TestingWhiz

Code Less, Test More

Team Collaboration with TestingWhiz

TestingWhiz 6.0

11/20/2017

Contents

What is Collaborative Software Development?

Collaborative Software Development is an approach allowing multiple team members from analysis, development and quality assurance to share work, ideas, and tasks for achieving common goals. The method specifically aims to increase the success of teams as they engage in collaborative problem solving, thereby goals of the project.

How Does Collaboration Work in Automated Software Testing?

In the context of automated software testing, collaboration can occur between developers and testers in terms of creating test plans, test scenarios, test scripts, and test data jointly along with analyzing test results.

Need for Version Control in Collaborative Approach

Collaborative approach is not limited to sharing a common workspace, environment or a shared files/folder. Rather, it is enabling teams to work on a common file yet impart absolute freedom to make changes at any time with Version Control System.

In traditional set-up, team members verbally inform about working on a file so that other team members do not work on it to avoid duplication. This process is extremely prone to errors, as someone will overwrite someone else's changes, sooner or later.

With a version control system in place, everybody in the team can work absolutely freely - on any file at any time. The version control system will later allow you to merge all the changes into a common version. There's no question where the latest version of a file or the whole project is. It's in a common, central place, i.e. your version control system.

Benefits of Using Version Control System in Collaborative Approach

Following Standard Practice – To be a Good Team Player

Version control system allows independent working in a team or on your own. It acknowledges that there is only one project. Therefore, there's only one version on your disk that you're currently working on. Everything else -- all the past versions and variants are neatly packed up inside the VCS. When you need it, you can request any version at any time and you'll have a snapshot of the complete project right at hand.

e.g. Test Case 1 for Jira ID 1 committed to a repository as Version 1. Later there was a negative scenario added for the same requirement and you label it as Version 2.

Restoring Different Versions without Messing Things

Being able to restore older versions of a file (or even the whole project) effectively means one thing: you can't mess up! If the changes you've made lately prove to be redundant, you can simply undo them in a few clicks. Knowing this should make you a lot more relaxed when working on important bits of a project.

e.g. Test case 2 for Jira ID 2 committed to a repository as Version 1. Someone modified the case and made it unstable and committed it as Version 2. You know the issue, revert to Version 1.

Unlimited Tracking of Who did What

Every time you save a new version of your project, your VCS requires you to provide a short description of what was changed. Additionally, (if it's a code / text file), you can see what exactly was changed in the file's content. This helps you understand how your project evolved between versions.

e.g. Test case 3 for Jira id 3 committed by developer1. Test case 3 got modified by developer 2 for covering exception handling reasons. Test case 3 got modified by developer 3 for optimization reasons.

TestingWhiz Features enabling Collaboration

Methods

Any method that performs a single conceptual task should be able to stand on its own as a first-class candidate for reuse and hence labeled as a Method. TestingWhiz allows creating Methods out of Test Steps. These Methods can take parameters, can call other methods, can call itself, can make use of data tables, can reference parameters, can return values, can perform conditional execution and lot more. Once Methods are created, they can be called in a test case using operation named 'Call Method'.

Besides calling Methods from the same Test Project, TestingWhiz also allows calling Methods from other TestingWhiz project file, allowing dynamic linking features. This feature offers unique reusability and maintenance benefits to enterprise teams.

