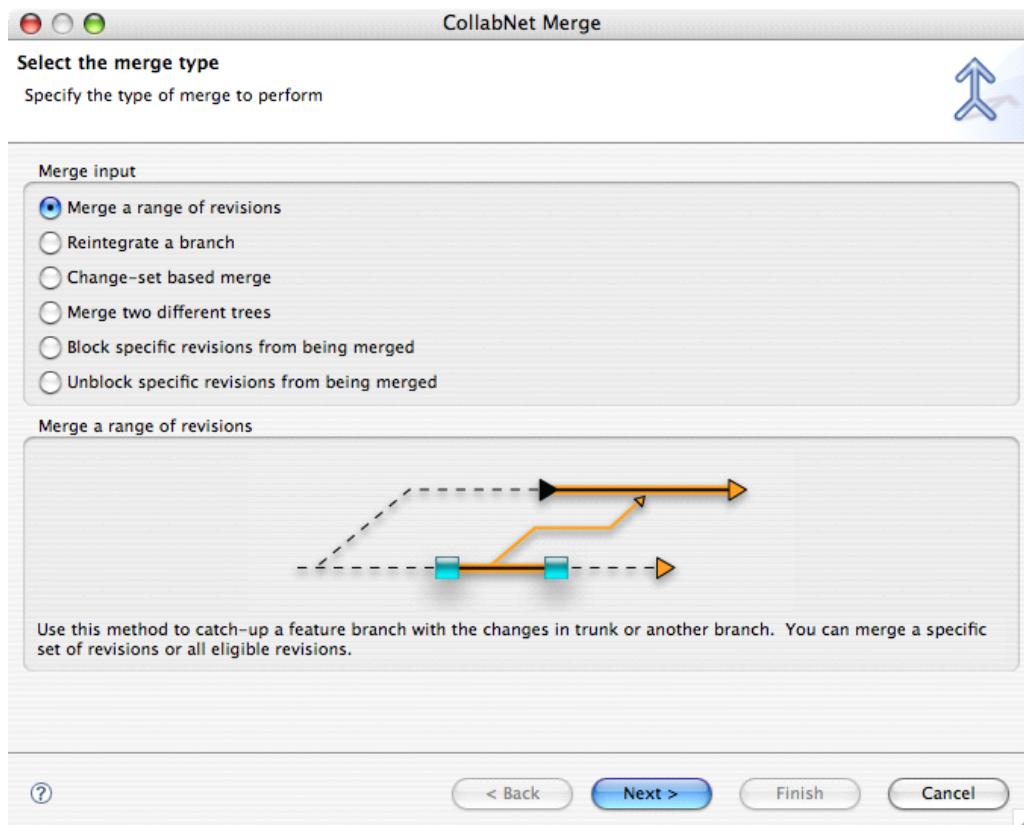
In order to perform a merge you must have a working copy that is pointing to the area of the repository that you want the merge results to be committed to. So to start the process you must first checkout a new project or update/switch an existing project in your Eclipse workspace. It is a best practice to not have any uncommitted changes in this working copy, and it usually a good idea to update the entire working copy to HEAD before beginning the merge.

Right-click on the project and choose Team > Merge. This will launch the CollabNet Merge wizard. We decided to adopt a wizard-based solution to the merge process for a couple of reasons. Most users are familiar with the wizard UI concept and it affords us the opportunity of sending the user down different paths in the UI based on the type of merge they are doing. This allows us to show or hide relevant aspects of the merge process based on their selections.

Note: The merge client is built on top of Subclipse, which has its own merge implementation. You can set the merge input provider in the Subclipse preferences. By default, the CollabNet Merge wizard, when installed, will be the default provider. If you are not seeing the wizard then check the preferences under Team > SVN > Diff/Merge.

Select the Merge Type

The first page of the wizard asks you to specify the type of merge you want to perform. The merge client was designed to be extensible via Eclipse's plug-in mechanism. The screenshot shows the inputs that are available by default:



The terminology used to describe the merge inputs is based on the type of merge operation you want to perform. Most merge scenarios would be covered by either the first or second option in the list. The first option is designed to cover the scenario where you want to merge changes that have happened in the "branch source" to the branch. This covers a lot of scenarios, such as synchronizing a feature branch with the changes that have happened in trunk, as well as selectively back-porting fixes to a release branch.

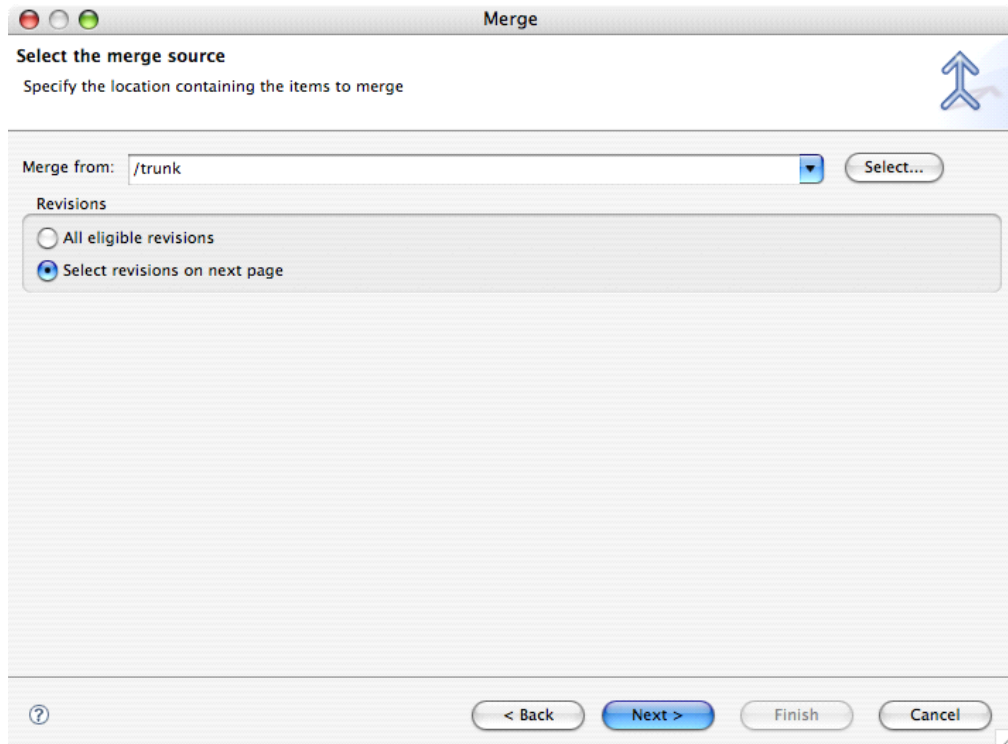
The second merge input, reintegrate, is for merging the changes that have happened on a branch back to the "branch source" or trunk. This would typically cover the scenario where you have completed the work on a feature branch and want to reintegrate the changes back to trunk.

