|  |  |
| --- | --- |
| 199.jpg | Subversion & Trac |
|  | BitsChipsRC30Target_HANtune.jpg |
| Version 1.0  2/20/2013 | Manual |
|  | A manual for the installation and usage of Trac and Subversion. |

Copyright (c) 2020 [HAN University of Applied Sciences]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# Table of contents

[1 Table of contents 2](#_Toc349128308)

[2 Revision history 3](#_Toc349128309)

[3 Introduction 4](#_Toc349128310)

[3.1 Purpose and scope 4](#_Toc349128311)

[3.2 Problem definition 4](#_Toc349128312)

[3.3 Definitions 4](#_Toc349128313)

[3.4 References 4](#_Toc349128314)

[4 Installation 5](#_Toc349128315)

[4.1 Trac 7](#_Toc349128316)

[4.2 Subversion 9](#_Toc349128317)

[5 Appendix A: How to use Trac 12](#_Toc349128318)

[5.1 Create a ticket (task, problem report or change request) 12](#_Toc349128319)

[5.2 View tickets 12](#_Toc349128320)

[5.3 Modify a ticket 13](#_Toc349128321)

[6 Appendix B: How to use TortoiseSVN 15](#_Toc349128322)

[6.1 Checkout a working copy 15](#_Toc349128323)

[6.2 Adding files to the repository 17](#_Toc349128324)

[6.3 Commit your changes to the repository 19](#_Toc349128325)

[6.4 Update the working copy to a revision 21](#_Toc349128326)

[6.5 Revert local changes (working copy) 22](#_Toc349128327)

[6.6 Revert changes from a revision (repository) 24](#_Toc349128328)

[6.7 Switching your working copy 25](#_Toc349128329)

[6.8 Create a new branch in an existing project 27](#_Toc349128330)

# Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Revised by** | **Date** | **Description** |
| 0.9 | Jeroen Scheerder | 2/19/13 | Changed the layout of the document to match the standard layout. |
| 1.0 | Jeroen Scheerder | 2/20/13 | Expanded the manual with chapter 6.8 Create a new branch in an existing project. |

# Introduction

*What is Trac and why should you want to use it?*

Trac is a lightweight project management tool that is implemented as a web-based application, written in the Python programming language. It emphasizes ease of use and low ceremony, and is open source.

*What is Subversion and why should you want to use it?*

Subversion (often abbreviated as SVN) is a versioning system which lets you track and manage changes you make to files. It allows you to revert the changes you have made and helps you merge changes that you and others made to the same file. The repository can also be seen as a back-up facility with the additional benefit of easily sharing that backup with others.

Each Subversion project on the repository hosting server consists of the Subversion repository itself and a Trac project tracker to track implementation tasks, problem reports (PRs) and change requests (CRs).

## Purpose and scope

This document is written as a reference document for using Trac and Subversion. Trac is a web-based project management and bug-tracking tool and Subversion is a version control system.

It describes the following:

* Howto use Trac
* Howto use Subversion

## Problem definition

Currently DTI doesn’t have a clear process for reporting bugs and keeping track of software versions. This has proven a basis for errors. On several occasions bugfixes were delayed due to lost bugreports. Also previous (released) versions could not be reproduced. Although these exact processes have yet to be defined, it is clear that Trac and Subversion will be beneficial to a clear and simple process execution.

## Definitions

The following definitions are used in this document:

|  |  |
| --- | --- |
|  | **Definition** |
| ECU | Engine Control Unit |
| UML | Unified Modeling Language |
| TCU | Transmission Control Unit |

## References

“Techtrack Tutorial Subversion & Trac”, Tjeerd Abma, UMC Utrecht Metabolics Centre [2012]

“Version Control with Subversion”, Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato [2012]

“Subversion and Trac Tutorial, Project in Automatic Control, FRT090”, Johan Akesson [2012]

“Using ToroiseSVN (tutorial)”, IgorExchange [2012]

# Installation

The Trac system can be accessed via a webbrowser you do not need to install any software for this purpose, besides a webbrowser of your choice.

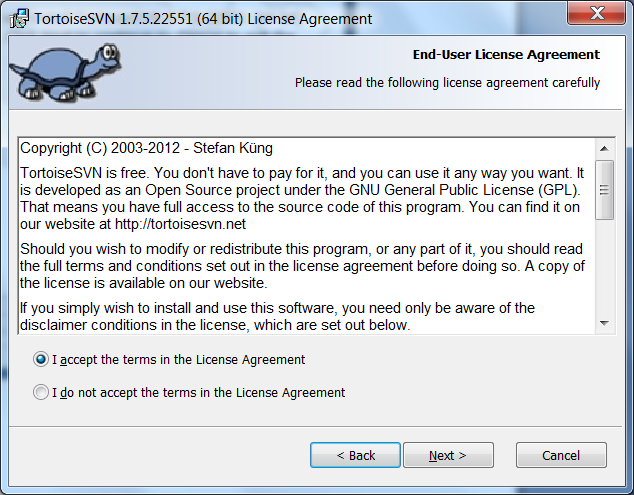
For using Subversion you need to install a Subversion client on your personal computer. The most used client for Windows is TortoiseSVN. TortoiseSVN integrates in the Windows Explorer. All SVN actions will be available from the right click menu in the Explorer

TortoiseSVN can be downloaded from <http://tortoisesvn.net/downloads.html>. Select the right installer for your personal computer.

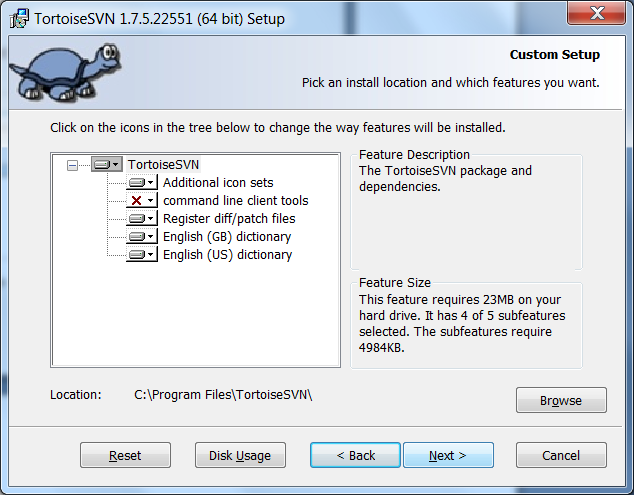
Start the installer, and press Next.