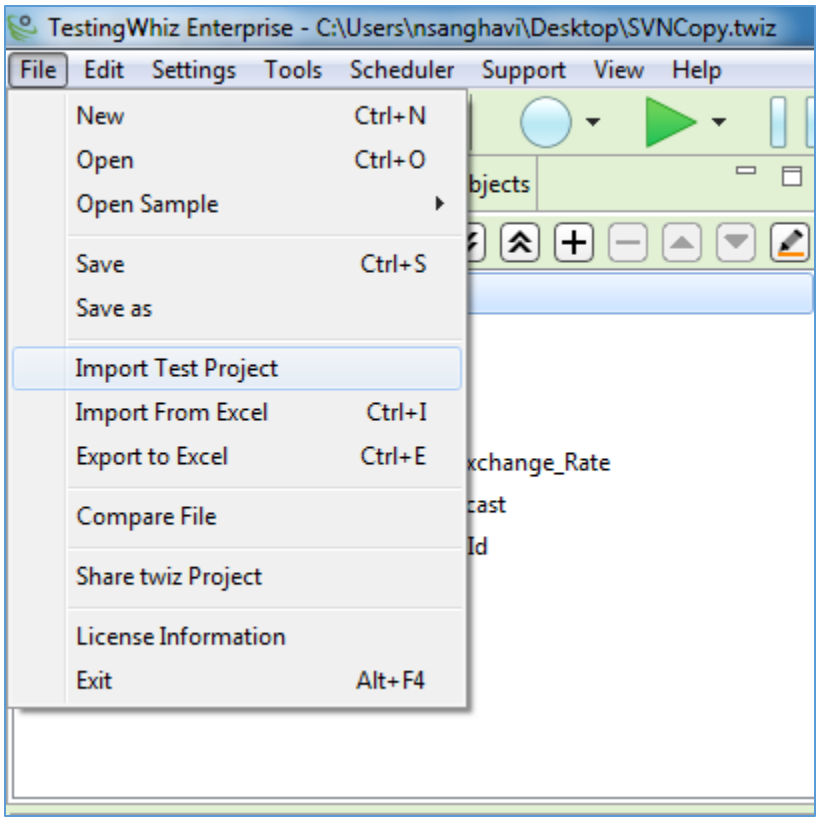
Data Tables

Data tables are interim data storage locations in TestingWhiz simulating an Excel-like interface. These data tables can be populated by querying a database, importing excel files, pasting data from the clipboard or generating data using built-in test data generator. Once data is inside data table, it can be shared across Test Project.

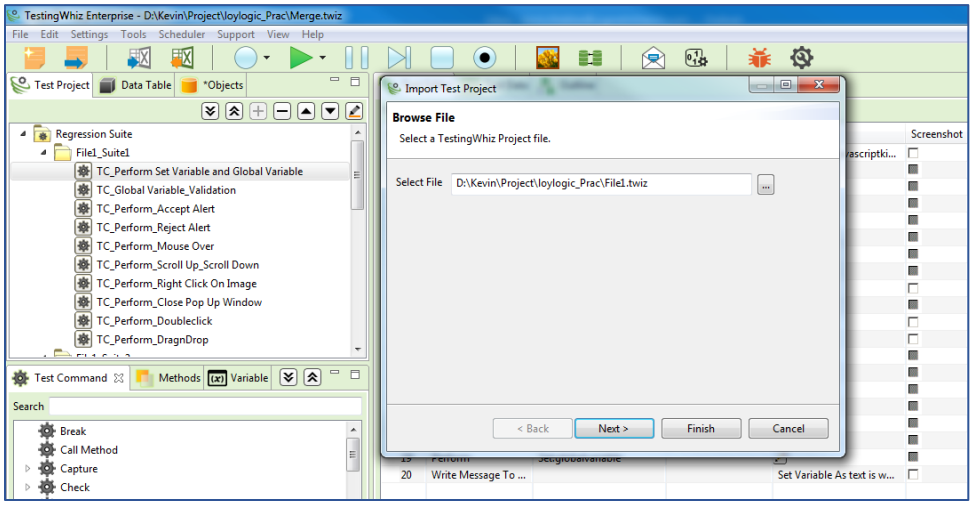
Import Test Wizard

TestingWhiz offers a unique wizard to import Test Cases, Methods and Data Tables. Using this wizard, scripts can be merged and updated easily. Below are the steps demonstrating import wizard capabilities from the tool for merging TestingWhiz projects.