The change set-based merge input is covered in its [own document here](#).

The last two merge inputs cover the scenarios of manually recording or removing merge information which can also be used to block or unblock revisions.

Select the first merge input and click Next.

Select the Merge Source



On this page of the wizard you are indicating what you want to merge. There are a couple aspects of this page that are leveraging the merge tracking feature:

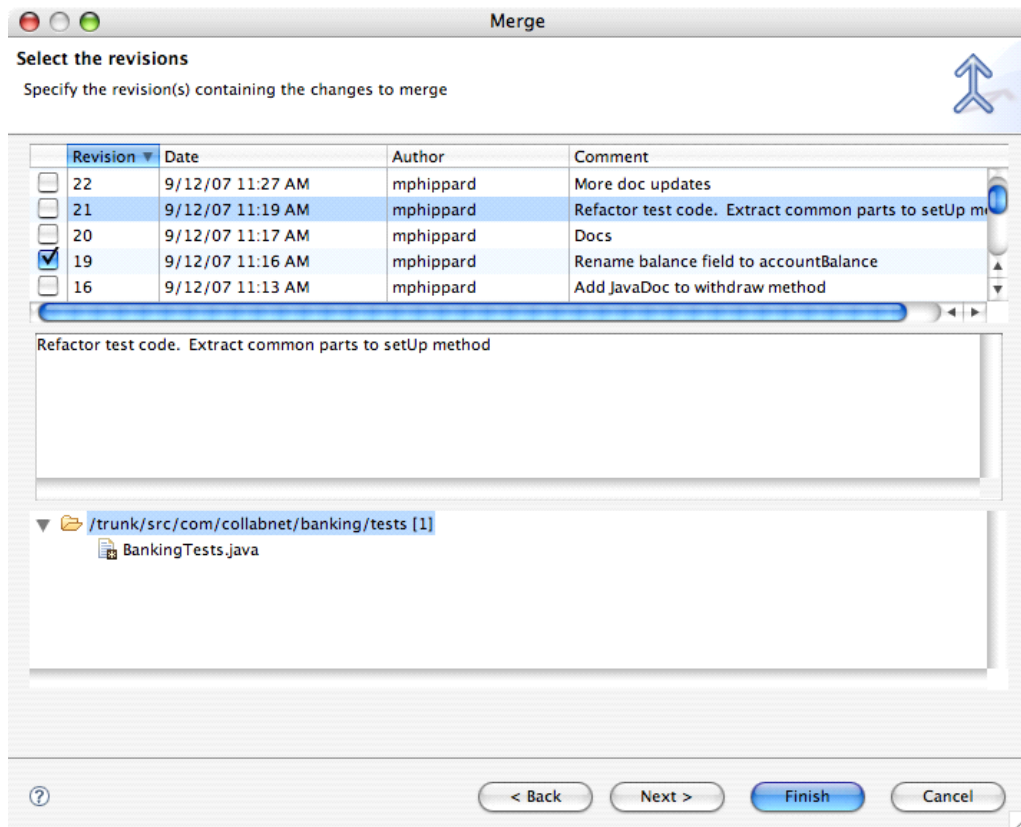
1. The "merge from" drop-down will contain a list of suggested merge sources. This is populated from the contents of the `svn:mergeinfo` property. The value will default to the last location you merged into the target location, or the location it was originally copied from if no merges have happened yet.
2. In the revisions section, there is an "All eligible revisions" option. This is equivalent to running the `svn merge` command with no revision arguments specified. When this option is used, Subversion will merge all revisions in the from location that have not already been merged into the target.

Both of the above features require a Subversion 1.5 server to work properly. For example, if the server is running 1.4 then only the location the target was copied from will appear in the combo box (or just the location itself if it was never copied). Likewise, the All eligible revisions feature will not work as expected if the server is not running 1.5. In that scenario you can still use the option to select specific revisions, which is similar to what you would do today.

The "All eligible revisions" feature would be useful if you are doing something like merging all changes that occurred on trunk since a branch was created or last synchronized with trunk. If you selected the "All eligible revisions" option you would essentially be done and could click Finish or Next to go to the last page of the wizard. For the sake of explaining all of the possible wizard pages, I will assume you chose to select specific revisions.

Before moving on, I just want to point out a small feature we added to make the process a little easier. Note in the above screenshot that the UI does not require you to enter URL's. You can enter paths that are relative to the repository root, and the client will convert those into URL's for you. Of course you can still enter a full URL, but this was just a simple feature we added to make the client a little easier to use.

