QlikView

Use Source Control Management with QV10

How to use SVN+TortoiseSVN in a Qlikview Development Environment

Version: 1
Date: 2010-11-11
Author(s) RVA

”A best practice is a technique or methodology that, through experience and research, has proven to reliably lead to a desired result.”

Contents

1 Motivation

Revision control, also known as version control, source control or software configuration management (SCM), is the management of changes to documents, programs, and other information stored as computer files. It is most commonly used in software development, where a team of people may change the same files.

As the development team of a Qlikview Application grows, the need for SCM grows as well. This document describes how to use QV10 together with the popular Open Source version control system SVN and the SVN-client TortoiseSVN. It will describe typical developer/multi-developer scenarios and how one can take advantages out of SCM in these situations.

To simplify work with SCM QV10 introduced the feature Qlikview Project Files. This new feature is described in the Qlikview Developer Reference Manual in section “8.1 QlikView Project Files” on page 59. See screenshot.

8.1 QlikView Project Files

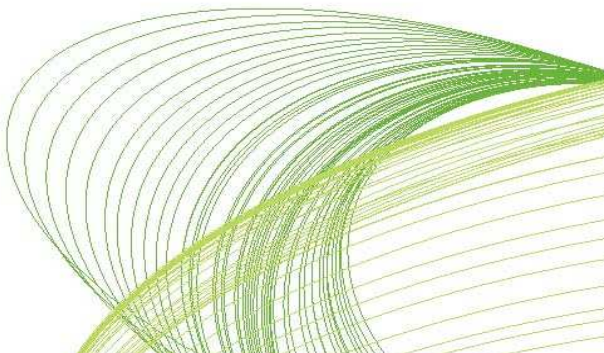
It is possible to save a QlikView document into several files, that can be used for versioning. Each file defines a property of the document, a sheet, an object, the script etc.

Each time the document is opened and an object or a setting is changed, these changes are saved to the different files, making it easy to follow the changes made in the document. This way you can also see who made a change and to which part of the document.

To create these project files you must create a folder next to the qvw file with the same name as the QlikView document and add -prj, e.g. the project folder for a document called Finance.qvw should be Finance-prj.

Note No data from the document will be saved in the project files.

The file **`QlikView.txt`** contains a list of all the objects part of the QlikView document. The different sheets and objects in the list are named after their object ID. The files **`DocProperties.xml`**, **`AllProperties.xml`**, **`DocInternals.xml`** and **`TopLayout.xml`** all contain property settings for the different parts of the document. **`DocBinary.dat`** contains user sensitive data, such as passwords.

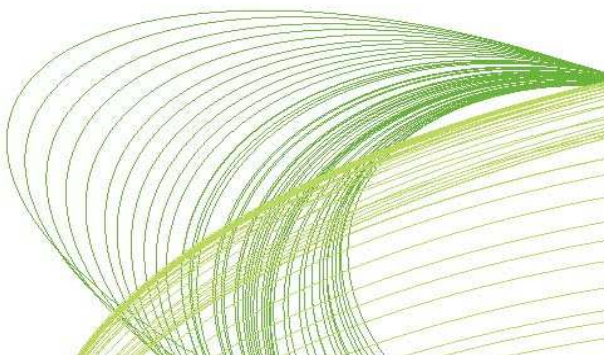


2 Setup Version Control Environment

This document will not describe how to set up an SVN environment. There are lots of good tutorials how to set up a SVN repository in the internet. If you want to setup SVN under Microsoft windows, take for example a look at:

http://www.codinghorror.com/blog/2008/04/setting-up-subversion-on-windows.html

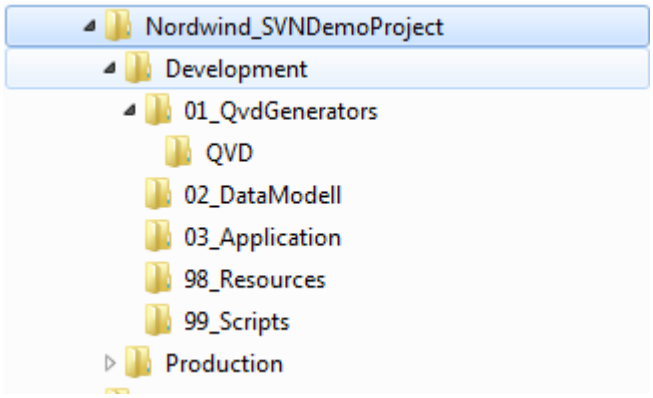
TortoiseSVN (http://tortoisesvn.net/) is an open source SVN-client that integrates seamlessly with Microsoft Explorer. Download the setup and install it on your developers' clients.



3 Starting your Qlikview project

3.1 Define a common folder structure

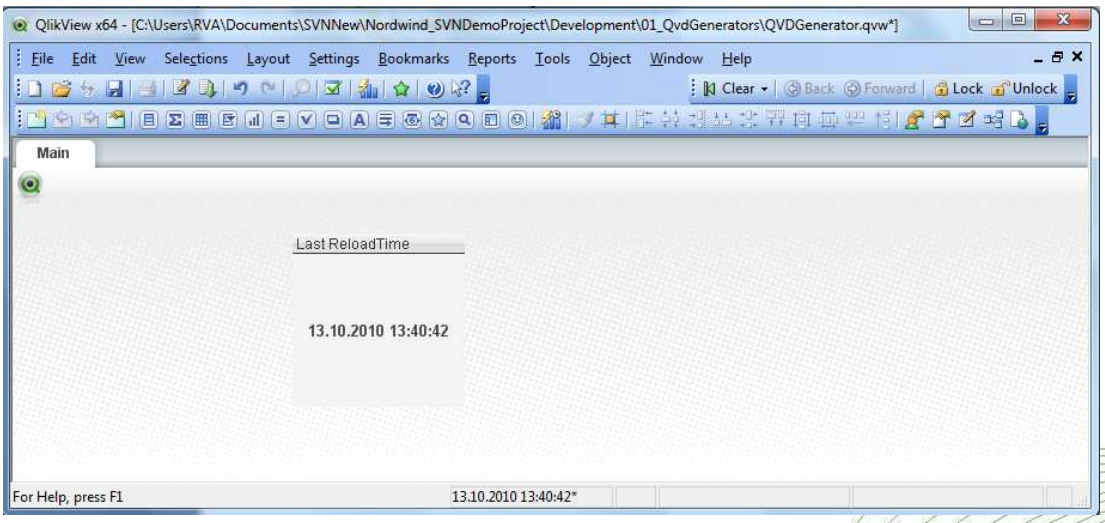
If you are planning for a bigger Qlikview Project, first a clear folder structure should be defined. Following structure is suggested:



Create two folders with identical subfolders: “Development” and “Production”. Within each folder create a folder structure for the three layers typically used in a Qlikview development: QvdGenerators, Datamodel, Application. Add additional folder for external Resources and Scripts as necessary.

3.2 QVD Generator

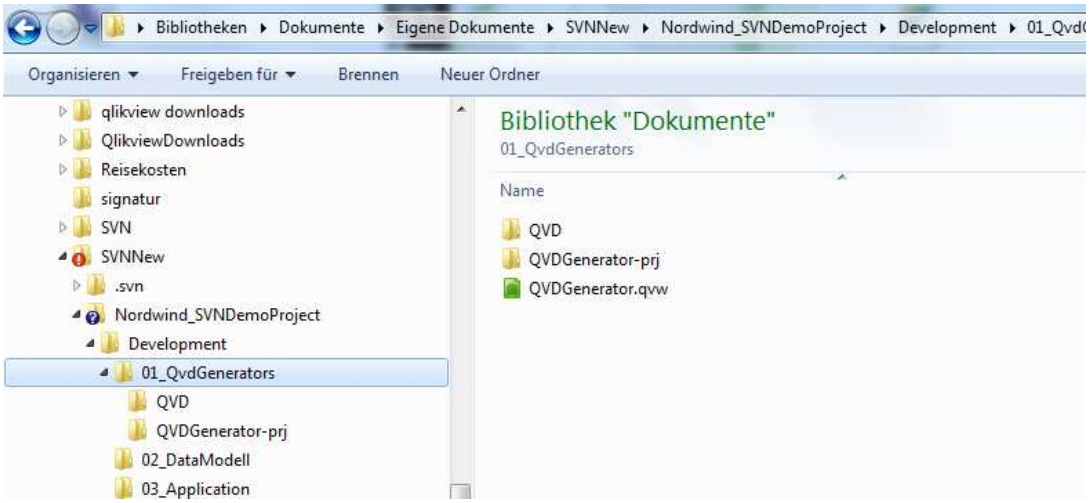
Typically one of the first steps in a project is to develop the QVD Generator to extract data from the source systems. In our example this is a straight forward script to extract some tables from a Microsoft Access database. For documentation purposes I added a textbox with the title “Last ReloadTime” on the frontend. The application is stored as QVDGenerator.qvw.