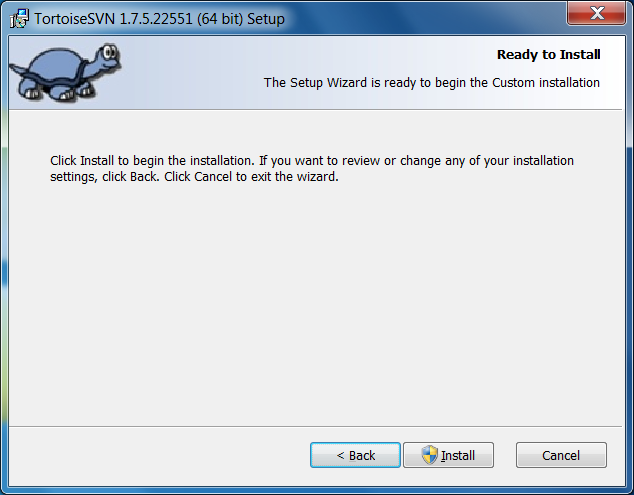
Accept the terms and press Next.



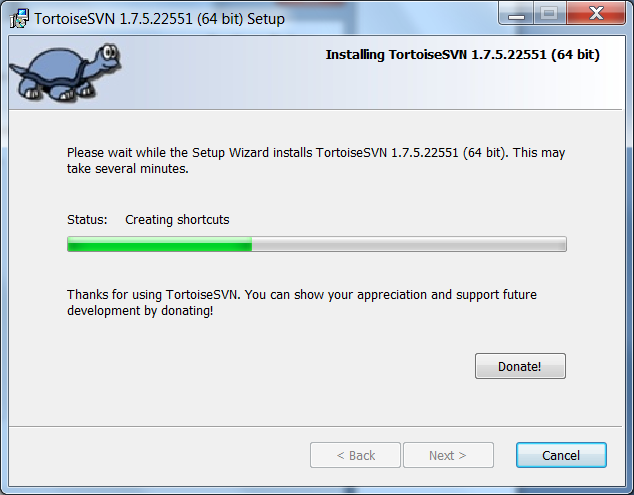
Press Next.



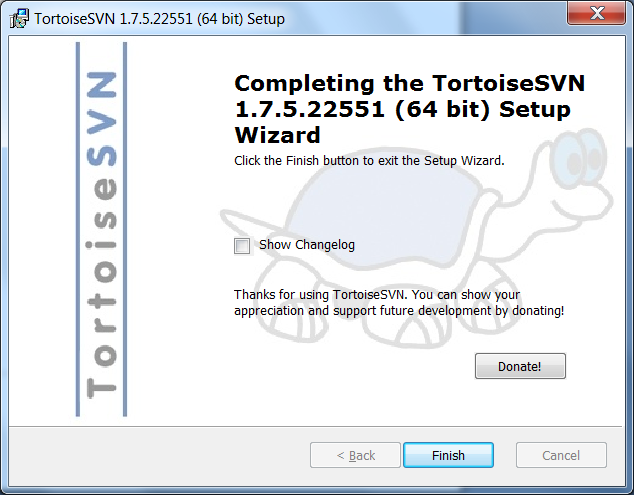
Press Install.



Wait for the installation to finish.



Press Finish.



The ToirtoiseSVN client is now installed. You need to restart your personal computer.

## Trac

In larger, software-oriented projects involving several developers, it is usefulto have a tool for planning, coordination, and documentation of the work. Forthis purpose, you will use Trac. Trac is a web and wiki-based planning tool inwhich tickets/bugs can be registered and assigned to developers, and milestonesdefined. Trac is also integrated with Subversion.

The user’s guide for Trac is integrated with the system itself. The followingguidelines summarize basic usage of Trac.

* Read the Trac documentation, and in particular the material related to wiki formatting and Trac references.
* The name space for wiki-pages is global. Therefore, it is important to use a convention that does not introduce conflicts. Here we use the CamelCase (<http://en.wikipedia.org/wiki/CamelCase>) strategy for naming wiki pages in combination with name mangling where CamelCase names are concatenated and separated with slashes (’/’) to create a hierarchical structure. For example, if the wiki page ’GuideLines’ is created at the top level, a typical name of a sub-page is then ’GuideLines/Trac’. Notice that ’GuideLines/Trac’ is considered by Trac as a global name and not a hierarchical one.
* Tickets typically follow a life cycle:
  + The ticket is created and a description is added.
  + The ticket is accepted by a developer that undertakes to perform the task specified in the ticket.
  + The ticket is resolved, usually by ’fixed’, but there are also other options:
    - If a duplicate ticket is created, it can simply be resolved as ’duplicate’. This resolution is used also when a ticket is divided into two or more new tickets. Sometimes it is useful to create a very general ticket which is then divided into more specific ones. In such case, make sure that the new tickets refer back to the original ticket.
    - For some tickets it may be decided that the task will not be performed, for some reason. In this case, the ticket is resolved as ’wontfix’.
* Make frequent use of cross-references between tickets, wiki pages and subversion revision numbers. In particular, it is important to add a reference from a ticket to a subversion commit whenever a commit is made relating to the particular ticket. Conversely, try to always add a reference to a ticket in subversion commit comments. This makes it much easier to search and trace the origin of code changes. Tickets are referred to using the syntax ’#1’ (or ’ticket:1’), revisions by ’r1’ (or ’chageset:1’). Please also see the Trac documentation for more information on wiki formatting and Trac references.
* The ’component’ field of a ticket is used to classify tickets and group them into different categories. Different projects typically have the need for different components and are accordingly created based on the specifics of the particular project. Since adding components requires admin privileges, please contact the Trac administrator if new components need to be added.
* Once a ticket has been created, it is typically assigned to one of the team members by entering his/hers user ID in the Owner field of a ticket. In this way, Trac provides a means to allocate tasks to team members and for team members to get an overview of their individual tasks.

### View progress

Trac gives you a number of convenient ways to stay on top of events and changes within a project. You can set milestones, and view a roadmap of progress towards them (as well as historical achievements) in summary. There is a timeline of individual changes so you can see the order of events, starting with the most recent. Trac supports RSS for content syndication: allowing people to subscribe to those changes outside Trac itself, as well as email notification.

### Online repostitory viewing

Trac gives a highly usable browsing and management front-end for Subversion, a modern open-source version control system. Trac gives you clear and elegant code highlighting and file comparison, so you can easily see how files differ. Using plugins, Trac also supports other version control software.

### User management

Trac has a simple permission system to control what users can and can't access.