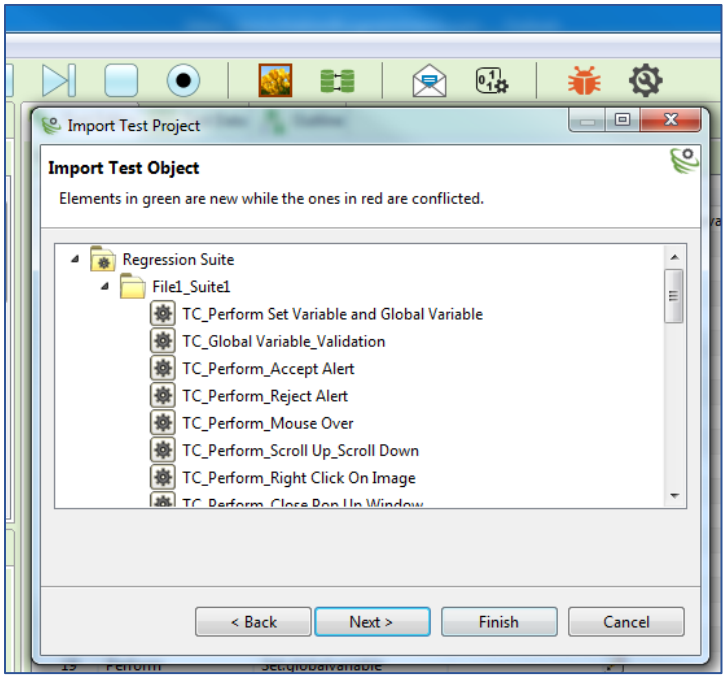
Step 1: Use File > Import Test Project



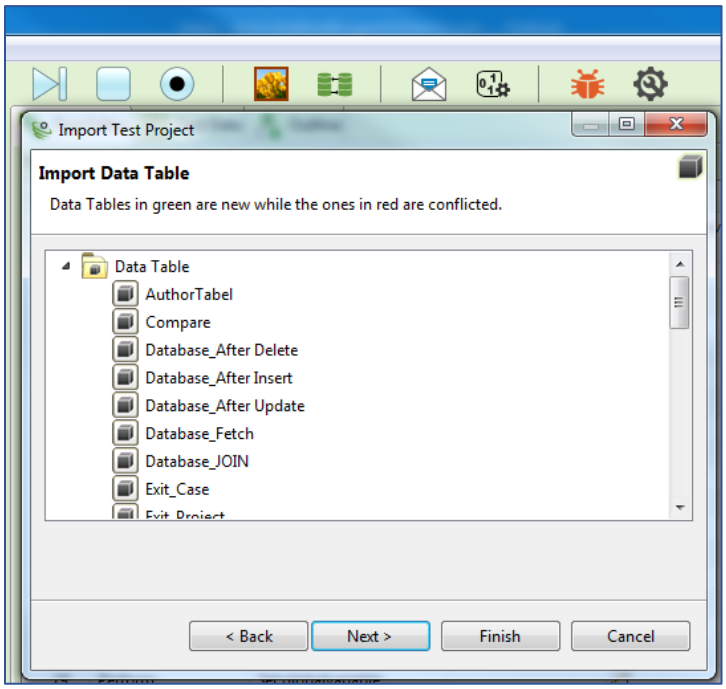
Step 2: Select the TestingWhiz project from where you want to import data



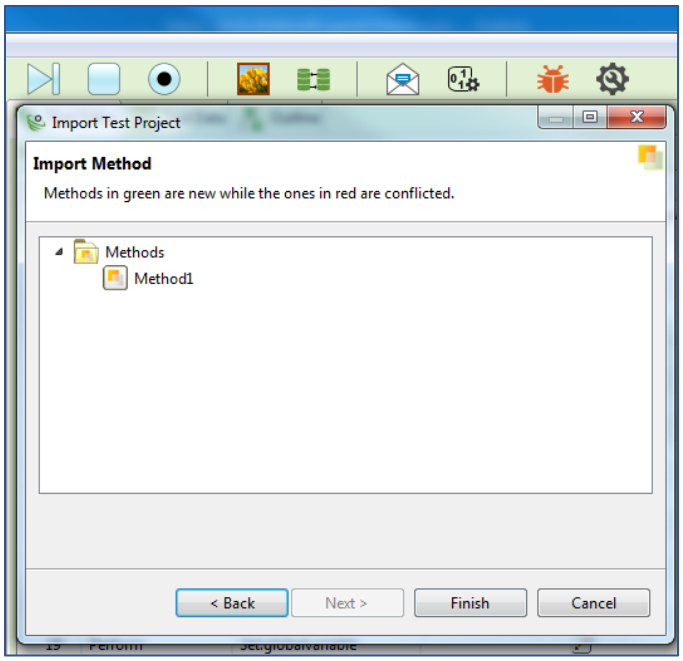
Step 3: Select test cases you want to import