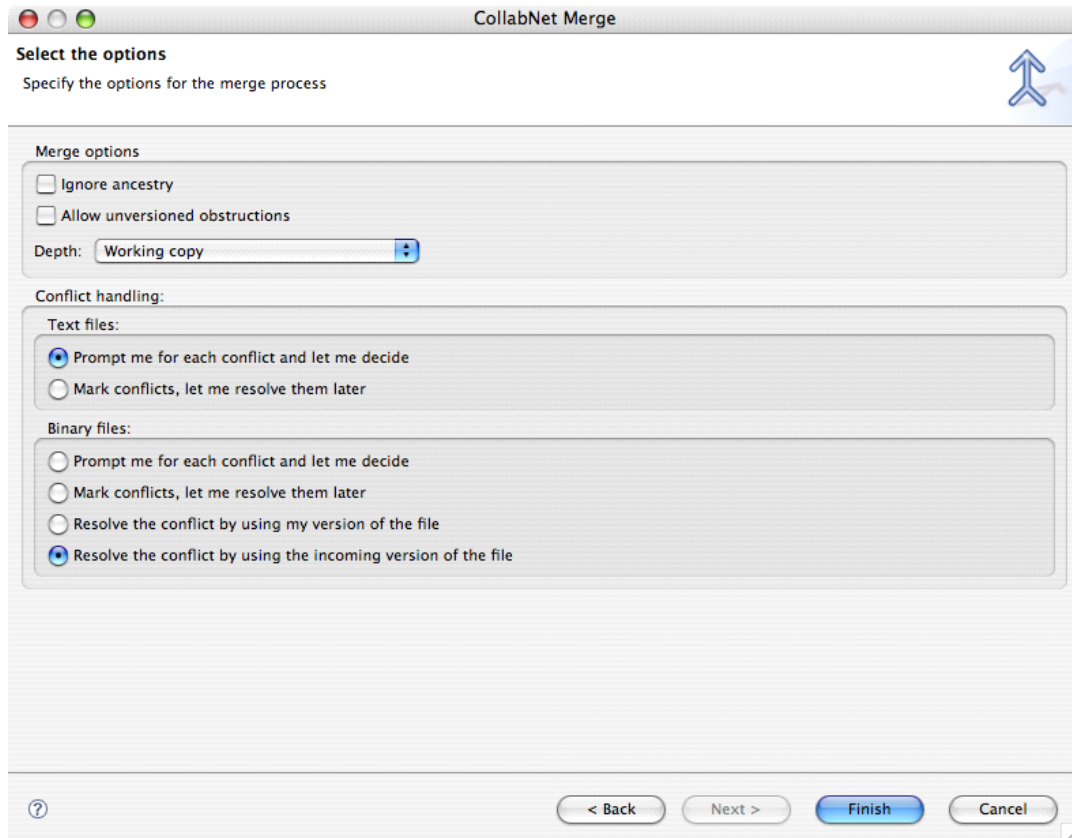
Select the Revisions



This page will list all of the revisions that are eligible to be merged. When talking to a 1.5+ server, it will filter out any revisions that have already been merged into the target. A nice feature of this page is that it should be clear to the user that they are selecting the specific revisions they want to merge. They do not need to be confused with the underlying syntax it requires to merge those revisions. In addition, the wizard allows you to select non-contiguous revision ranges. You do not have to merge an entire range of revisions, you can omit some revisions.

We will assume you selected some revisions and clicked Next to see the last page. However, note that the Finish button is also enabled. We do not want you to have to visit the last page of the wizard unless you want to.

Select Merge Options



This final page of the wizard is common to all merge inputs, so the wizard will always end with this page. As mentioned earlier, the Finish button is always enabled before you get to this page. If you do not visit the page, then default values will be used. The defaults are explained below.

This page has two sets of options on it. The first part contains some standard options that are available on the merge command (`--ignore-ancestry`, `--force` and `--depth`). These always default back to what we believe are the most common and sane defaults as shown above. The second part contains options for how you want to handle conflicts, which you are more likely to want to change, so we remember the last values you used for these options.

The conflict handling options will be explained in more detail later. Hopefully, they are fairly self-explanatory. Essentially, you can now ask to be prompted during the merge process to resolve conflicts as they occur, or use the default behavior of creating conflicts in the workspace and resolving them after the merge completes. A nice feature is that we allow you to specify different options for binary files. So maybe you are merging a change to your web design into trunk and you know if there are any graphics conflicts you just want to take the version you are merging in. We let you do that with an option up front so that you do not even have to be asked what to do.

That is it. Click Finish to perform the merge. The rest of this document will cover the features we have added to help you while the merge is running and after it completes.

The Merge Results View

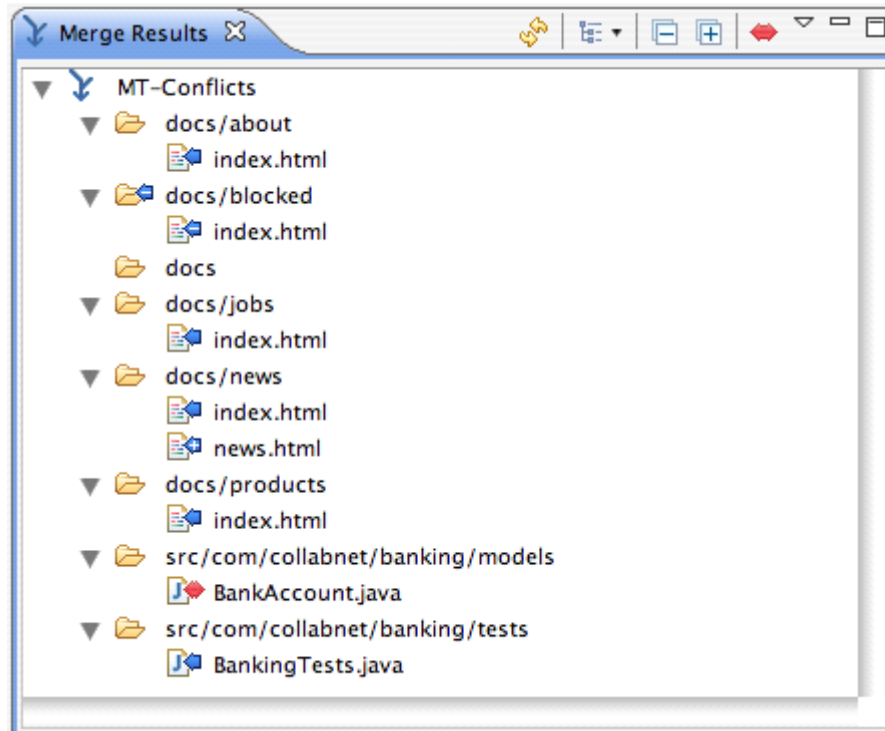
Once the merge begins, it does not matter which wizard option you used, they all use the same merge process the rest of the way. For now, let's set aside the option of being prompted to resolve conflicts. We can come back to that later.