### Features provided through plugins

There are a range of plugins available supporting additional features, from anti-spam to Gantt charts and time tracking.

### Timeline

The timeline provides a historic view of the project in a single report.

It lists all Trac events that have occurred in chronological order, a brief description of each event and if applicable, the person responsible for the change.

The timeline lists these kinds of events:

* Wiki page events — Creation and changes
* Ticket events — Creation and resolution/closing (and optionally other changes)
* Source code changes — Repository check-ins
* Milestone — Milestone completed

Each event entry provides a hyperlink to the specific event in question, who authored the change as well as a brief excerpt of the actual comment or text, if available. It is possible to filter the displayed events with the various filters in the option panel.

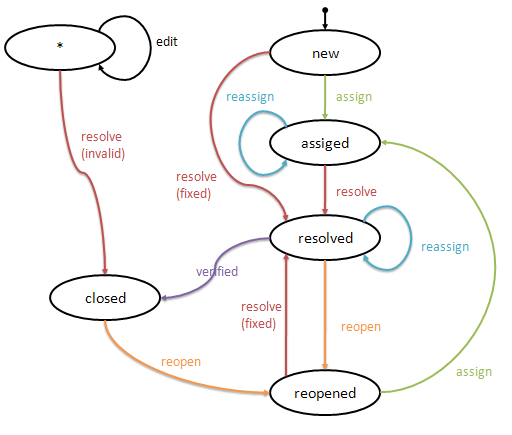
### Roadmap

The roadmap provides a view on the ticket system that helps planning and managing the future development of a project. Basically, the roadmap is just a list of future milestones. You can add a description to milestones (using WikiFormatting) describing main objectives, for example. In addition, tickets targeted for a milestone are aggregated, and the ratio between active and resolved tickets is displayed as a milestone progress bar.

On the roadmap page you will see when the milestone should be finished, how many tickets are still open for the particular milestone and how many tickets are closed. And the total amount of tickets for a milestone are shown.

### Workflow

The status of a ticket will change over time starting as “New” and ending as “Closed”. The following workflow defines the available status changes.



## Subversion

A version control system is useful when working on a collection of files that evolves over time, in particular if several people are modifying the files. For example, version control is often used in software projects to store the code in a way that several developers can access and contribute to the code without the need to send files by e-mail etc. Another example is when you are writing a paper with your colleagues and you need to efficiently and safely share your additions to the paper. A version control system also keeps track of the history of your contributions; most version control systems can be used to retrieve the state of your files as of a given date. This means that there is no need to create local back-up copies. To make it short and sweet: version control means you can relax.

The version control system Subversion is installed on the repository hosting server. An excellent source of knowledge about using Subversion can be found at <http://svnbook.red-bean.com/>. Here you will find basically everything you need (and more) to use Subversion efficiently. Some basic guidelines for using Subversion are:

* A typical Subversion repository has three directories at the top level: ’trunk’, ’tags’, and ’branches’. Development usually takes place in ’trunk’, so all the project files should reside there. ’branches’ are typically used to create snapshots of the trunk to enable concurrent development, and ’tags’ are used to mark releases. See the Subversion book for more info.
* Try to make atomic commits, meaning that each commit has a well defined scope. If several unrelated changes have been made to the code it is good to commit them separately.
* A good practice when using Subversion in association with Trac is to always work with a particular ticket in mind. The DTI Subversion uses a custom configuration that forces a user to do commits in relation to tickets. The log-message of the commit must contain a reference to an active ticket. This makes it much easier to trace the origin of code changes.

### Subversion architecture

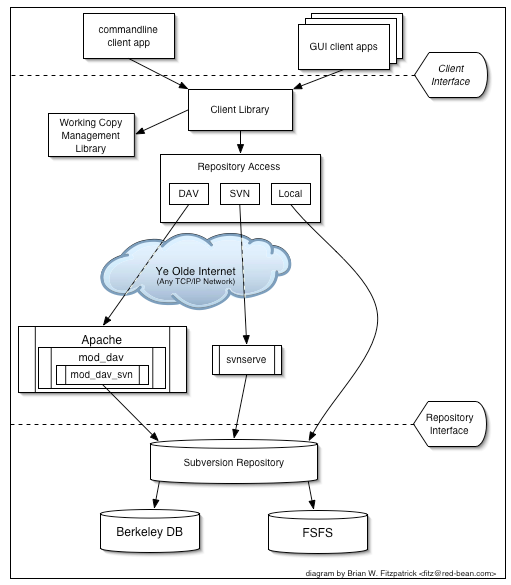


Figure 1: Subversion architecture

On one end is a Subversion repository that holds all of your versioned data. On the other end is your Subversion client program, which manages local reflections of portions of that versioned data. Between these extremes are multiple routes through a Repository Access (RA) layer, some of which go across computer networks and through network servers which then access the repository, others of which bypass the network altogether and access the repository directly.

Each project has its own Subversion repository. In this tutorial we will use one repository called “demo\_project”.

### Subversion Clients

Just for your knowledge, the basic (official) Subversion client is available for all major platforms (Windows, MAC and Linux) and can be downloaded from: <http://subversion.apache.org/packages.html>. This official client is just a command-line program with all the basic tools you need to do version control with. I recommend not using this client.

For our convenience, we will use TortoiseSVN client instead of the official client. Tortoise SVN is a GUI based SVN client which integrates in the Windows Explorer. All SVN actions will be available from the right click menu in the Explorer.

### Text vs. Binary data

Both text and binary data can be stored in a Subversion repository, however, Subversion is only able to track changes in text files as it has no idea of how to interpret (changes in) binary data (doc, docx, zip, etc). The binary 'blobs' are automatically flagged as being binary. The Subversion Server is configured use the lock-modify-unlock model for these files. Since AutoCAD and Matlab files are hard to merge these files will use the lock-modify-unlock model (see Appendix C for a list of extensions).

In this model, the repository allows only one person to change a file at a time. This exclusivity policy is managed using locks. Tester1 must “lock” a file before he can begin making changes to it. When Tester1 has locked a file, Tester2 cannot also lock it, and therefore cannot make any changes to that file. All Tester2 can do is read the file and wait for Tester1 to finish his changes and release his lock. After Tester1 unlocks the file, Tester2 can take her turn by locking and editing the file. Figure 1.3, “The lock-modify-unlock solution” demonstrates this simple solution. The locks are locally forced by altering the read-only property of a file.