Step 4: Select data tables you want to import



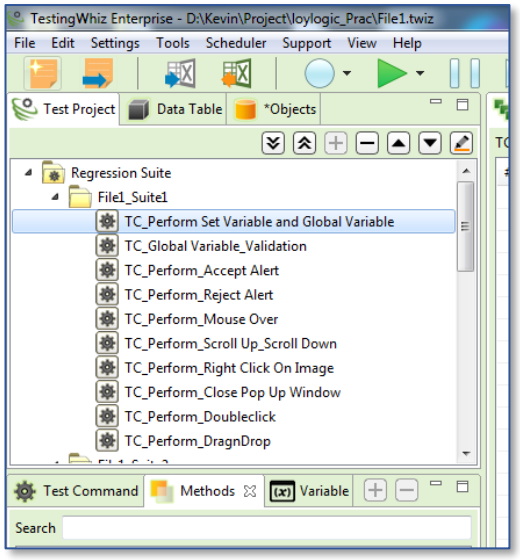
Step 5: Select methods you want to import



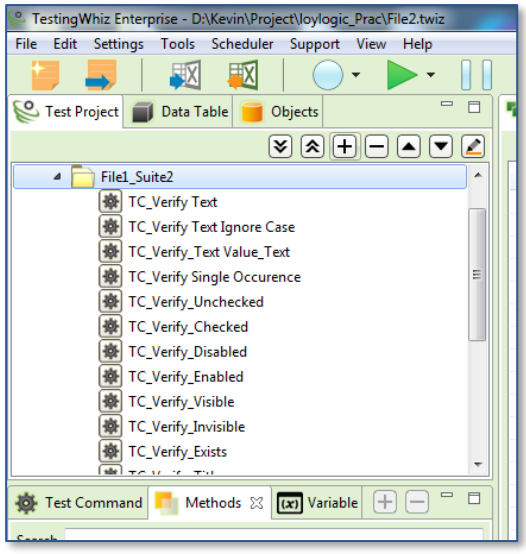
Below are some sample scenarios demonstrating state, pre-merge condition and post merge outcome.

1. Sample Scenarios

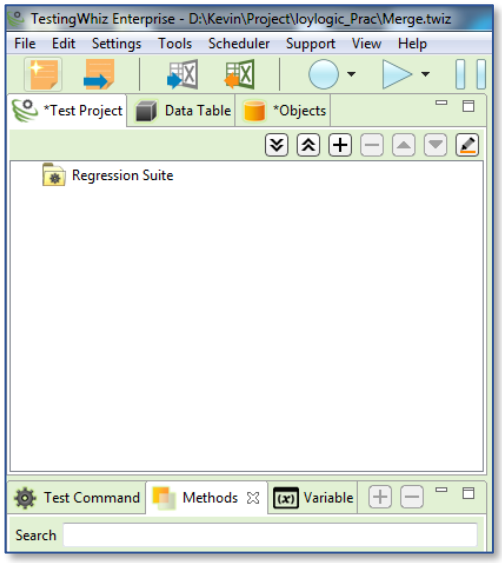
Person 1	Person 2	Action	Expected outcome
File1.Suite 1.TC 1-10	File1.Suite 2 TC 1-10	Merge	Generate a new merged file with Suite 1 – TC 1-10 from file 1 and Suite 2 – TC 1-10 from file 2