When we designed this client, one of the main things we wanted to do was make it real easy to understand what the merge process did, and let you work with those results. To meet those objectives, we created a special view for working with the merge results -- the Merge Results view. If you have used Eclipse you will notice this view looks and behaves a lot like the Synchronize view. That was on purpose as we modeled our view on the Synchronize view. We decided to create our own view rather than use the Synchronize view for a few of reasons:

1. **Simplicity.** The Synchronize view API's are fairly complex and we did not want to spend forever debugging problems where we did not understand the API or it did not work quite

the way we wanted it to.

2. Features. There are some features we implemented in this view, and possibly more we have not yet, that we did not think we could implement in the Synchronize view. We did not want to get the Synchronize view working well only to later realize we would have to scrap it.
3. Properties. This view makes an attempt to show changes to Subversion properties. We think this would have been difficult to do in the Synchronize view.



Anyway, we decided to create our own view and model it as closely as possible on the Synchronize view and its features and usability. In the screenshot you can see that it is very easy to see the changes that were made by the merge, including adds, deletes and conflicts. The view offers two presentation modes, flat and collapsed folders. The screenshot shows the collapsed folders presentation after using the Expand All option. The view also has a toolbar button to toggle between showing just conflicts or all changes. After performing a really large merge, this toggle makes it very easy to find and focus on the conflicts.

Working with the View

One feature we decided was very important was that the contents of the view had to be persistent. In other words, you might do a really large merge with tens or hundreds of conflicts that will take several days to resolve. It was important that the contents of the view still be there when you close and restart Eclipse each day. Consequently, you have to manage the contents of this view. When you are done with a merge and ready to commit the changes, you should take the option to remove the result from the view. The view can contain the results from many merges at the same time. Here are a few more features of this view (in no particular order):

- All of the normal Eclipse and Subclipse editing options are available.
- If you double-click on a file we launch the compare editor or the edit conflicts option depending on the current state of the file.
- We extended the Subclipse edit conflicts feature to support custom conflict resolution programs by file type. So you could use the built-in Eclipse editor for some file types, WinMerge or some other external tool for other file types, and even a custom script to resolve conflicts in something like a Microsoft Word document.

- One of the features we wanted to implement in this view that we did not think the Synchronize view could handle well, was "skipped" files. The Subversion merge process will often skip a file for various reasons. Usually you did something like delete the file in a branch and when merging changes from trunk, any changes to that file are skipped. When running the Subversion command line, this is just a message that goes to the console. If you do not see it, you do not know it happened. The merge results view shows anything that was skipped using a grayed out font. We then provide two special options on these types of items (remember they do not exist locally).
 1. You can examine the history of the item in the merge source. This lets you see more information about the item which might lead you to decide that you want it back in your branch.
 2. A copy option was added that does an svn copy URL WC command to create the item in your working copy by copying it from the merge source. This allows you to put the item back in your branch if you want it.
- Finally, this view contains a special "Mark Resolved" option you can use after resolving conflicts. This will show the item with a special resolved check-mark decorator in the view. The item will still show in the filtered list of conflicts. Normally, once you resolve a conflict in Subversion the file now just shows as a file with local modifications. There is no way to know that it originally had conflicts. So this feature lets you track your progress in resolving conflicts as well as remember all of the files that had conflicts in the first place.

Merge Previews

Another feature, really more of a concept, we felt was important to implement was a merge preview. The design of Subversion and its merge feature does not easily accommodate a feature like this. We could have used the merge --dry-run option to populate the merge results view, but there is no way to reliably show you the details of what would have been merged. In other words, there is no way to show a 100% accurate graphical compare if the user wants to see one. Also, from a performance standpoint, a --dry-run merge does everything that a full merge does except write the final file to disk. So there is no performance advantage to using --dry-run.

The Subversion developers we talked to about this felt that the best way to see what a merge was going to do was to just run the merge. This lets you see exactly what happens as well as gives you plenty of tools for examining the results. If you really only wanted to preview the results or you decided that you do not want to do the merge, you can just run revert and discard the results. That is the approach we decided to take, but with some features to make it easier. If you want to preview a merge, you just do the same thing you would do if you were going to do a real merge. When the merge completes you can work with the results as you normally would. When you are done, we have added a special "Undo merge" option on the merge result node in the view. This option launches a wizard which recursively reverts all changes in the working copy, and then follows that up with a secondary process that lets you remove any files that are unversioned. This is because if the merge process added files, the Subversion revert option will not delete them. So the undo merge option cleans up the working copy. In addition, since we know that we want to revert the entire working copy, we just do a recursive revert which runs considerably faster than the normal revert option used in Subclipse.