4 Qlikview Preparations to work with Version Control

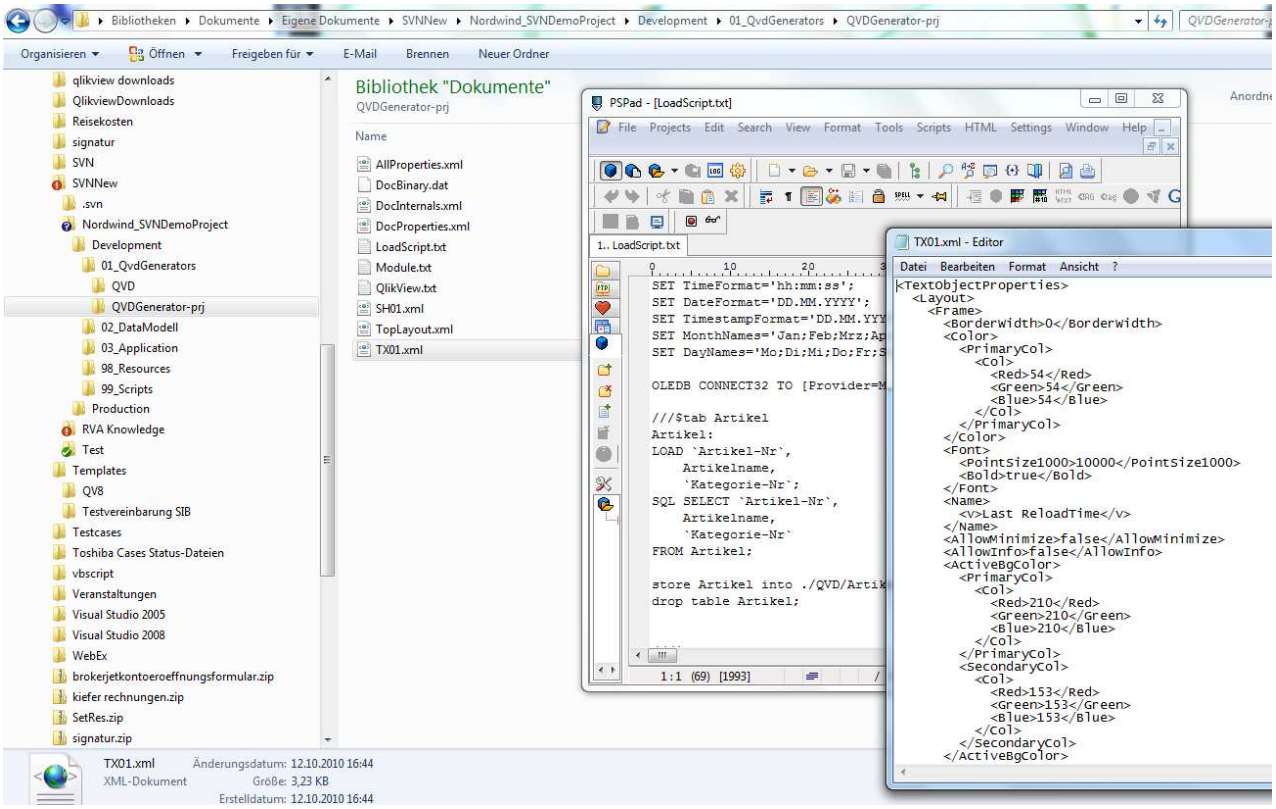
4.1 Creating the “-prj” Folder

The new feature QV10 provides for QV10 is the “-prj” folder. If you create a folder with a name equal to the .qvw-filename plus the suffix “-prj”, Qlikview will store all document-, script- and object-settings into this folder.



1) Create a folder “QVDGenerator-prj” next to your QVDGenerator.qvw
2) Save QVDGenerator.qvw with your QV10 Developer.

3) If you now return to the folder “QVDGenerator-prj” you should see a number of files. Most noticeable the “Loadscript.txt” stores the full script of the QVDGenerator.qvw. The file “TX01.xml” stores the definition of the “Last ReloadTime” textbox that was defined in the .qvw.



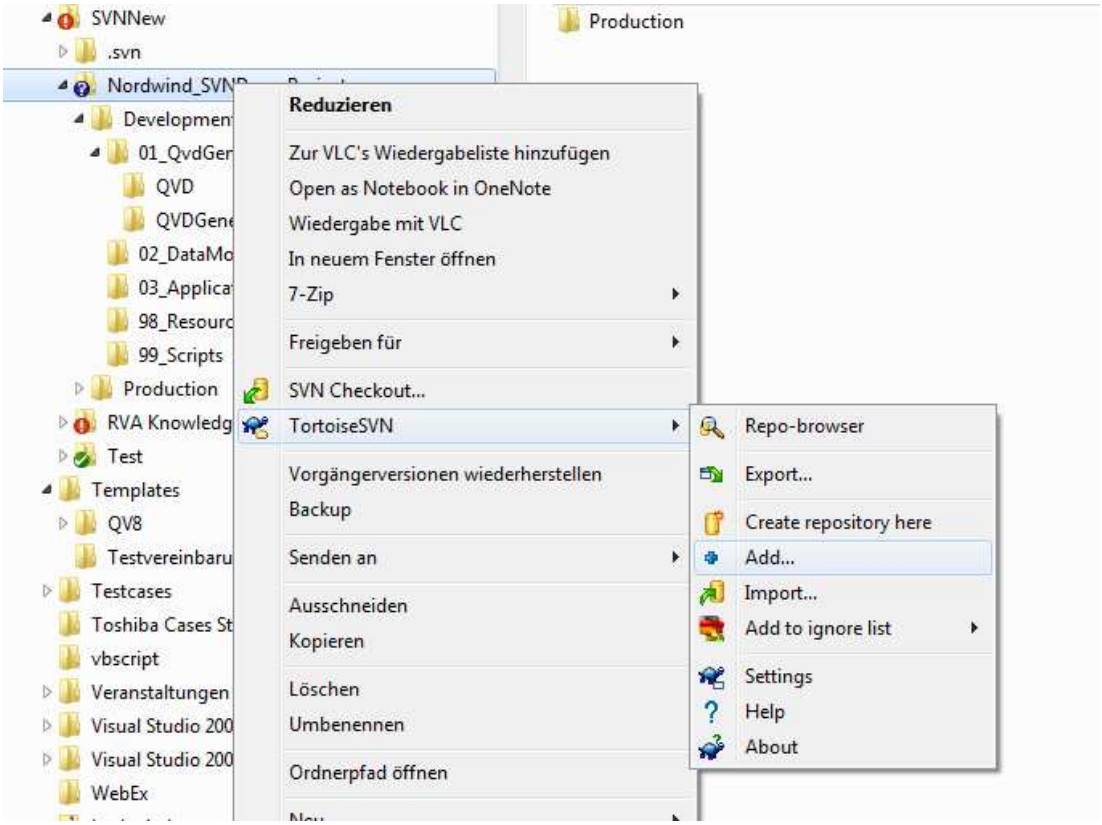
Definition: opening a .qvw with Qlikview 10
If Qlikview 10 finds a “-prj”-folder for the .qvw, it will read the settings from the “-prj” folder. Otherwise it will read the information stored in the .qvw! The files in the “-prj” folder are always updated when you save a .qvw in Qlikview Developer.

The next sections will describe how you can utilize the “-prj” in combination with your Source Control Management Tool.

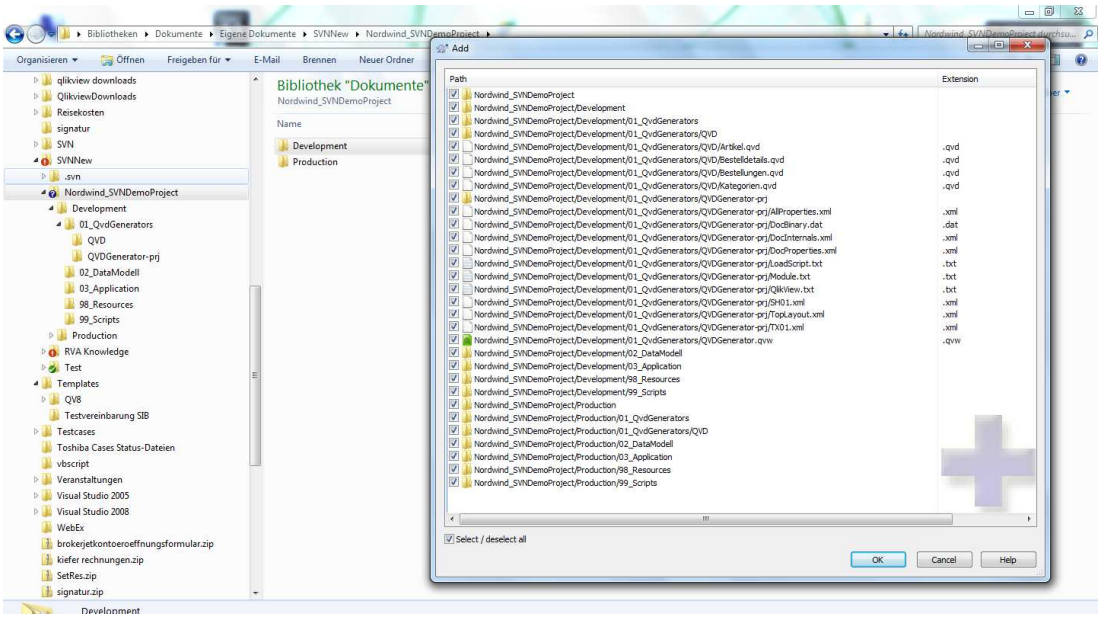
5 Develop a Qlikview Application with SVN

5.1 Check in Project

If you want to version control a project with SVN, the first step is to add the current folder structure and its files to the SVN Repository. Therefore right click your outmost project folder and say: “TortoiseSVN|Add”.



This will pop up a new dialog where you can define which files should be added to the SVN repository.



Take care!

In the screenshot above the .qvw is “added” to the repository as well. In a real word scenario this does not make much sense:

- .qvw files are binary and pretty big. They will increase the size of your SVN repository dramatically. Because they are binary, you will not be able to compare file versions anyhow.

- The .qvw files can be recreated from the “-prj”-Folder. So there is no need to store it twice in the repository.

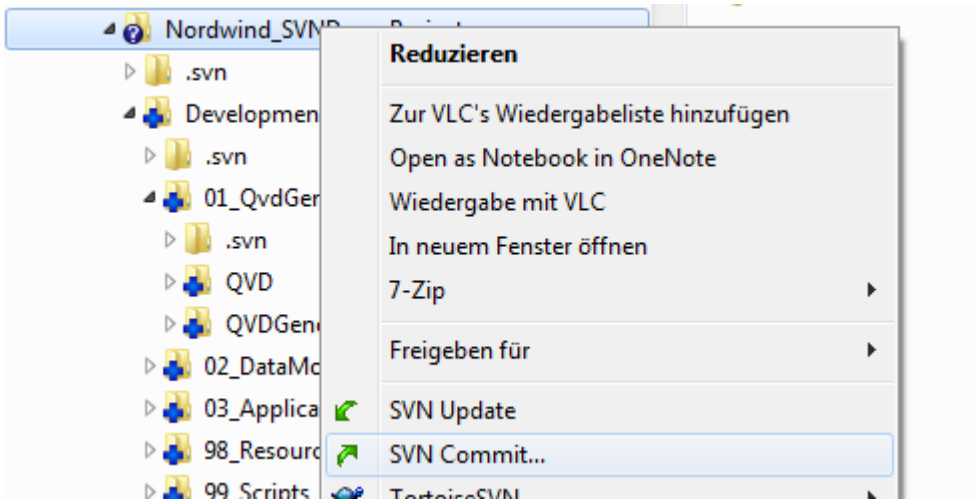
Same for .qvd files; they can be typically be recreated from the source system.