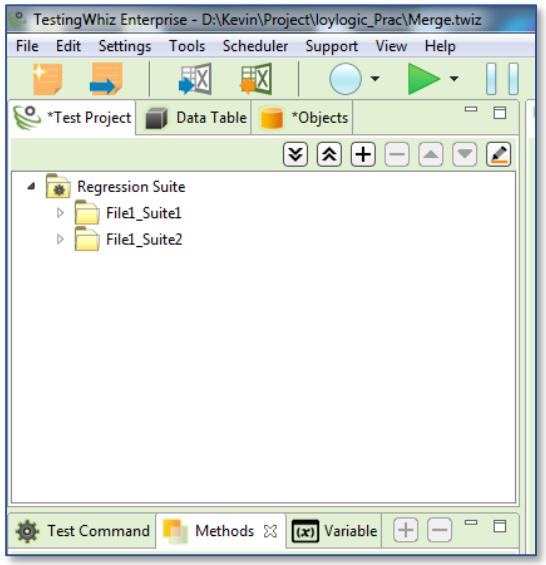
Person1 File_Suit1 having 10 Test Case



Person2 – File_Suit2 having 10 Test Case



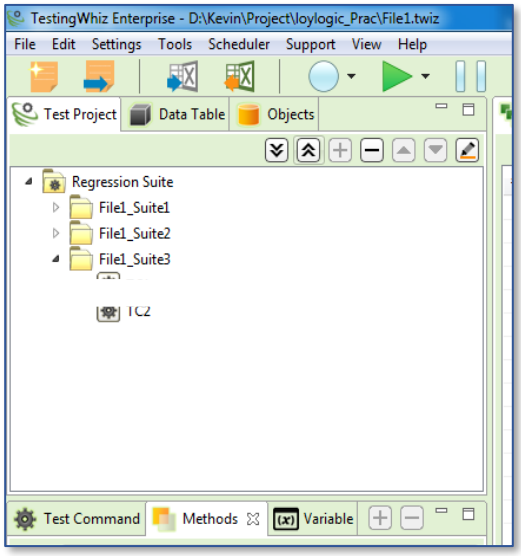
Before Merge



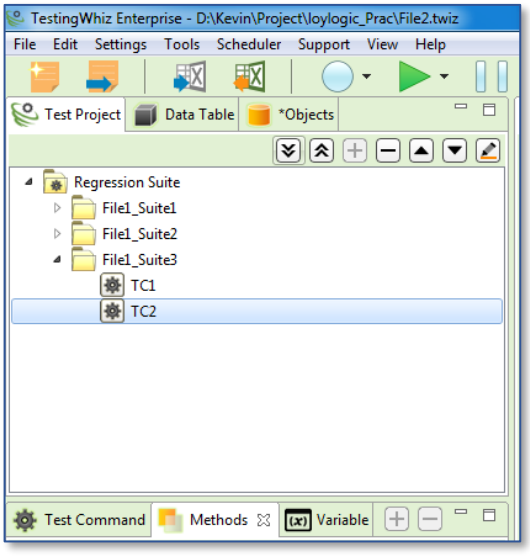
After Merge [Import Test Project] from Person 1 File Suit1 & Person 2 File Suit2

2. Sample Scenarios

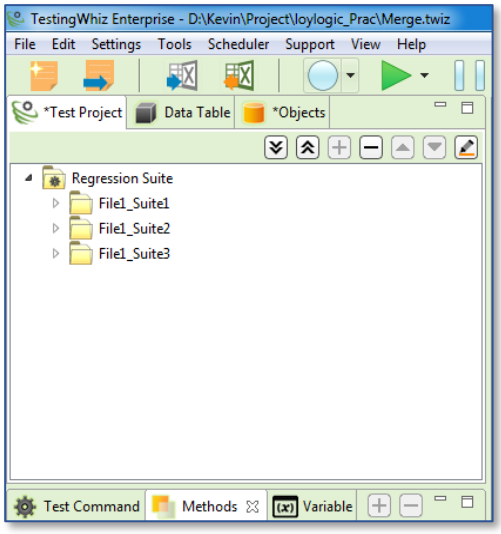
Person 1	Person 2	Action	Expected outcome
File1.Suite 3. TC 1	File2.Suite 3. TC 2	Merge	Generate a new merged file with Suite 1 – TC 1-10, Suite 2 – TC 1-10, Suite 3 – TC 1-2