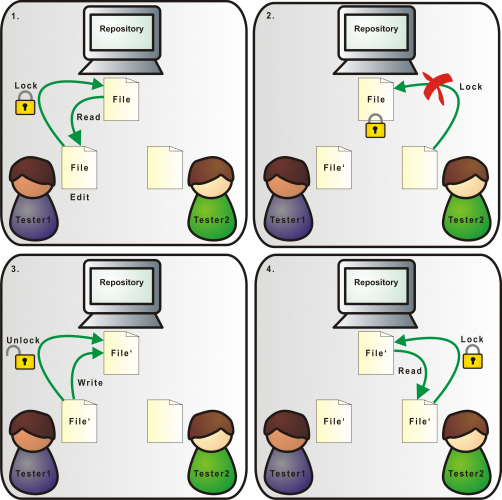


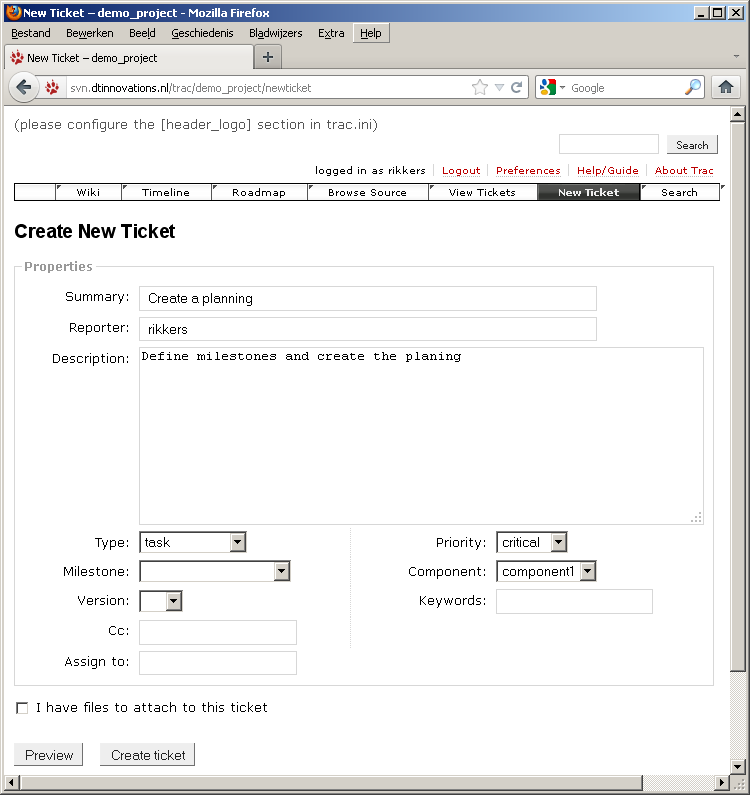
Figure 2: The lock-modify-unlock solution

# Appendix A: How to use Trac

The DTI Subversion server can be accessed via <http://svn.dtinnovations.nl/trac/>. This page will provide an overview of all projects.

## Create a ticket (task, problem report or change request)

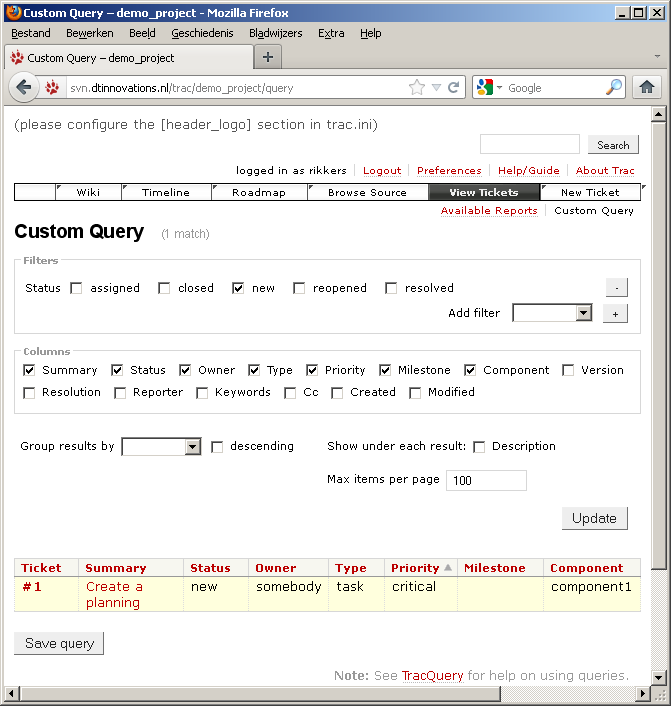
To report a defect, enhancement or task create a new ticket by clicking on the “New Ticket” link and fill in the following fields



Click on the “Create ticket” link to permanently create the ticket.

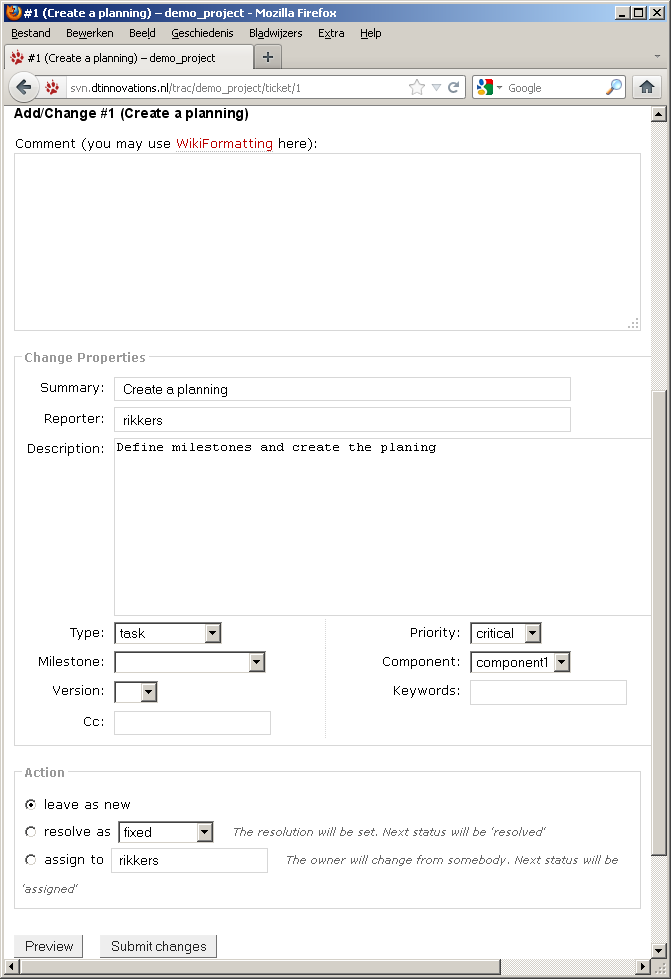
## View tickets

Click on the “View tickets” link to get an overview of the tickets for the project. Below “Custom Query” you will see a box named “Filters” which allows you to filter for specific tickets. Uncheck all filters, but leave the “New” filter checked. Press ¨Update” to apply the new filters. You will now only see tickets which are new.