Attention: storing .qvw+.qvd in SVN repository does not make sense from a software development perspective. However please don’t forget to regularly make backups of your .qvw’s+.qvd’s on a backup drive!

5.2 Commit Files to Repository

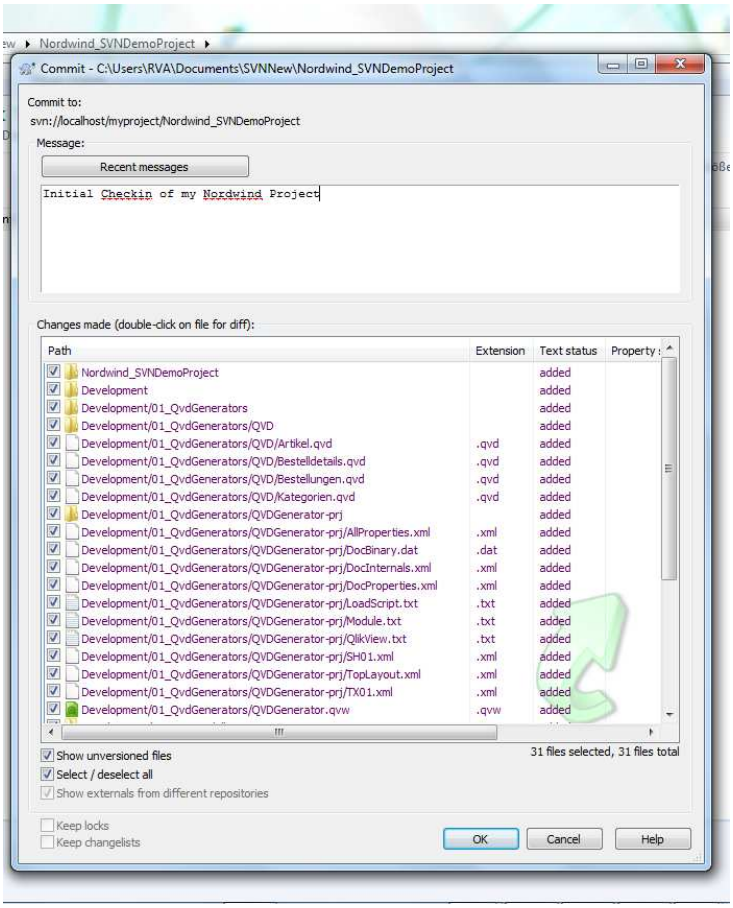
“Add” just prepares a file for the SVN repository. To really commit (== store a version of your file) in the repository is an additional step.

Right click your outmost folder and select “SVN Commit..”.

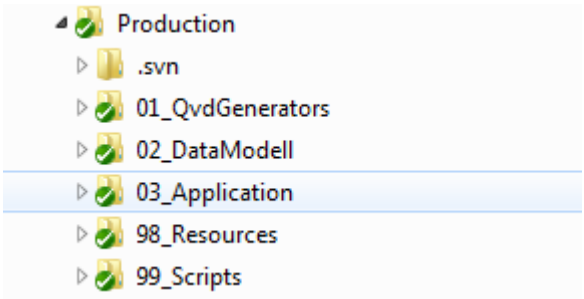


TortoiseSVN will show the following dialog:

- In the upper part of the dialog: Insert a comment describing the changes you did. As this is the first commit the message will be similar to “Initial Checkin of my Project”
- In the lower part of the dialog: double-check which files should be commit/stored in the SVN Repository (Uncheck the .qvw file). These files will be added for versioning.



If everything works out correctly, all versioned files should have a green icon in your Explorer. This way TortoiseSVN visualizes, that your local version is up to date with the SVN repository.



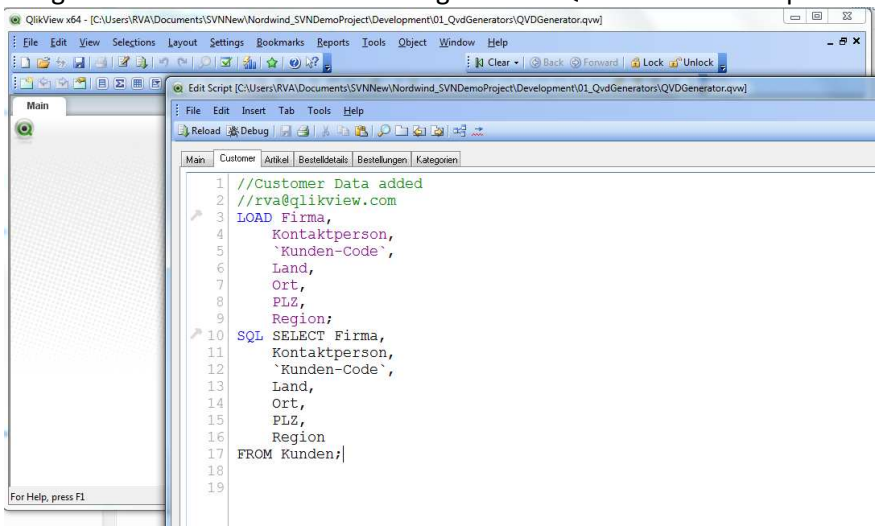
5.3 Changes to the Qlikview Application

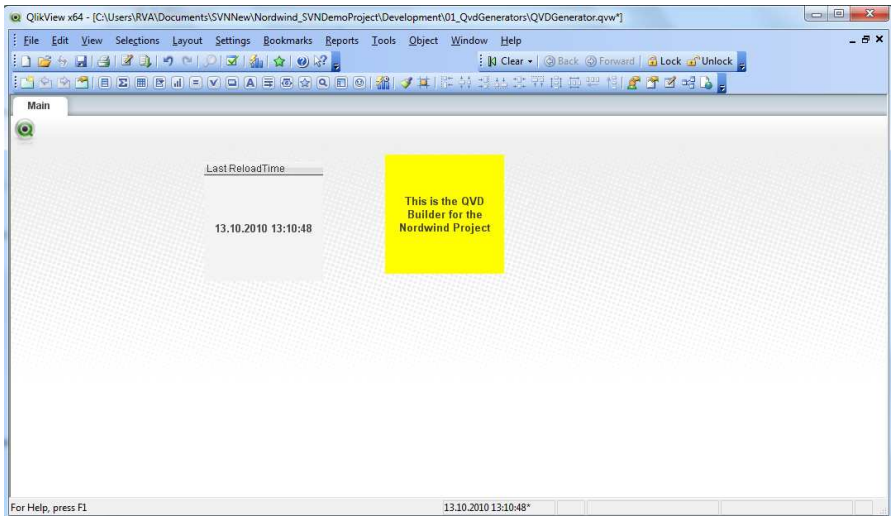
Now, as we have committed the current version of our Qlikview Application to SVN, we can continue to work on our QVDGenerator.qvw .

The following two enhancements should be done:

1) The customer dimension should be extracted from the source system
2) A yellow textbox should be added on the “Main” tab to make everyone clear that this is the QVDGenerator for the “Nordwind Project”.

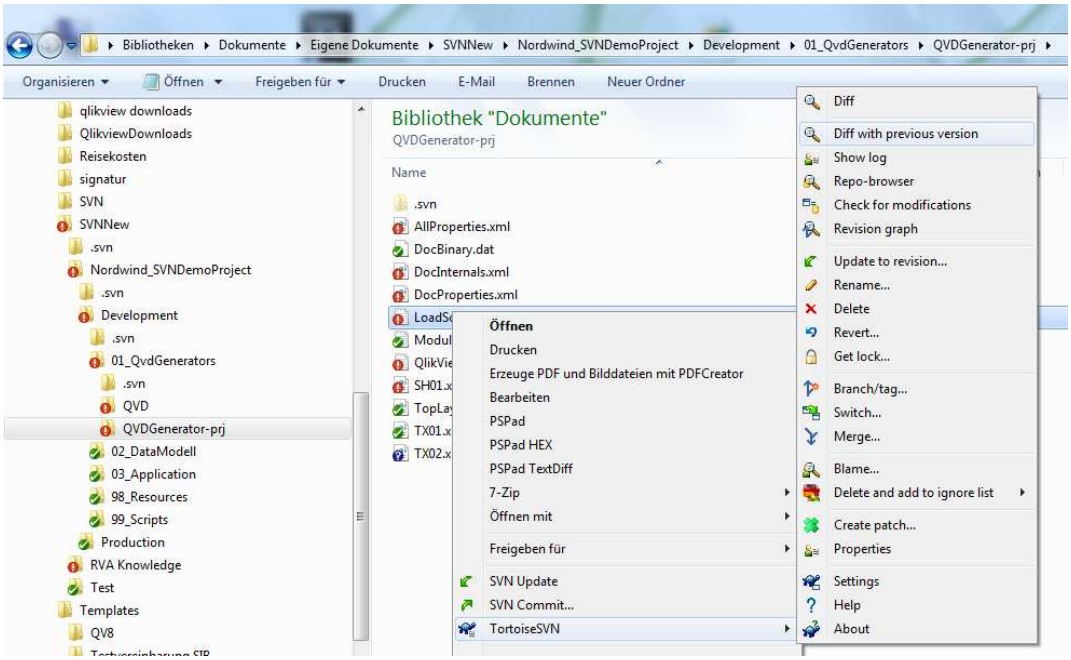
The following two screenshot show the changes in the Qlikview 10 Developer.