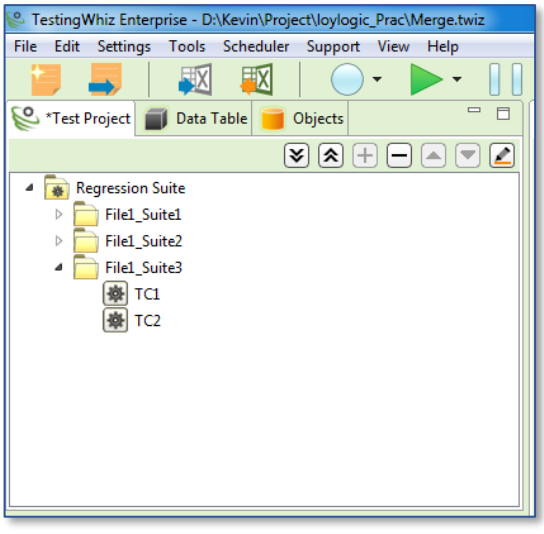
Person1 File_Suit3 TC1



Person2 – File Suit3 TC2



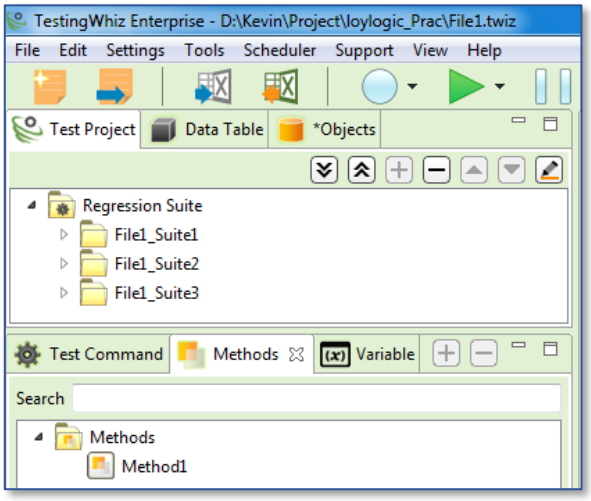
After Import Person1 Suit3_TC1



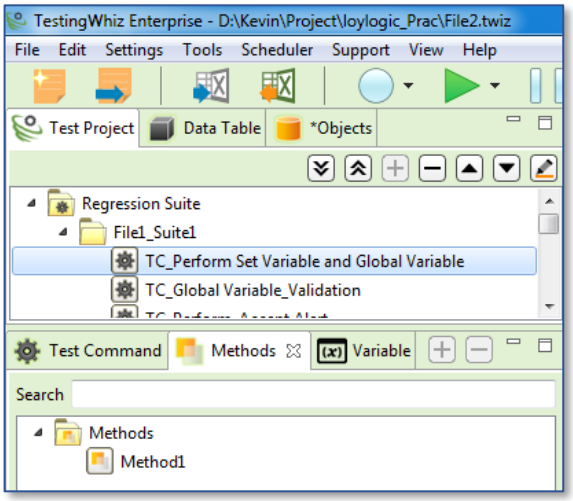
After Import Person2 Suit3_TC2

3. Sample Scenarios

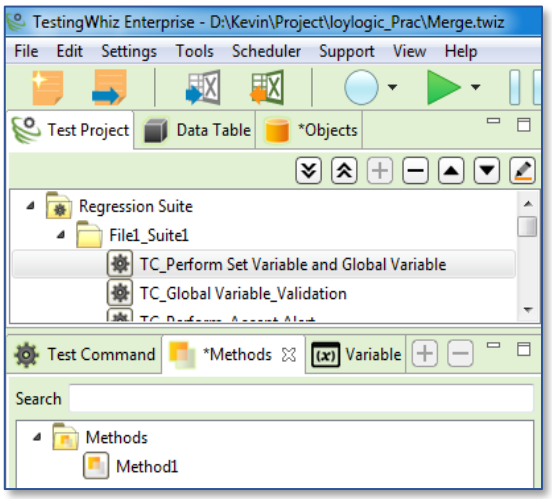
Person 1	Person 2	Action	Expected outcome
File method 1	File method 1	Merge	Keep Person 1's changes