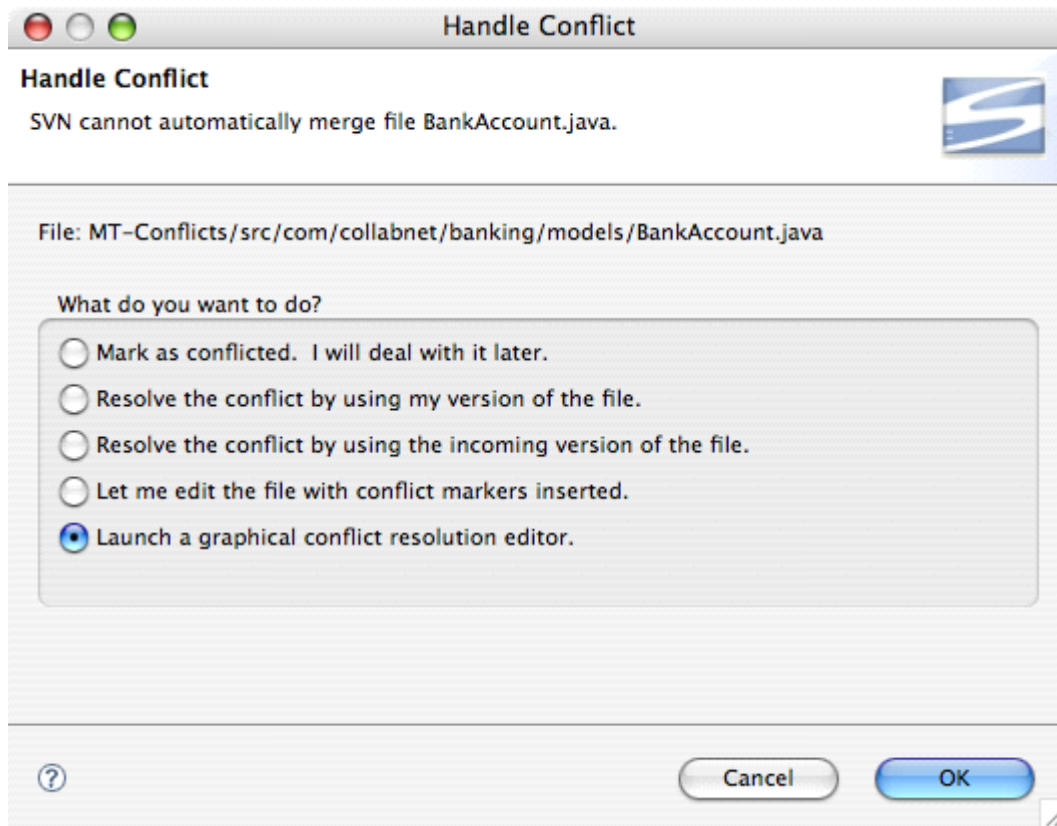
In the end, we were very happy with how this feature worked out and think it does the job well. It is very important that you do not have any local uncommitted changes in your working copy before doing this. If you do, the Undo merge feature will remove them when it cleans up your working copy.

Resolving Conflicts

One of the most difficult parts of any merge process is manually resolving the conflicts that cannot be automatically merged by the tool. The Subversion merge process does a good job automatically merging most changes, but there are always going to be certain changes it cannot automatically merge. There are a number of things we have done in the merge client to make this process easier for you. Let me first go back and cover two features that were already discussed.

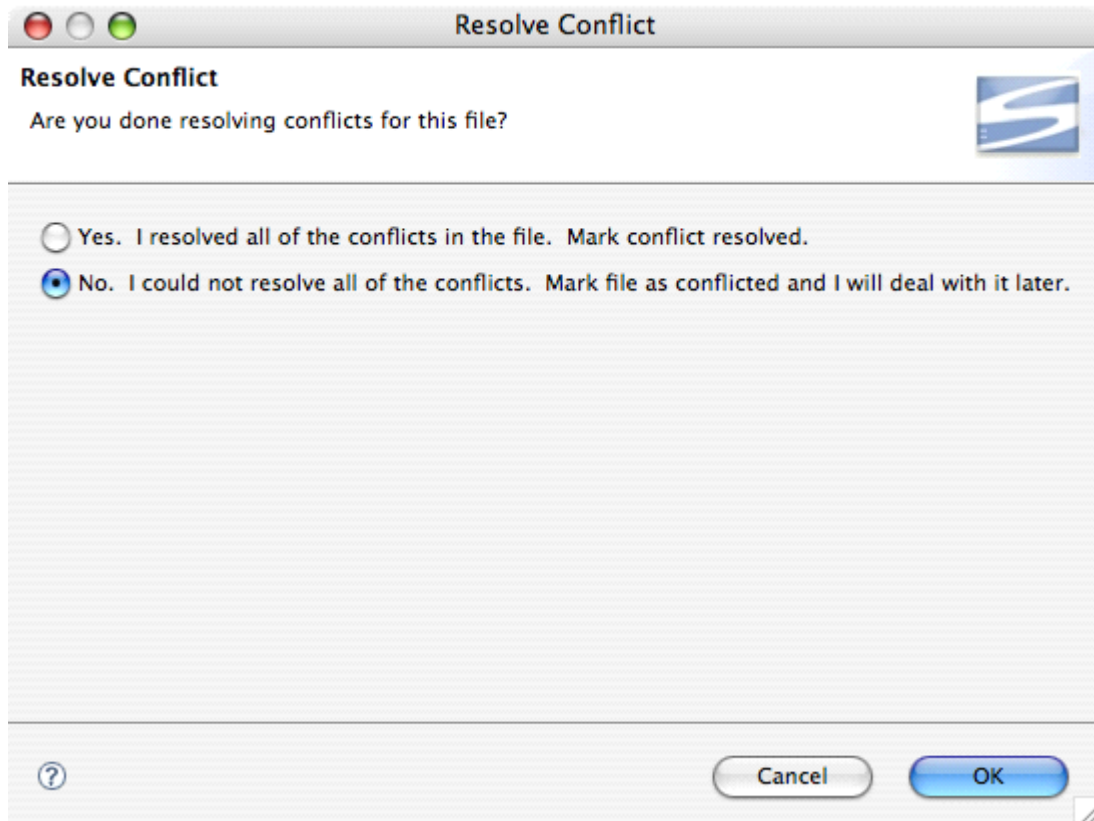
1. On the final page of the merge wizard you are given some options on how to handle conflicts. The default out of the box is to just let Subversion record the conflicts in the working copy so you can resolve them later. If you already use Subversion, then this is how it works today with Subversion 1.4. We also allow you to be prompted during the merge process (which is what the rest of this section will cover). Finally, for binary files only, we allow you to tell us before the merge starts that you want to take a specific version of the file. Most binary files cannot be merged anyway, which means you resolve most conflicts by taking a specific version of the file. This lets you just do that up-front so that you do not even have to be asked during the process.
2. The merge results view has a toggle that allows you to focus on the folders and files that have conflicts and then it also has a visual indicator that shows you which conflicts you have resolved. Essentially, it allows you to more easily manage the process of resolving the conflicts.