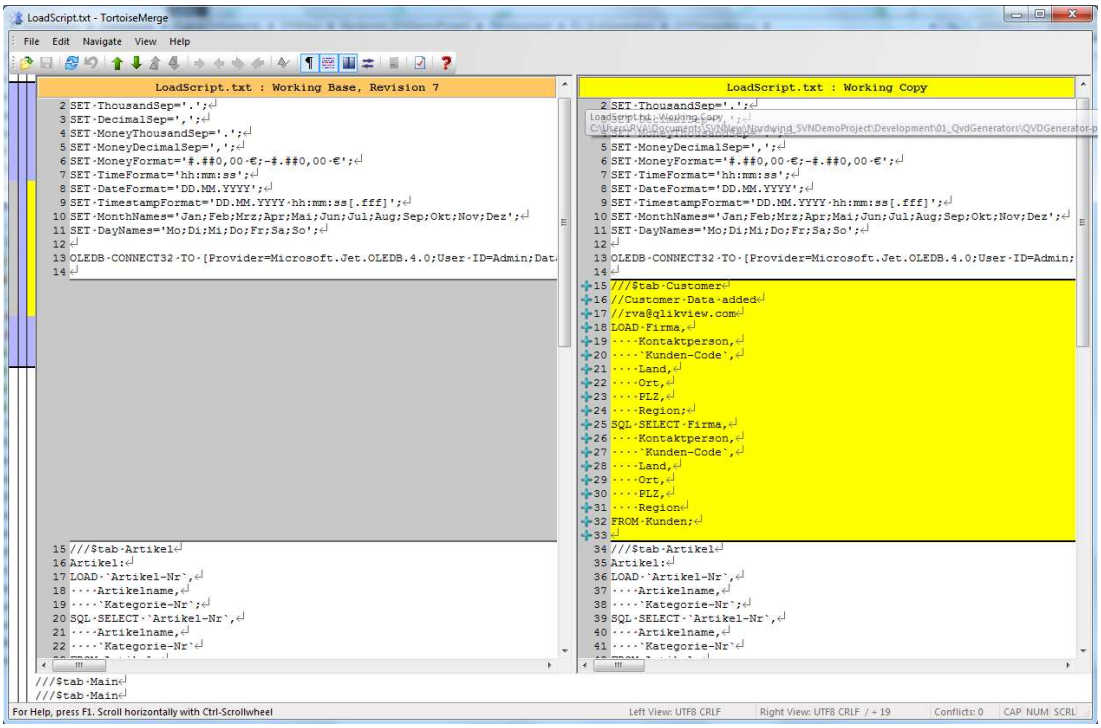
Save the .qvw and close the Qlikview Developer. We want to see what has happened to our files in Windows Explorer.

In the folder “QVDGenerator-prj” we now see the files with various red and a one blue icon.

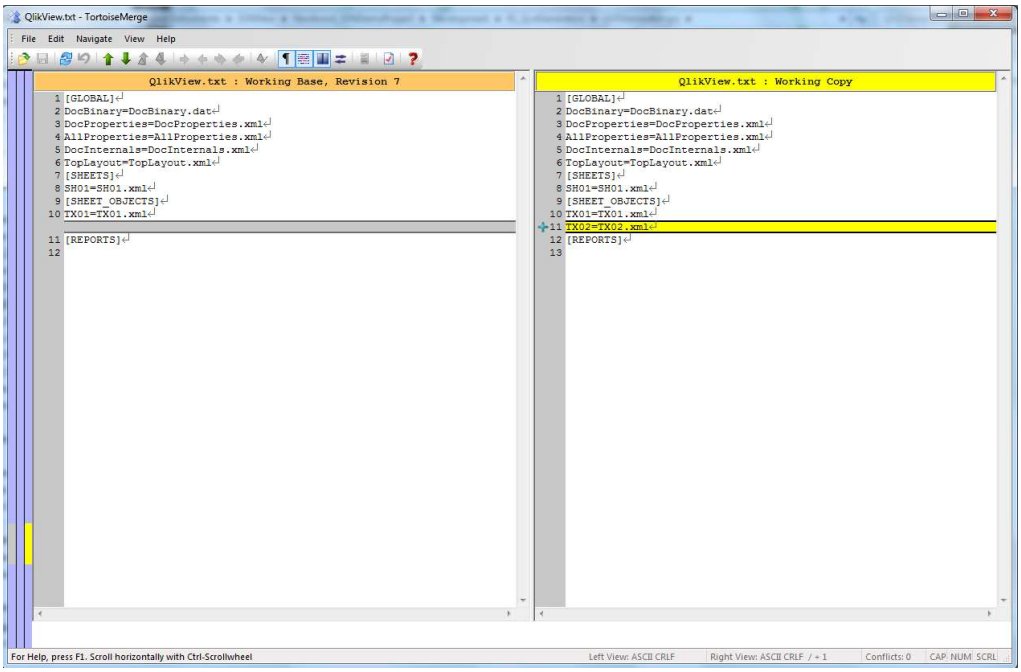


Most noticeable the LoadScript.txt now has a red icon. Right click the file and select “TortoiseSVN|Diff with previous version”.

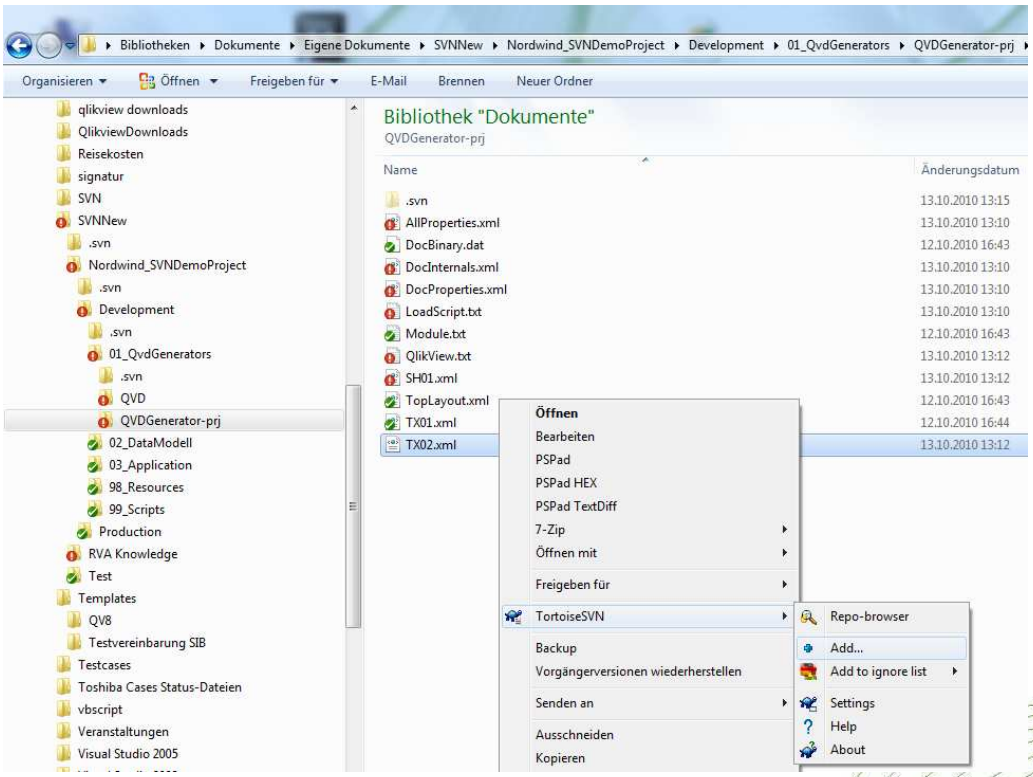
This will pop up the TortoiseSVN Merge Editor. You can easily see the changes we did to our script between our initial version, and our newer version. We added the LOAD statement to retrieve the Customer.



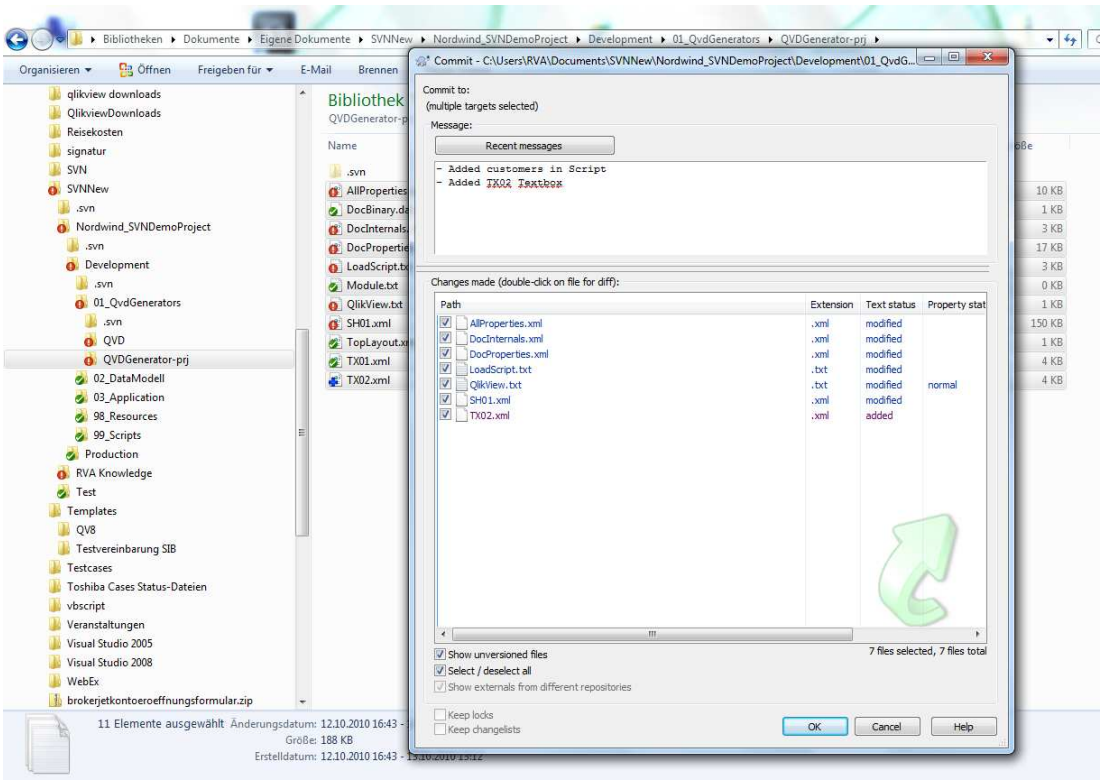
TX02.xml has a blue icon. This file represents the new yellow textbox we added on the “Main” tab. A hint to the new sheet object can also be found in the Qlikview.txt. Between the initial version (left side) and our current version we can see that the TX02 textbox was added.