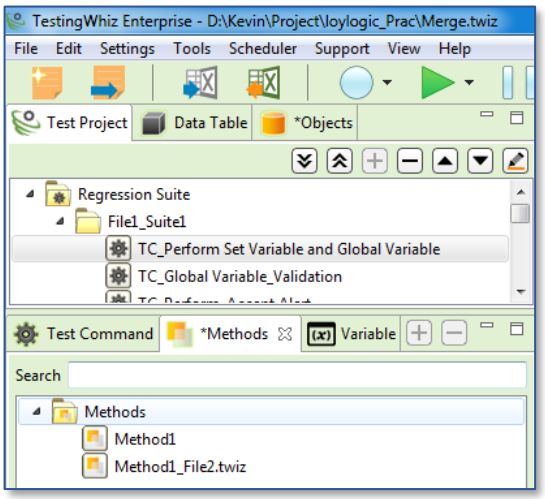
Person1 File Method1



Person2 – File Method1



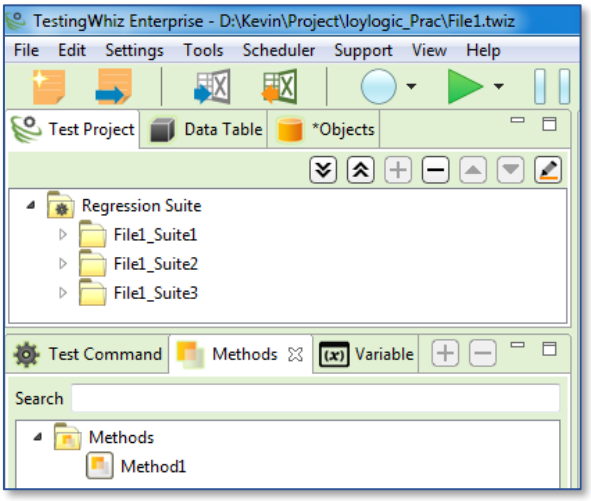
After Import method1 from Person1 File



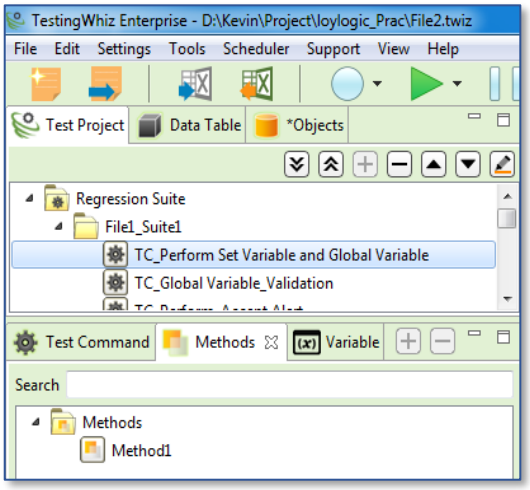
TestingWhiz preserves both the files if they are having same name. User can decide to discard one based on code inside.

4. Sample Scenarios

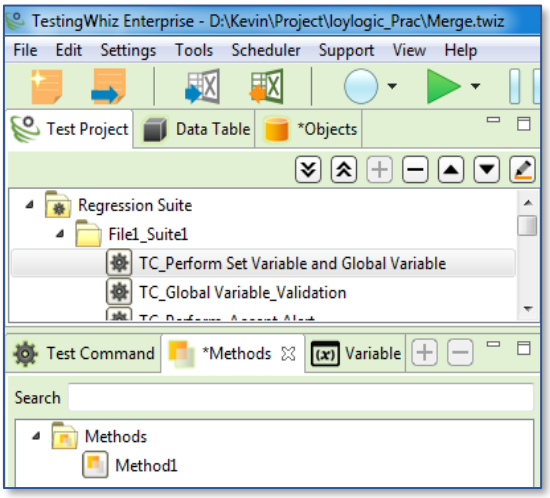
Person 1	Person 2	Action	Expected outcome
File method 1	File method 1	Merge	Keep Person 2's changes