## Modify a ticket

By clicking on the ticket, you will be able to edit the properties or add comments to the ticket.



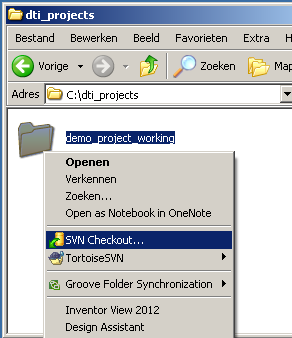
# Appendix B: How to use TortoiseSVN

## Checkout a working copy

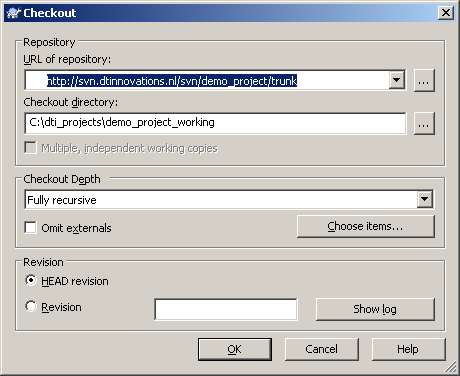
Whenever you want to work on a project, you first need to create a working copy of the project files on your personal computer, and then you can modify the files in your working copy and later send those changes to the repository on the server. The repository hosting Subversion server can be accessed via <https://hanautomotive.repositoryhosting.com/>

Every project will have a default directory structure imported into the repository. Any additional folders and files will have to be added by you or your team. You first need to check out the contents of the trunk directory from the repository into a working copy on your computer. You will then copy any of your project's files into this working copy, and then commit those changes to the repository. From this point on, any changes you make to the repository will be made by first making changes to the working copy of your project on your computer and then committing those changes to the repository.

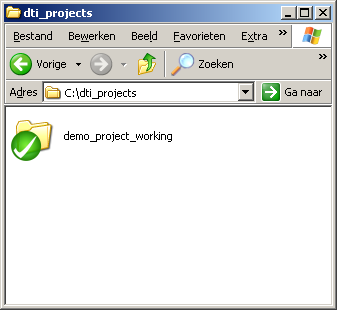
1. From Windows Explorer, create or change to a directory on your computer under which you want to store any working copies of HAN Automotive projects. You might want to call this directory HANAutomotive\_projects.
2. Create a directory your computer under the HANAutomotive\_projects directory to hold the working copy you are about to check out. Call this directory demo\_project\_working. Because this new directory will contain the working copy of your project, it is best to name that directory with the \_working suffix so that it is obvious that this directory contains a working copy of a project.
3. Right click on the demo\_project\_working directory and choose SVN Checkout.



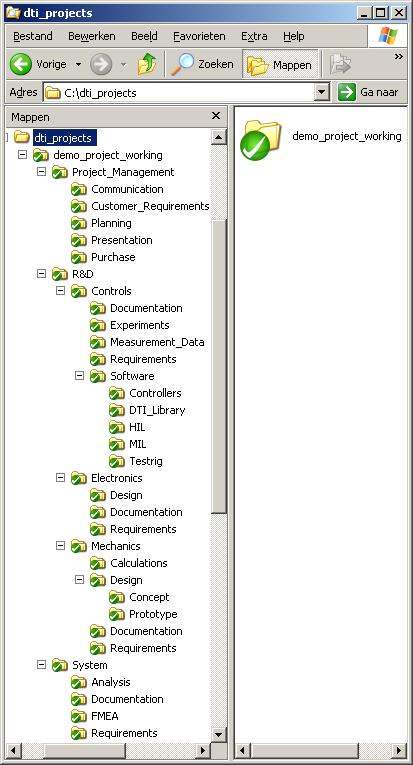
1. Since right now you just want the contents of the /trunk directory in your working copy, change the "URL of repository" setting to <https://hanautomotive.repositoryhosting.com/svn/demo_project/trunk> and click the OK button.
2. Fill in your credentials, tic the Save authentication and click the OK button.



1. You should now have a green check mark icon next to your demo\_project\_working directory. At this time, the default directory structure will be in this directory and a hidden directory called .svn. You should never delete or modify the .svn directory or any files within the .svn directory, as they are used internally by TortoiseSVN.



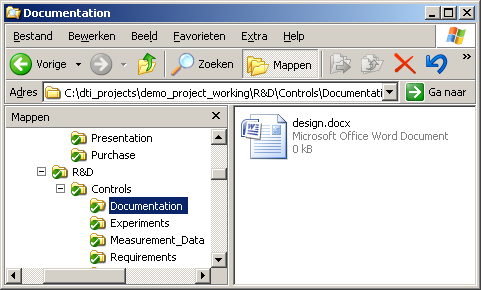
1. To maintain consistency throughout projects a default directory structure has been defined. This structure is imported into projects during creation.



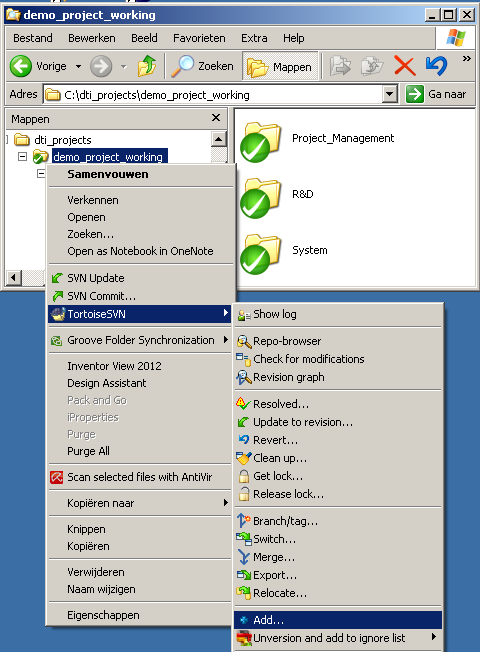
## Adding files to the repository

Now that you have checked out a copy of the /trunk directory from the repository into your working copy directory, it is time to add some files to the repository.