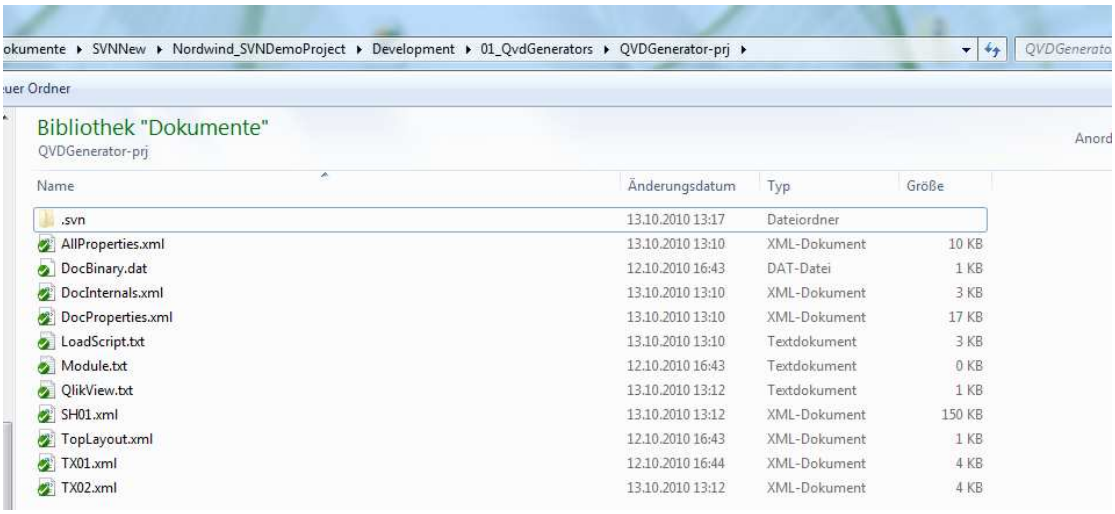
To add the TX02.xml to version control, you must add it to the SVN repository as shown before.



Next step is to commit all changes to the repository. This will commit the changes to already versioned files (red icon), and newly added files (blue icon on TX02.xml). Again, add a meaningful comment.



After the “SVN commit” has succeeded, ensure that all icons are green again. That means all your local changes are committed to the SVN repository.



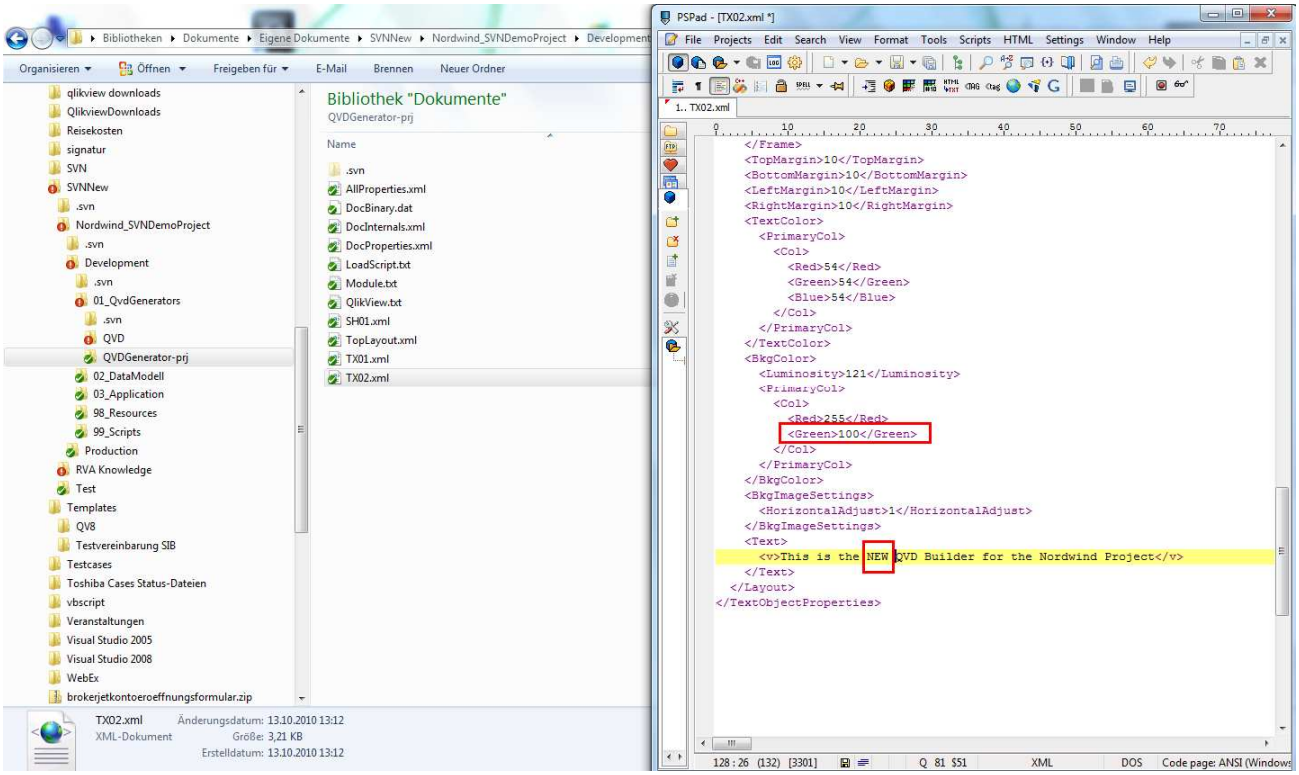
5.4 Changes directly to XML files

This is not a supported way to change your .qvw, but the .xml definitions allow you to change settings with any other tool.

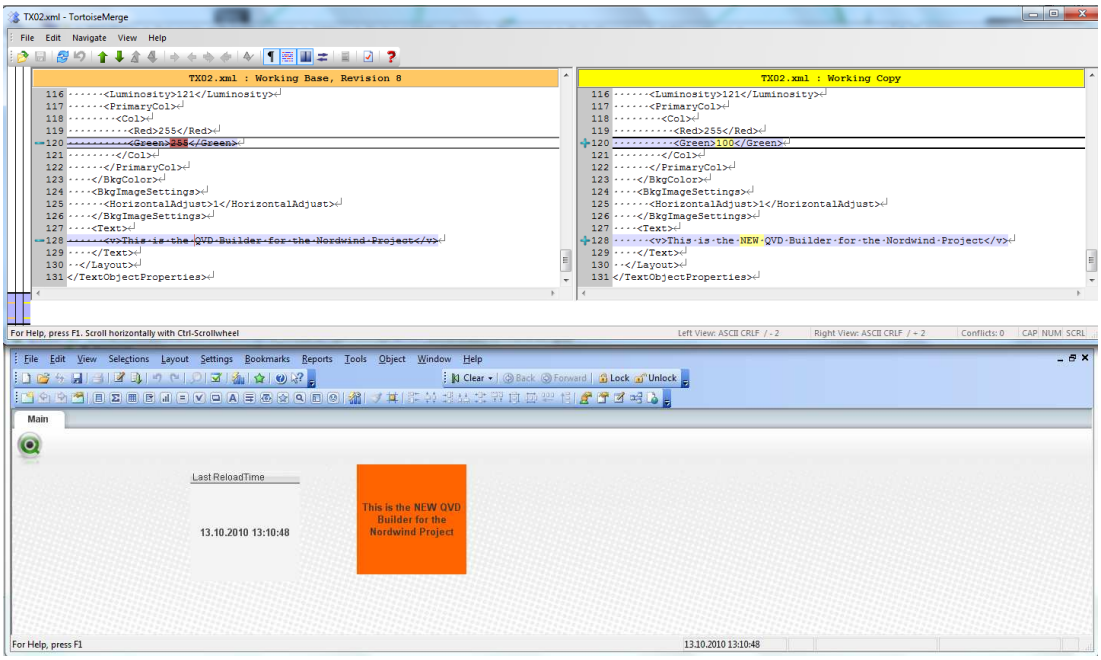
For example I want to change the TX02.xml

- Add the word “NEW” to the text
- Change the background color of the textbox from yellow to orange