As mentioned, the other option that we provide is to allow you to be prompted to resolve conflicts during the merge process as the conflicts are encountered. When this feature is enabled and you run into a conflict, you receive a dialog like this:



As you can see, there are several options available. You can just let Subversion mark it as a conflict and deal with it later. You can also choose to resolve the conflict by just selecting a specific version of the file. This feature is most useful for binary files, but it can sometimes be useful for text files too. Finally, there are also two options for resolving the conflicts manually. One is to open the file in an editor with conflict markers inserted. The other option is to use a graphical conflict resolution tool. Eclipse includes such a tool and that is used by default, but you can also configure the client to use an external tool such as TortoiseMerge, WinMerge etc. As mentioned earlier, you can even configure a different tool based on the file extension. When you take this option, we just open the conflict resolution tool that you have configured for this file type, or the default if you have not configured a tool.

The details of actually resolving the conflict fall upon you. When you are done, if you made changes you want to preserve you need to save the file using the editor and then close it. You are then presented with this dialog.



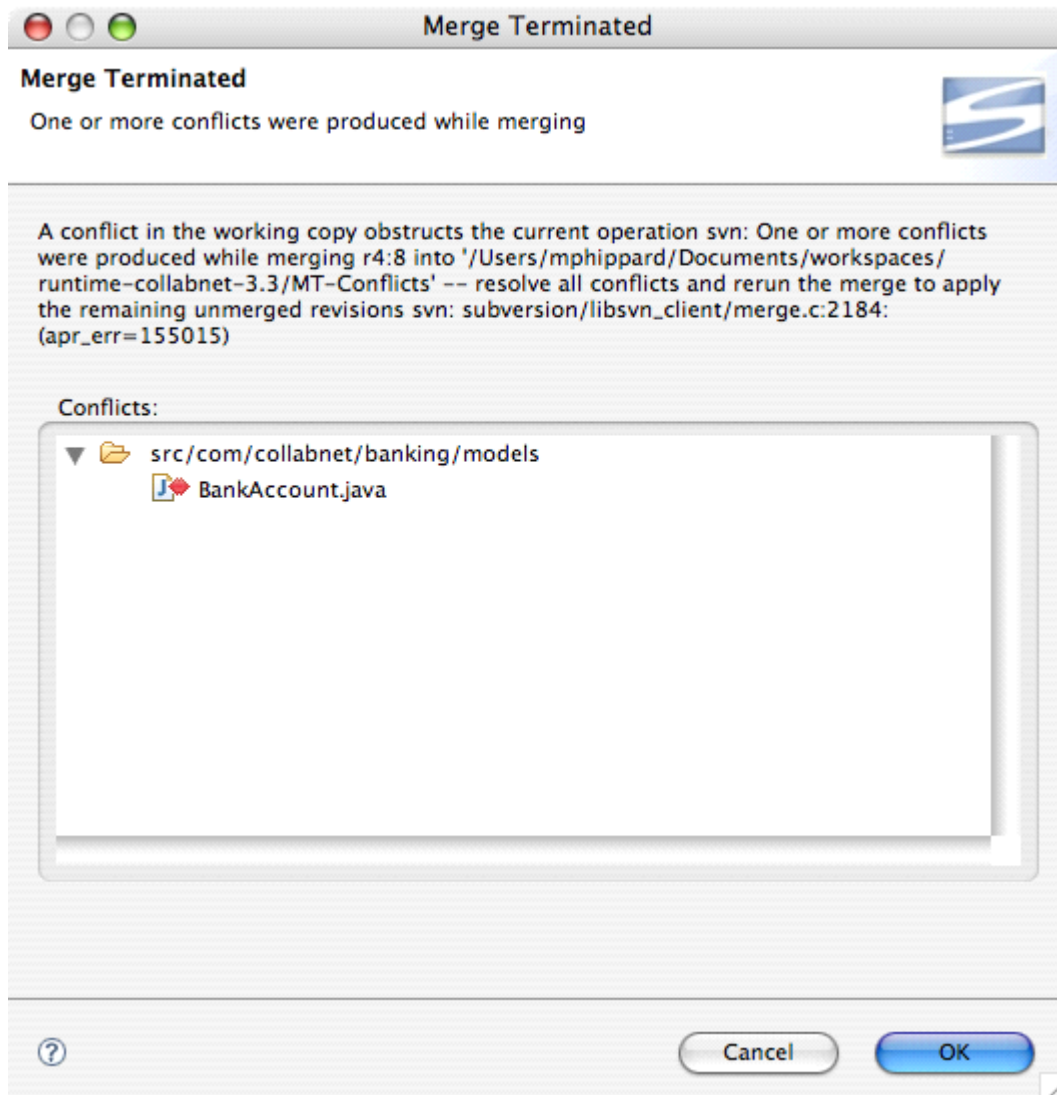
This dialog allows you to tell the client if you actually resolved the conflicts. You might have taken a look at the code, realized it was going to be more complicated than you expected, and just decided to resolve it later. In that case, just cancel the editor and give the appropriate answer in this dialog. The merge process will resume at the next file and possibly prompt you again as it encounters more conflicts along the way.

Resume Merge Process

The merge tracking feature in Subversion 1.5 introduces some new scenarios that you could not get with earlier versions. Suppose you tell Subversion to merge All eligible revisions. Let's assume you made a branch at r100, merged a specific change from trunk (r150) and trunk is now at r200. The "All eligible revisions" option will break this merge up into two parts. First it will merge r100:149 then it will merge r151:r200. I like to refer to these as "merge passes" but I do not believe it has an official term. Anyway, suppose the first merge pass creates some conflicts, and you choose not to resolve them as part of the merge process. When this happens, the Subversion merge process has to abort at the completion of that pass. The reason it needs to do this is that it is possible that the merge of the next pass will require merging into those same files that currently have conflicts. There would be no way to do this reliably, so instead the process aborts so you can resolve all of the conflicts and then resume the merge process where it left off.

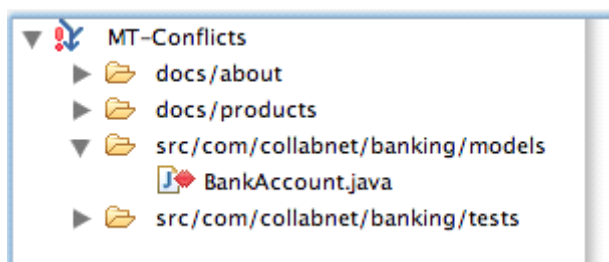
The CollabNet merge client manages all of this for you.