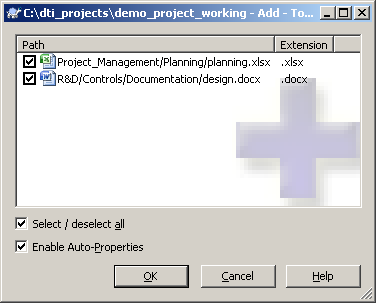
1. Using Windows Explorer, copy the files necessary for your project into your working copy directory. The example working copy directory now looks like this:



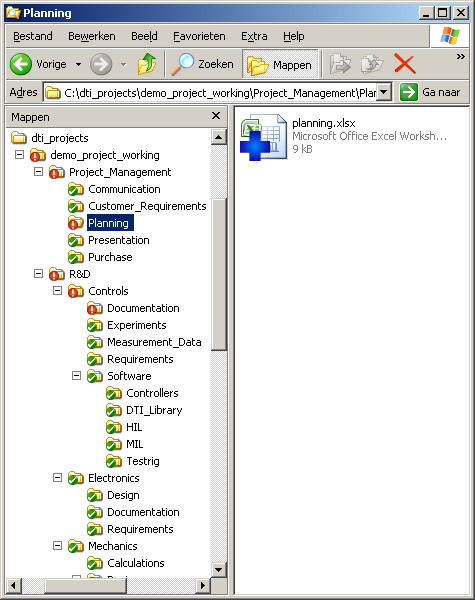
1. Right click on the name of the working directory (demo\_project\_working) and choose TortoiseSVN --> Add. In the dialog, check the box for each file you wish to add to the repository.



Note: Certain types of files, like temporary files and non-essential files, should not be added to the repository. Just make sure not to check the box next to these types of files so that you don't add them to the repository. You can also tell TortoiseSVN to ignore certain files. See Ignoring files in the TortoiseSVN manual for information on how to do this. The dialog box should now look something like this:



1. Click the OK button. The icons representing the files you have just added are now be marked with a blue '+' sign and the demo\_project\_working directory now is marked by a red exclamation point icon, like this:



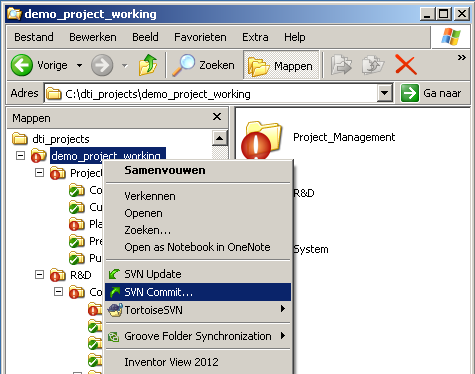
The red exclamation point indicates that the directory contains files that have been changed since the last time the working copy was updated (see Updating topic below).

1. At this point, you have told TortoiseSVN that you wish to add these files to the repository, but you have not yet actually added them to the repository. To do that, you have to commit the changes back to the repository. See the next topic for an explanation of how to do this.

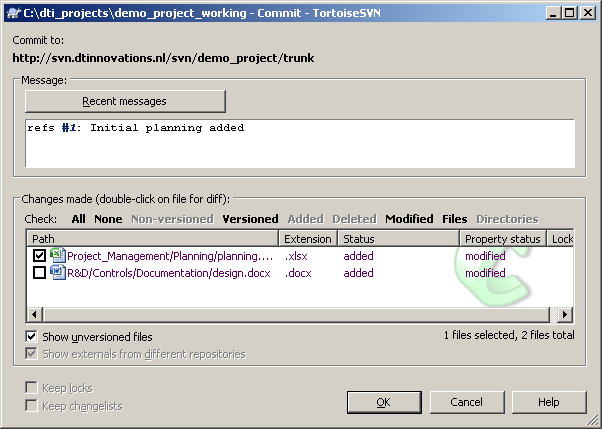
## Commit your changes to the repository

After you have added or removed files or made changes to your code (and tested those changes to make sure they work), you should commit those changes back to the repository so that others can access the changes. In this example, you have now informed TortoiseSVN that you wish to add “planning.xslx” and “design.docx” to the repository, but you now must commit that change to the repository.

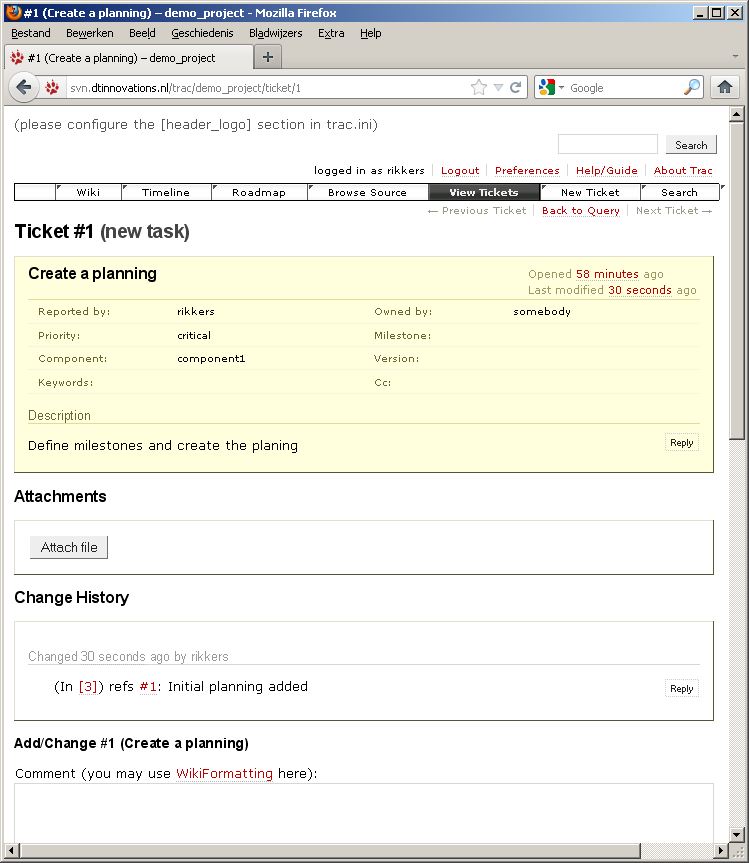
1. From Windows Explorer, right click on the demo\_project\_working directory and choose SVN Commit.