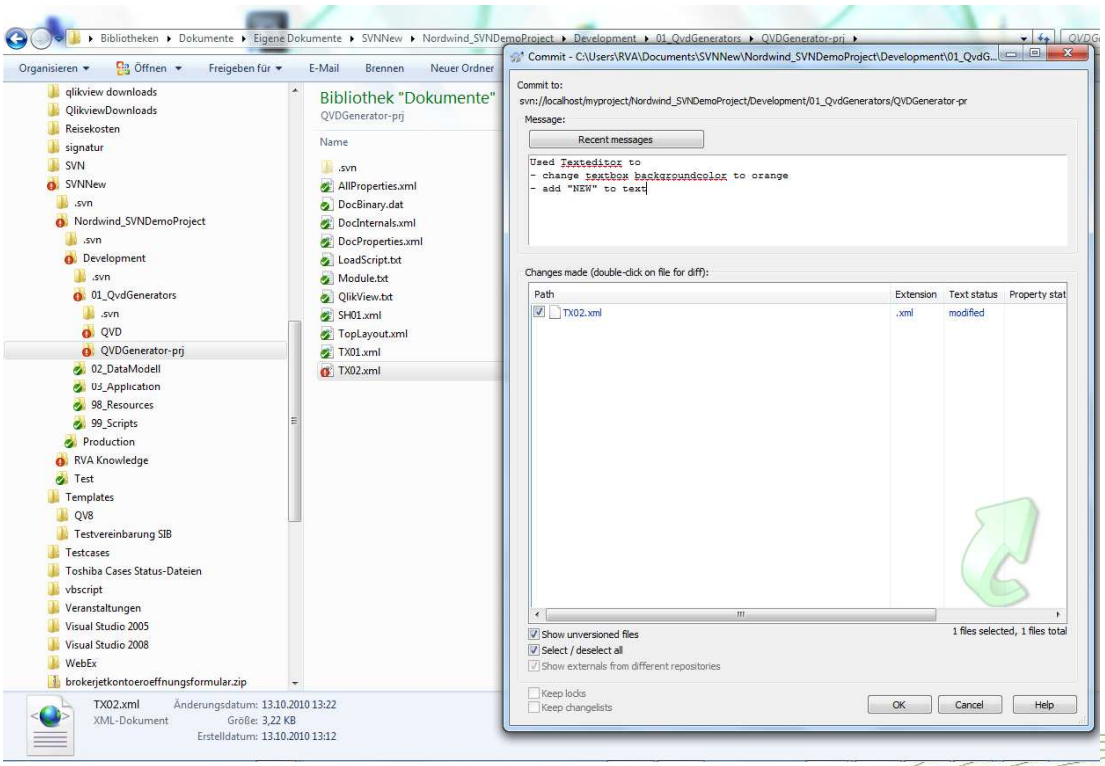
Open the TX02.xml in your favorite text editor. Change the settings according to the screenshot below. Save TX02.xml.



If you now open Qlikview 10, your manual changes to TX02.xml are immediately reflected in Qlikview. See screenshot.



As before, to store these changes in the SVN Repository, commit the new version of TX02.xml. Add a meaningful comment for your manual changes.

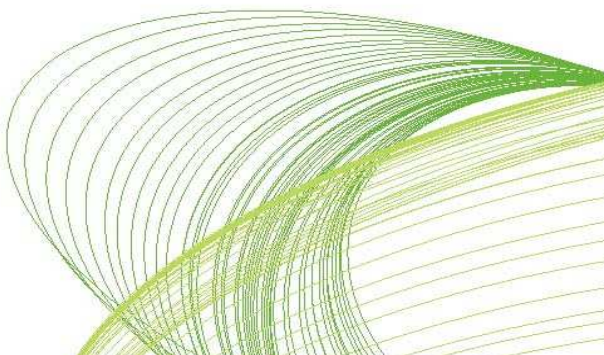


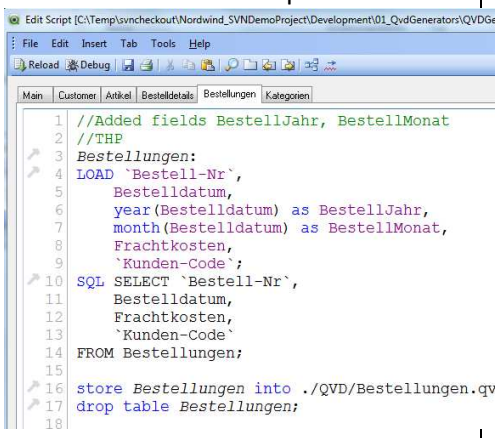
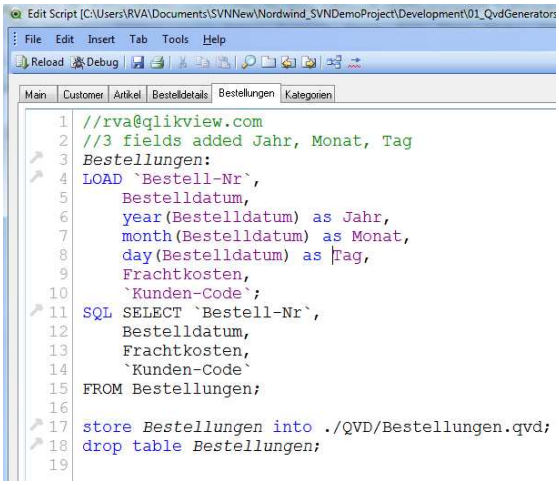
6 Multiuser Development of a Qlikview Application

Until now we have seen the advantages of a version control system when it comes to documentation (comments) and the ability to compare different version.

In this chapter we discuss an example what could happen if multiple developers work on the same .qvw. We will see how a version control system like SVN can help these developers to work together on the same .qvw.

In this example we have two developers THP and RVA. They both have checked out the same version of the .qvw-definition from the SVN repository. Each developer works on his individual copy of these files on his local machine. All changes must be committed to the SVN.



Developer THP	Developer RVA
0) Both Developer check out the same version of QVDGenerator.qvw from the SVN Repository	0) Both Developer check out the same version of QVDGenerator.qvw from the SVN Repository
1) THP adds two fields BestellJahr and BestellMonat to the script. 	2) On his machine RVA adds 3 fields Jahr, Monat and Tag 

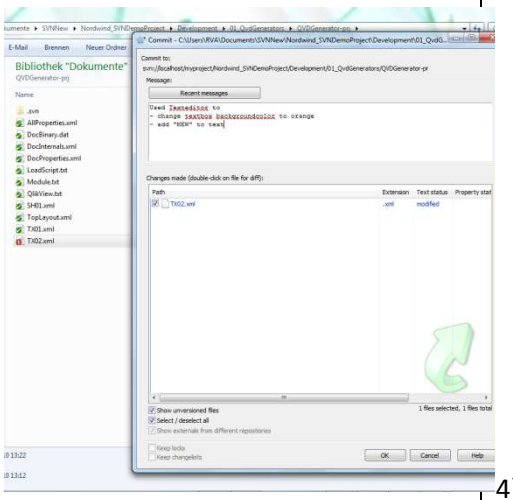
```
//Added fields BestellJahr, BestellMonat
//THP
Bestellungen:
LOAD `Bestell-Nr`,
     Bestelldatum,
     year(Bestelldatum) as BestellJahr,
     month(Bestelldatum) as BestellMonat,
     Frachtkosten,
     `Kunden-Code`;
SQL SELECT `Bestell-Nr`,
     Bestelldatum,
     Frachtkosten,
     `Kunden-Code`
FROM Bestellungen;

store Bestellungen into ./QVD/Bestellungen.qv
drop table Bestellungen;
```

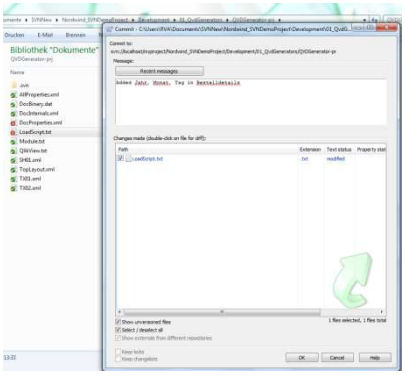
```
//rva@qlikview.com
//3 fields added Jahr, Monat, Tag
Bestellungen:
LOAD `Bestell-Nr`,
     Bestelldatum,
     year(Bestelldatum) as Jahr,
     month(Bestelldatum) as Monat,
     day(Bestelldatum) as Tag,
     Frachtkosten,
     `Kunden-Code`;
SQL SELECT `Bestell-Nr`,
     Bestelldatum,
     Frachtkosten,
     `Kunden-Code`
FROM Bestellungen;

store Bestellungen into ./QVD/Bestellungen.qvd;
drop table Bestellungen;
```

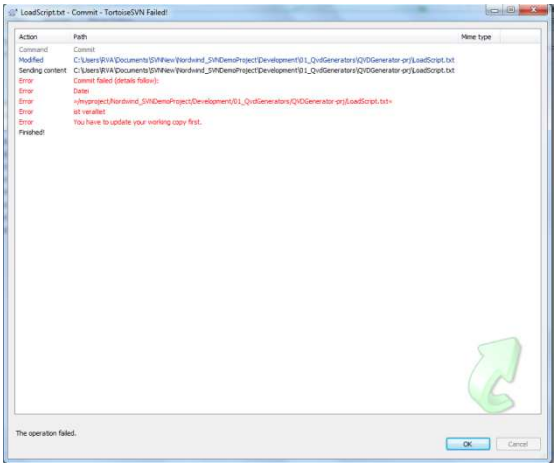
3) THP commits his changes in the file LoadScript.txt to the SVN repository. Now the changes are visible to every other developer.



4)RVA wants to commit his changes in LoadScript.txt to the SVN repository.



5) SVN detects a conflict situation.



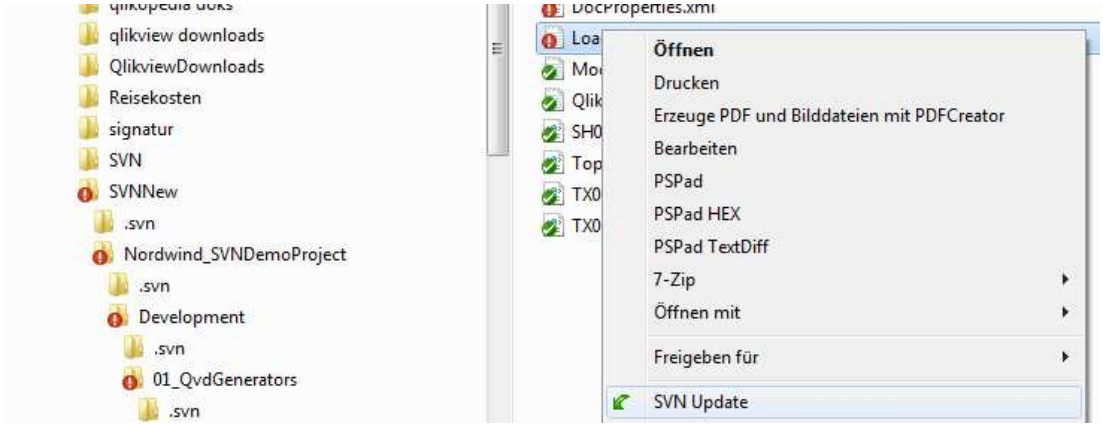
The source control system tells RVA that his working copy is outdated. He first has to retrieve the new version (committed by THP) before he can check in his changes.