Let's start when the merge process aborts at the end of a pass. Remember, it will do this when conflicts are completed during a merge pass, and you did not use the interactive resolution feature to resolve the conflict during the merge. As an aside, let me also just point out that most merges will not involve multiple passes and will therefore not run into any of this. Back to the story, let's say the merge process aborts. You will get a dialog that looks like this:



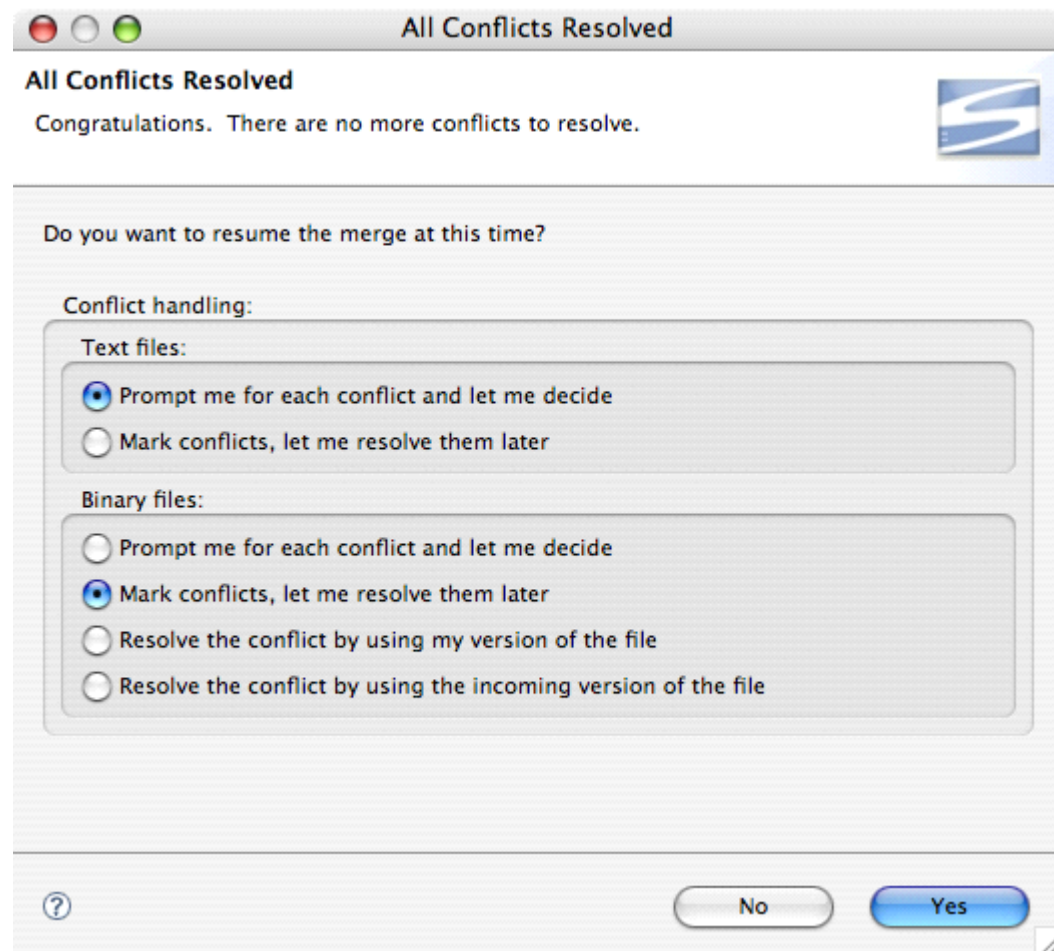
The dialog shows you all of the files that have conflicts as well as the message that Subversion produced. The message will tell you information about the pass it was working on when the merge aborted.

After you click OK, you need to resolve the conflicts. The merge results view gives you a special indication that shows you that this merge aborted early as you can see in the screenshot to the right.



You need to go through the conflict resolution process for each file and as you finish with each file use the Mark Resolved option on the right-click menu.

Eventually, you will use the Mark Resolved option on the final file that had conflicts remaining. When you do, you will get a dialog like this:



This dialog tells you that you completed resolving the conflicts and asks if you want to resume the merge process. Just click OK and it will be resumed where it left off. If you choose not to resume the process when the dialog comes up (maybe you want to go home for the day) you can right-click on the merge result and a Resume merge option will be enabled to start the process. This entire cycle could repeat several times if your merge involves many passes and there are unresolved conflicts after each. Eventually you will get to the end of the cycle and the merge will complete. At that point the icon in the merge result view will go back to normal and the cycle will end.

Summary

This document contained a lot of information. Print it out and read through it a few times. It might also help to look back at it again after having used the client for a while. The goal of the client was to make merge easier. This client is definitely a step in the right direction. It allows you to tap into the ease of use (and power) provided by the Subversion 1.5 merge tracking feature. At the same time, we took a look at the merge process in general and looked for ways to make it easier for you to manage and understand. Hopefully you will try this client out and find it helpful.

Finally, please read this document on [Additional features of the merge client](#) to learn about some other features we are providing which are not covered in the above document.