Type a commit message in the dialog. The message MUST have a minimum of 10 characters and MUST contain a reference to an open ticket (“refs #<number>”). The reference will immediately form a hyperlink to the Trac ticket. Make sure that the "Changes made" list has a check next to the file(s) you want to add, for example only “Planning.xlsx”.



1. Click the OK button. If the commit worked, you will now see a dialog that says "Finished" and that indicates that the files were added to the repository.
2. The commit message will automatically be added to the Trac ticket as change note.



Notes:

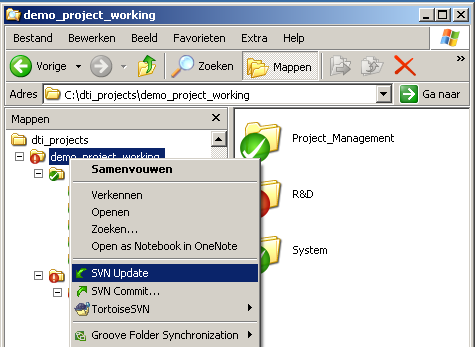
* If you want to commit changes from only one file in your working copy (instead of the entire directory), you can right click on the file itself and choose SVN Commit.
* Instead of right clicking on a file and choosing TortoiseSVN --> Add, you can also add a file to the repository by checking the box next to the file in the "Changes made" section of any commit dialog.
* In order to commit changes to a project, you must be the owner or a maintainer of that project.

## Update the working copy to a revision

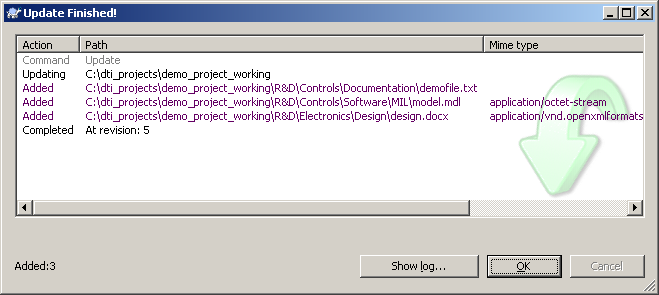
If your project has additional project maintainers other than yourself, these people are permitted to commit changes they have made to code for your project back to the repository. In order to be certain that the code you edit contains all the recent changes made by other project maintainers, you should update your working copy regularly.

Note: Because updating your working copy is a read-only operation, you do not have to be a project maintainer or have a Subversion account to follow these steps.

1. Right click on the working directory of your project (demo\_project\_working) and choose SVN Update.



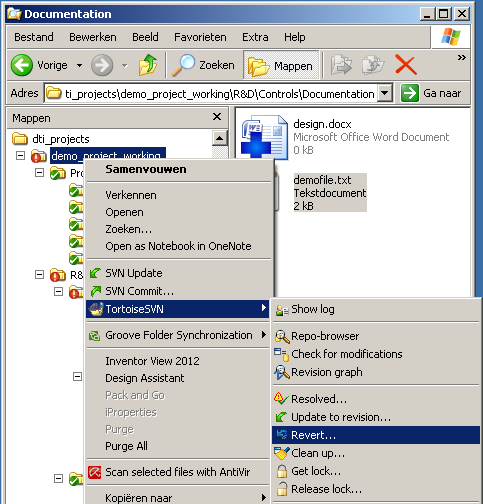
1. If, after updating your working copy, you receive any error messages or warning messages in the dialog, there may be a conflict between a file in your working copy and the version of the file in the repository (often by someone else). This can happen if you have edited a file since you last updated your working copy and during that time a change to that same file was committed back to the repository. If this happens, you will need to resolve the conflict in the two files. See Resolving conflicts in the TortoiseSVN online manual for more information on conflicts and how to resolve them.



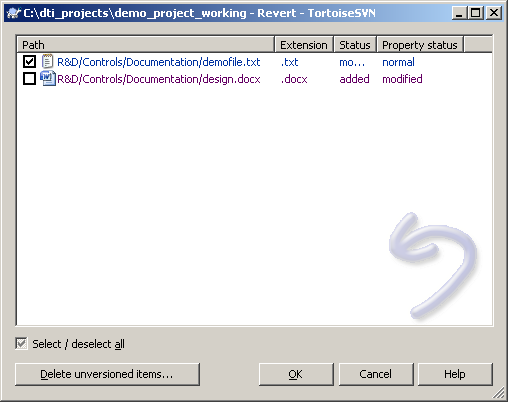
## Revert local changes (working copy)

After making changes to the files in your working copy, either manually or by applying a patch, you may wish to return those files back to the same state they were in after the last (or any prior) revision. Any changes made to files in your working copy that have not been committed will be lost (though you have the option of reverting changes to selected files in your working copy).

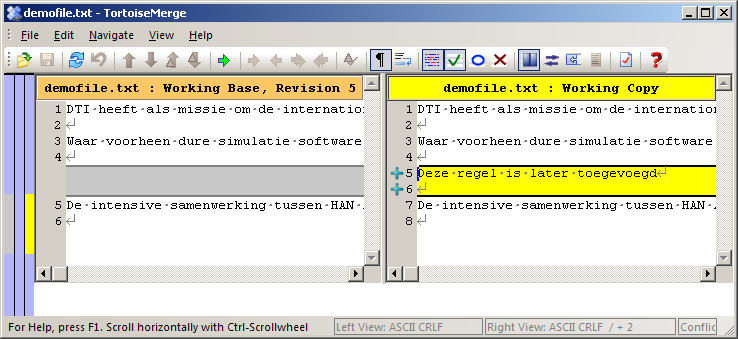
1. From Windows Explorer, right click on your working copy directory (demo\_project\_working) and click TortoiseSVN --> Revert



1. The dialog will contain a list of files in your working copy that have changed since you last updated your working copy. Make sure that each file you want to revert is selected. If a file is not selected it will not be changed by the revert operation. When you have selected all of the files you wish to revert, click the OK button.