This is the typical merge situation when multiple people work on the same project. It's the same with a script-file, a Java file or a C# file. These situations are nothing bad per se and can never be 100% avoided in a multi developer environment. As no software in the world can fix all merge situations automagically, a developer must solve such conflicts manually.

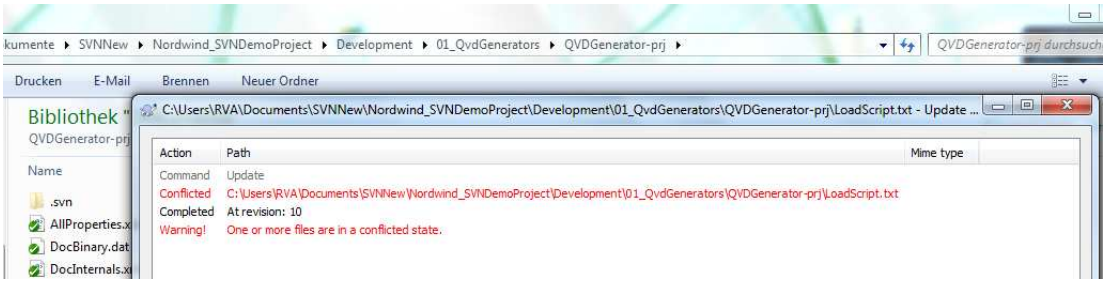
So the next step for RVA is to look which of his changes and THP's changes should merge into the next version of the LoadScript.txt.

6.1 Merging a conflict

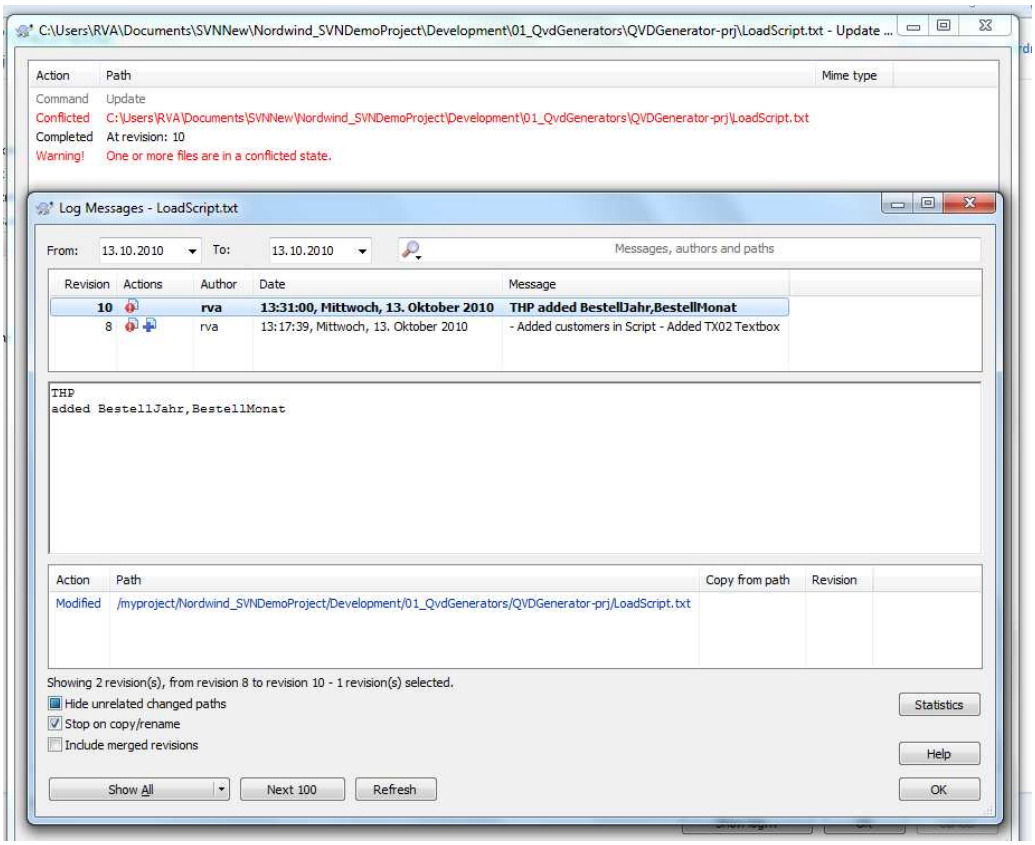
First RVA needs to update LoadScript.txt from the SVN repository to retrieve the version of THP. Right click LoadScript.txt and select "SVN Update".



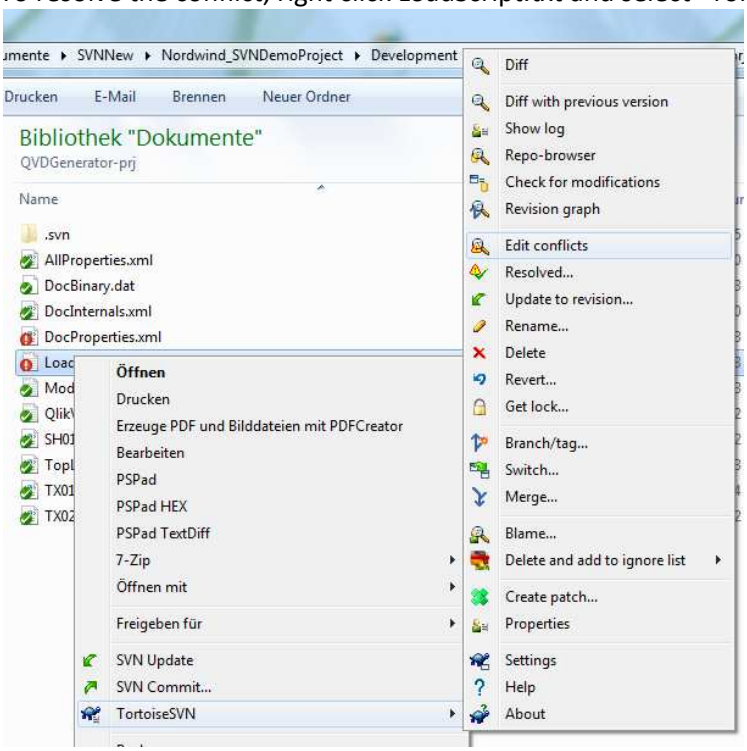
The SVN repository tells RVA "Warning! One or more files are in a conflict state".



First thing RVA should do is to look at the log-entries for this file. He can see the comment “THP added BestellJahr, BestellMonat”.

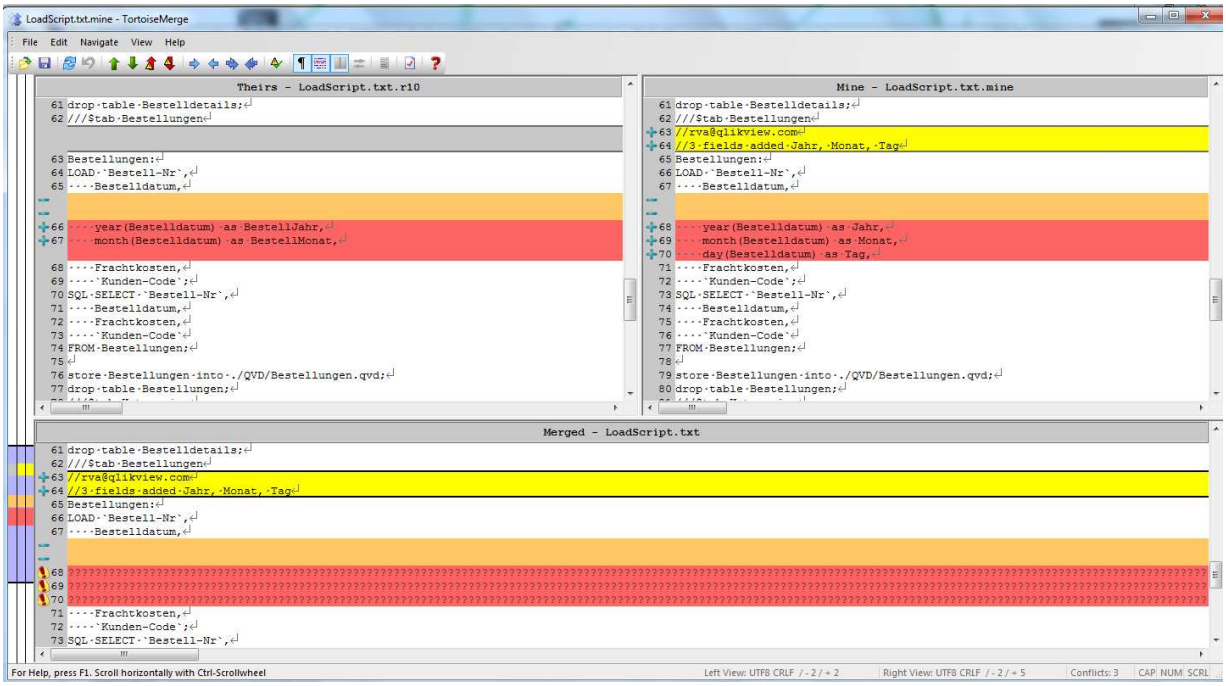


Now it becomes obvious to RVA that he has changed similar source code lines as THP did. To resolve the conflict, right click LoadScript.txt and select “TortoiseSVN|Edit conflicts”.