附件 3、Creating and Applying Patches

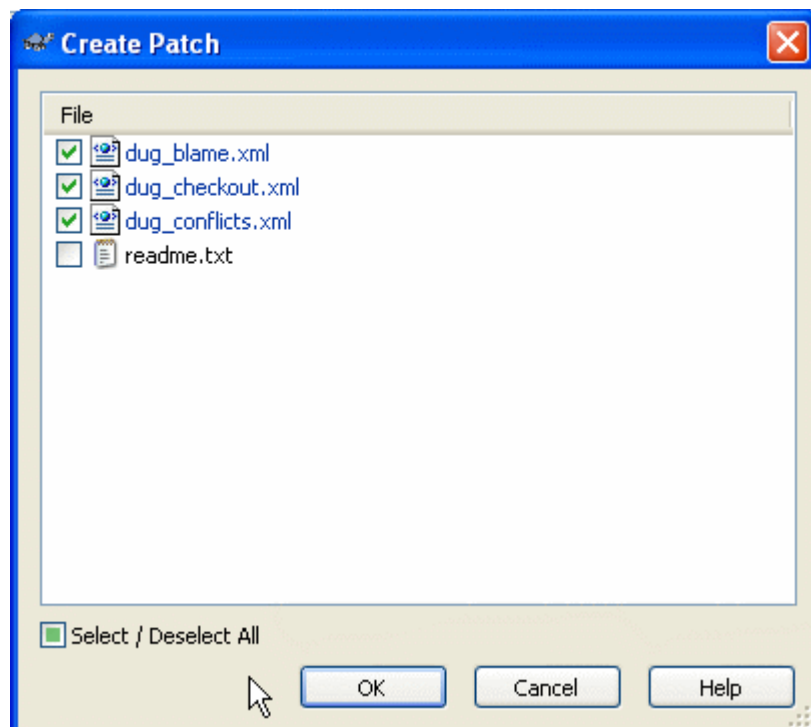
For open source projects (like this one) everyone has read access to the repository, and anyone can make a contribution to the project. So how are those contributions controlled? If just anyone could commit changes, the project would be permanently unstable and probably permanently broken. In this situation the change is managed by submitting a *patch* file to the development team, who do have write access. They can review the patch first, and then either submit it to the repository or reject it back to the author.

Patch files are simply Unified-Diff files showing the differences between your working copy and the base revision.

Creating a Patch File

First you need to make *and test* your changes. Then instead of using **TortoiseSVN** → **Commit...** on the parent folder, you select **TortoiseSVN** → **Create Patch...**

Figure 4.43. The Create Patch dialog



you can now select the files you want included in the patch, just as you would with a full commit. This will produce a single file containing a summary of all the

changes you have made to the selected files since the last update from the repository.

The columns in this dialog can be customized in the same way as the columns in the **Check for modifications** dialog. Read [the section called "Local and Remote Status"](#) for further details.

You can produce separate patches containing changes to different sets of files. Of course, if you create a patch file, make some more changes to the *same* files and then create another patch, the second patch file will include *both* sets of changes.

Just save the file using a filename of your choice. Patch files can have any extension you like, but by convention they should use the *.patch* or *.diff* extension. You are now ready to submit your patch file.

You can also save the patch to the clipboard instead of to a file. You might want to do this so that you can paste it into an email for review by others. Or if you have two working copies on one machine and you want to transfer changes from one to the other, a patch on the clipboard is a convenient way of doing this.

Applying a Patch File

Patch files are applied to your working copy. This should be done from the same folder level as was used to create the patch. If you are not sure what this is, just look at the first line of the patch file. For example, if the first file being worked on was *doc/source/english/chapter1.xml* and the first line in the patch file is *Index: english/chapter1.xml* then you need to apply the patch to the *doc/source/* folder. However, provided you are in the correct working copy, if you pick the wrong folder level, TortoiseSVN will notice and suggest the correct level.

In order to apply a patch file to your working copy, you need to have at least read access to the repository. The reason for this is that the merge program must reference the changes back to the revision against which they were made by the remote developer.

From the context menu for that folder, click on **TortoiseSVN** → **Apply Patch...** This will bring up a file open dialog allowing you to select the patch file to apply. By default only *.patch* or *.diff* files are shown, but you can opt for "All files". If you previously saved a patch to the clipboard, you can use **Open from clipboard...** in the file open dialog.

Alternatively, if the patch file has a *.patch* or *.diff* extension, you can right click on it directly and select **TortoiseSVN** → **Apply Patch...**. In this case you will be prompted to enter a working copy location.

These two methods just offer different ways of doing the same thing. With the first method you select the WC and browse to the patch file. With the second you select the patch file and browse to the WC.

Once you have selected the patch file and working copy location, TortoiseMerge runs to merge the changes from the patch file with your working copy. A small window lists the files which have been changed. Double click on each one in turn, review the changes and save the merged files.

The remote developer's patch has now been applied to your working copy, so you need to commit to allow everyone else to access the changes from the repository.

附件 4、Changelists

It is commonplace for a developer to find himself working at any given time on multiple different, distinct changes to a particular bit of source code. This isn't necessarily due to poor planning or some form of digital masochism. A software engineer often spots bugs in his peripheral vision while working on some nearby chunk of source code. Or perhaps he's halfway through some large change when he realizes the solution he's working on is best committed as several smaller logical units. Often, these logical units aren't nicely contained in some module, safely separated from other changes. The units might overlap, modifying different files in the same module, or even modifying different lines in the same file.

Developers can employ various work methodologies to keep these logical changes organized. Some use separate working copies of the same repository to hold each individual change in progress. Others might choose to create short-lived feature branches in the repository and use a single working copy that is constantly switched to point to one such branch or another. Still others use *diff* and *patch* tools to back up and restore uncommitted changes to and from patch files associated with each change. Each of these methods has its pros and cons, and to a large degree, the details of the changes being made heavily influence the methodology used to distinguish them.

Subversion 1.5 brings a new *changelists* feature that adds yet another method to the mix. Changelists are basically arbitrary labels (currently at most one per file) applied to working copy files for the express purpose of associating multiple files together. Users of many of Google's software offerings are familiar with this concept already. For example, [Gmail](#) doesn't provide the traditional folders-based email organization mechanism. In Gmail, you apply arbitrary labels to emails, and multiple emails can be said to be part of the same group if they happen to share a particular label. Viewing only a group of similarly labeled emails then becomes a simple user interface trick. Many other Web 2.0 sites have similar mechanisms—consider the “tags” used by sites such as [YouTube](#) and [Flickr](#), “categories” applied to blog posts, and so on. Folks understand today that organization of data is critical, but that how that data is organized needs to be a flexible concept. The old files-and-folders paradigm is too rigid for some applications.

Subversion's changelist support allows you to create changelists by applying labels to files you want to be associated with that changelist, remove those labels, and limit the scope of the files on which its subcommands operate to only those bearing a particular label. In this section, we'll look in detail at how to do these things.