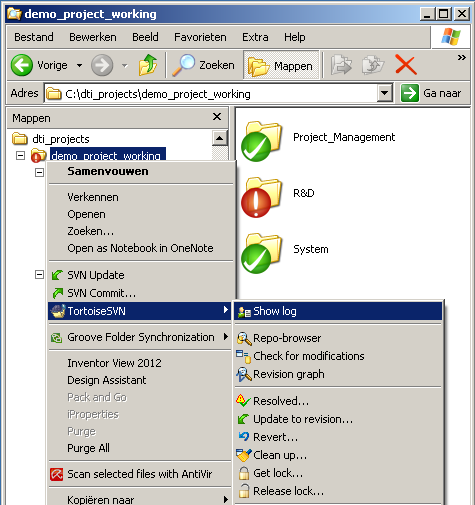
1. You can view the differences between the last (or any prior) revision and the working copy by double clicking a file in the list. This will only work for text files (no binary)



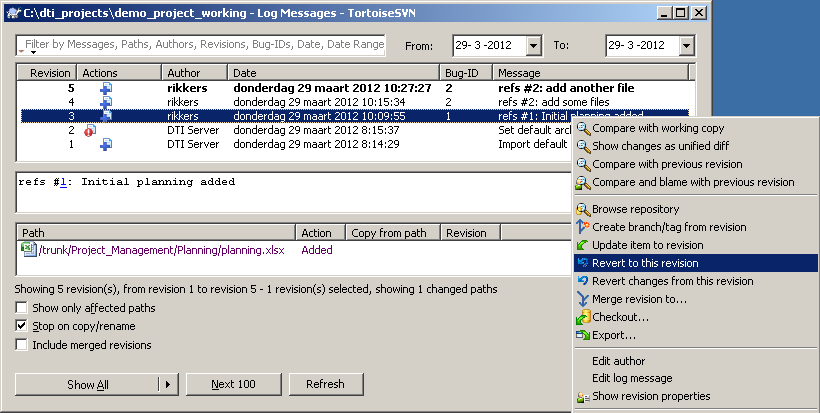
## Revert changes from a revision (repository)

If at some point you or someone messed up the repository, by committing erroneous code or deleted files that should not be deleted and you would like to undo those changes, you can revert the changes from this revision.

1. From Windows Explorer, right click on your working copy directory (demo\_project\_working) and click TortoiseSVN --> Show Log

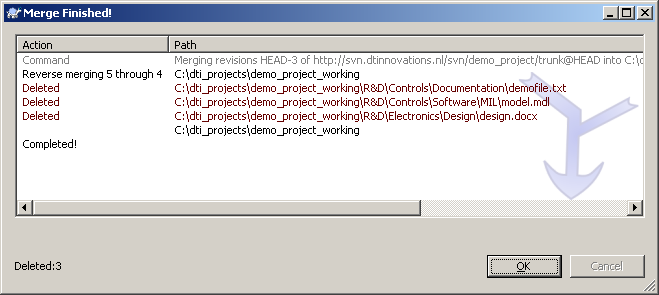


1. The dialog shows the history of the repository. Find the revision in which the errors were introduced, right click on this revision and click “Revert to this revision”. Agree with the changes to be make in the confirmation window. The changes will be reverted by reverse-merging into your working copy.





1. At this point, you have applied changes to you working copy that will undo the changes from the selected revision, but you have not yet actually reverted them in the repository. To do that, you have to commit the changes back to the repository. See the Commit the changes to the repository topic for an explanation of how to do this.



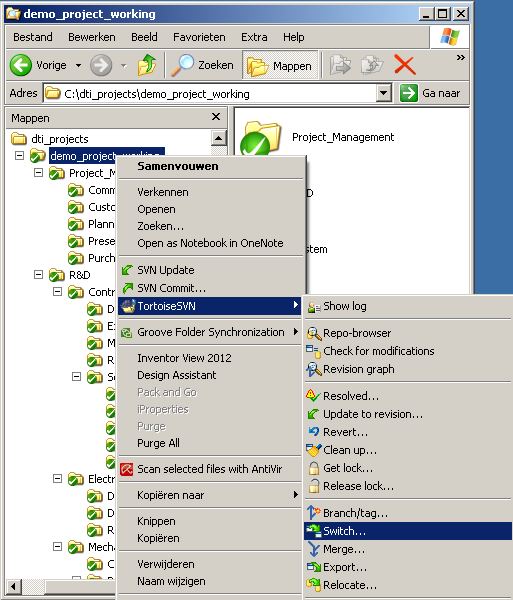
Now we have reverted all changes made since a certain revision. Another option is to revert the changes from a single revision. Select “Revert changes from this revision“ to do so.

## Switching your working copy

It is often necessary to work on several different branches of your project and/or the trunk, and there are two ways to do this.

* Manually create several directories on your computer and check out working copies based on different parts of the repository into each of them.
* Keep only one working copy on your computer and switch the part of your repository that that working copy represents. To use this method, follow these steps:

1. Using Windows Explorer, right click on the working copy directory (demo\_project\_working) and click TortoiseSVN --> Switch.



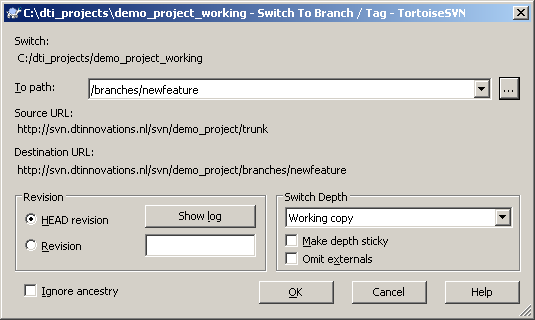
1. In the "To URL:" field, enter the location in your repository that you want to switch your working copy to represent. For example, if your current working copy points to

<https://hanautomotive.repositoryhosting.com/svn/demo_project/trunk>

and you want to switch to the branch newfeature, you would enter

<https://hanautomotive.repositoryhosting.com/svn/demo_project/branches/newfeature>

If you are not sure about the exact path that you want to switch to, you can click the "..." button next to the "To URL:" field and the repository browser will pop up and you can select the path this way.



1. Click the OK button and your working copy will be switched to represent the new location. Note that it would be wise to rename “demo\_project\_working” folder to “demo\_project\_newfeature”

## Create a new branch in an existing project

Sometimes you might want to try a new feature without editing the files in the trunk. In this case a new branch needs to be created.

1. Switch your working copy to the branches directory using chapter 6.7. Note that in this case you will need to switch to the <https://hanautomotive.repositoryhosting.com/svn/demo_project/branches> URL and not to the <https://hanautomotive.repositoryhosting.com/svn/demo_project/branches/newfeature> URL.
2. Create a new folder inside the branches directory. For the name you might want to enter “demo\_project\_newfeature2”.
3. Copy the files from the current working version in the trunk directory and paste them in the “demo\_project\_newfeature2” folder.
4. Add the “demo\_project\_newfeature2” folder to the repository using chapter 6.2.
5. Commit the new branch to the repository using chapter 6.3.