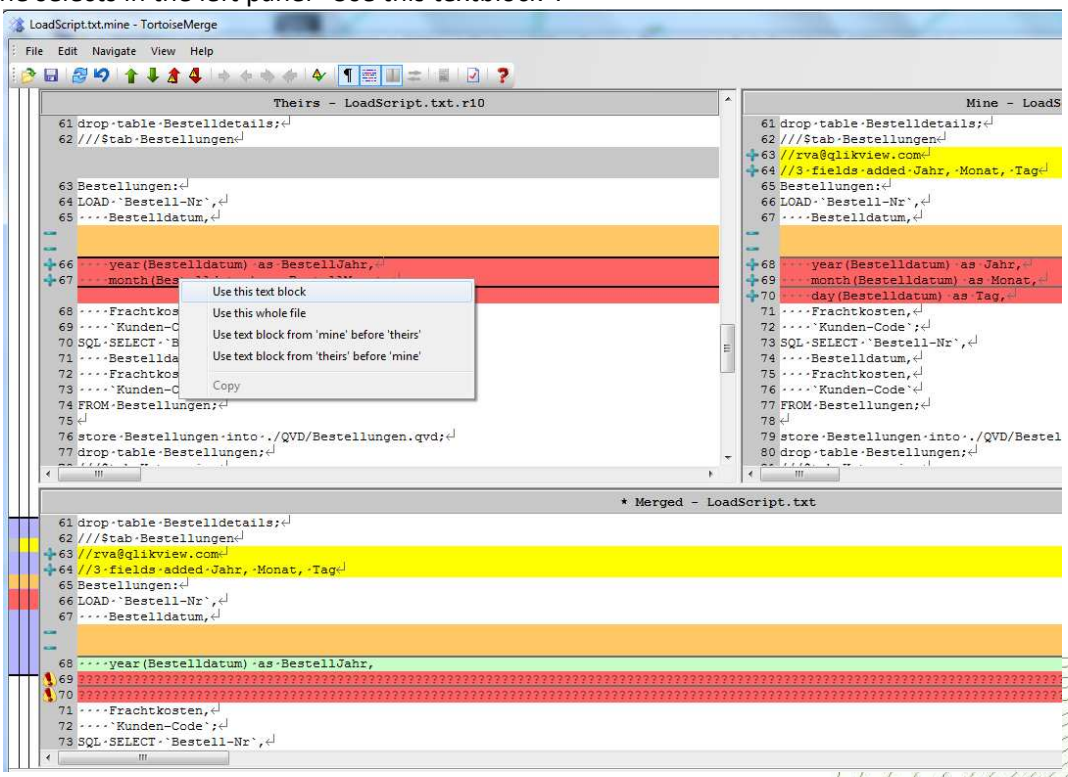


The editor TortoiseMerge opens. On the left hand side we can see revision10 of LoadScript.txt. That's the version of THP. On the right hand side we can see RVA's version. In the lower panel we see the "Merged" version of LoadScript.txt. Red lines show a conflict between the two file versions.

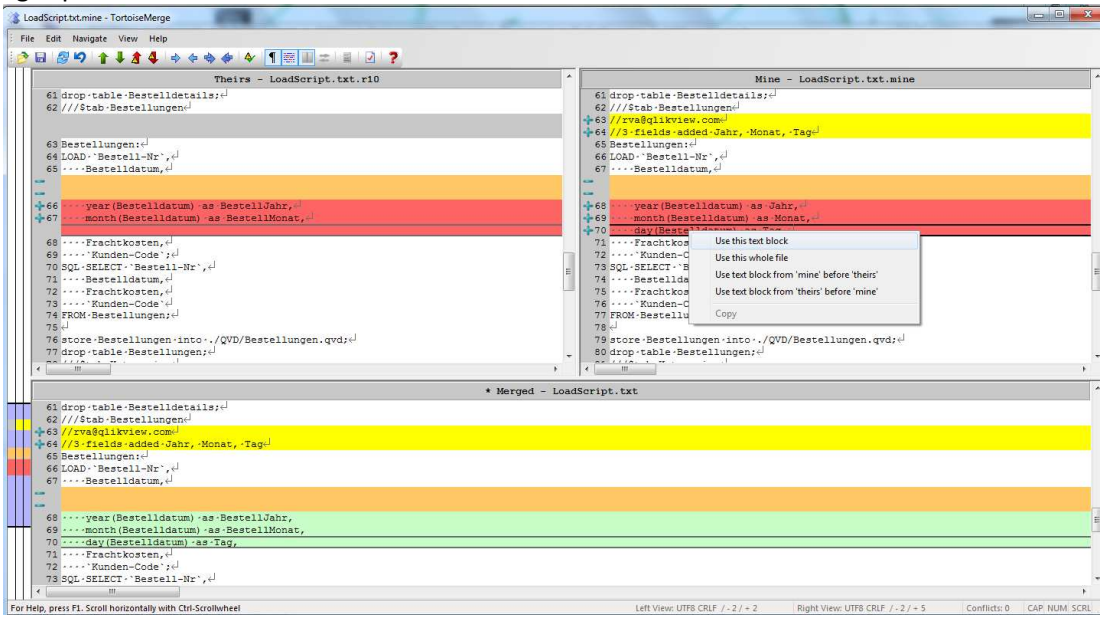
➔ It's now RVA's task to create this new merged version of LoadScript.txt



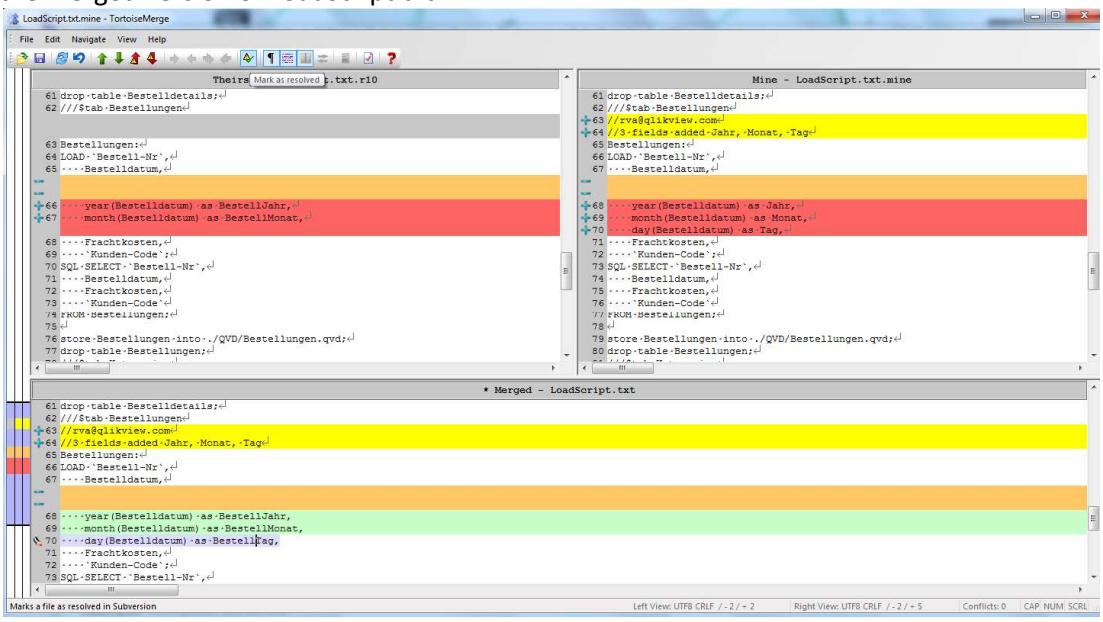
For the fields BestellJahr and BestellMonat RVA decides to use THP's field names. Therefore he selects in the left panel "Use this textblock".



For the field “Tag” RVA uses the line in his version of the file. Therefore he selects in the right panel “Use this textblock”.

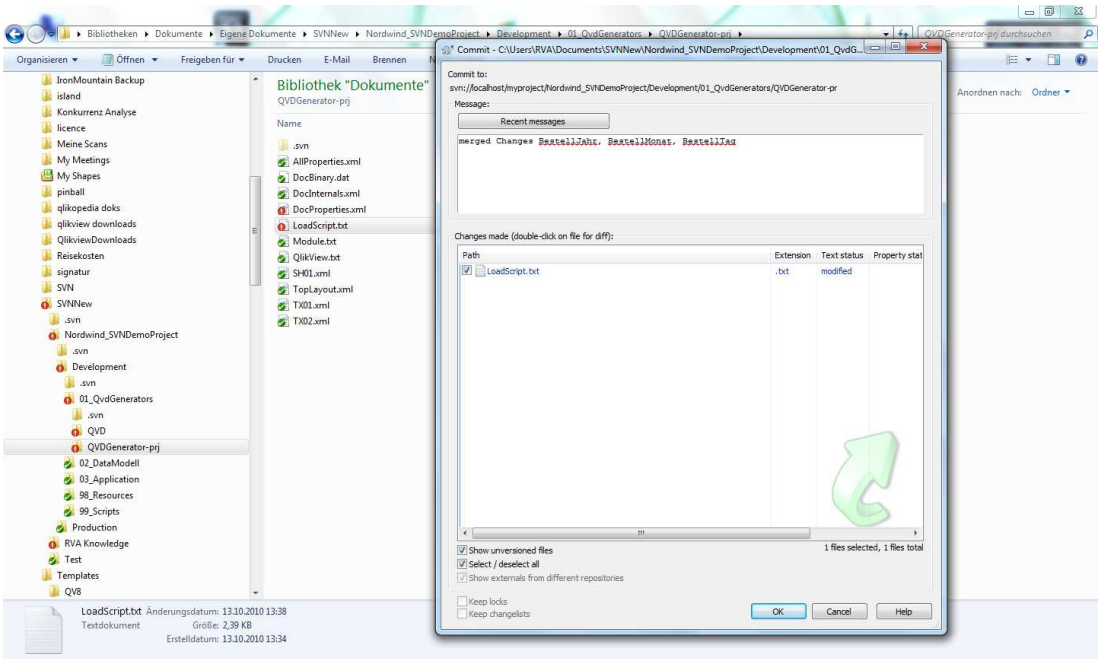


To have coherent field names RVA manually changes the fieldname “Tag” to “BestellTag” in the merged version of LoadScript.txt.



After doing so, RVA marks the conflict as resolved and stores the LoadScript.txt.

RVA can now commit his changes to the SVN repository. He adds a comment that he as merged the two versions into one.



The next time RVA, THP (and all other developers on the project) update their local version against the SVN, they see the new, merged script in the QVDGenerator.qvw.