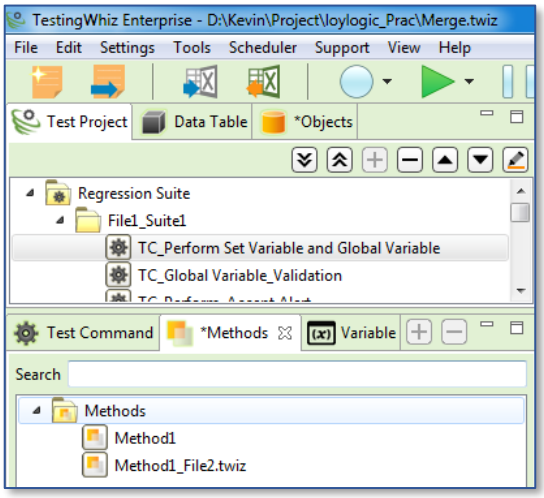
Person1 File Method1



Person2 – File Method1



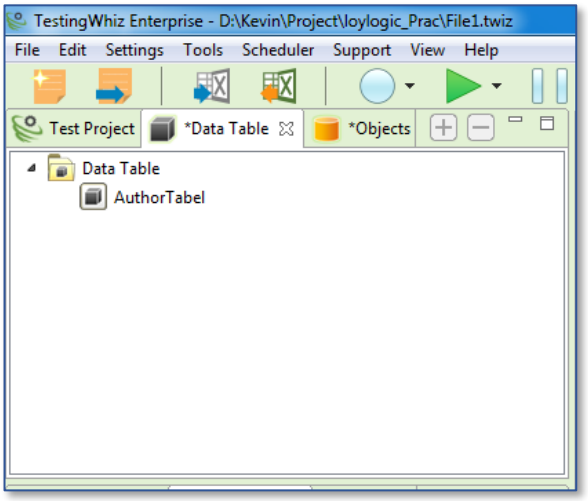
After Import method1 from Person1 File



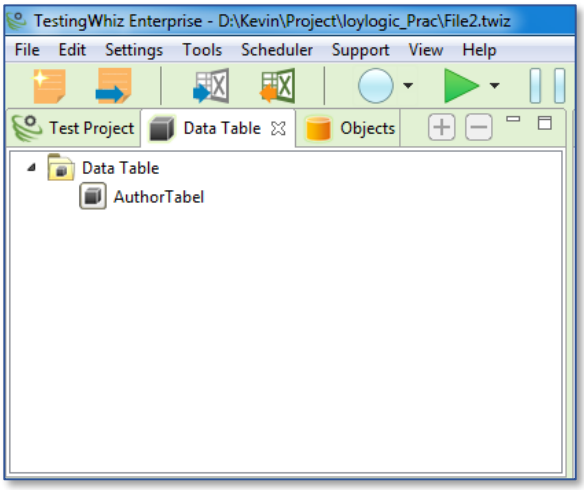
TestingWhiz preserves both the files if they are having same name. User can decide to discard one based on code inside.

5. Sample Scenarios

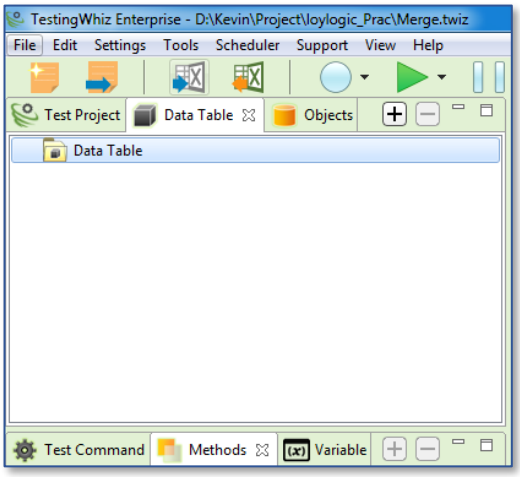
Person 1	Person 2	Action	Expected outcome
File data table 1	File data table 1	Merge	Create a new file with both data tables 1 and 2



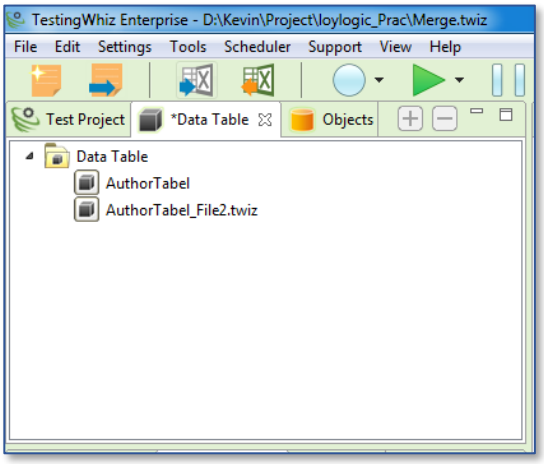
Person1 DataTable



Person2 Data Table



Before Import Data Table

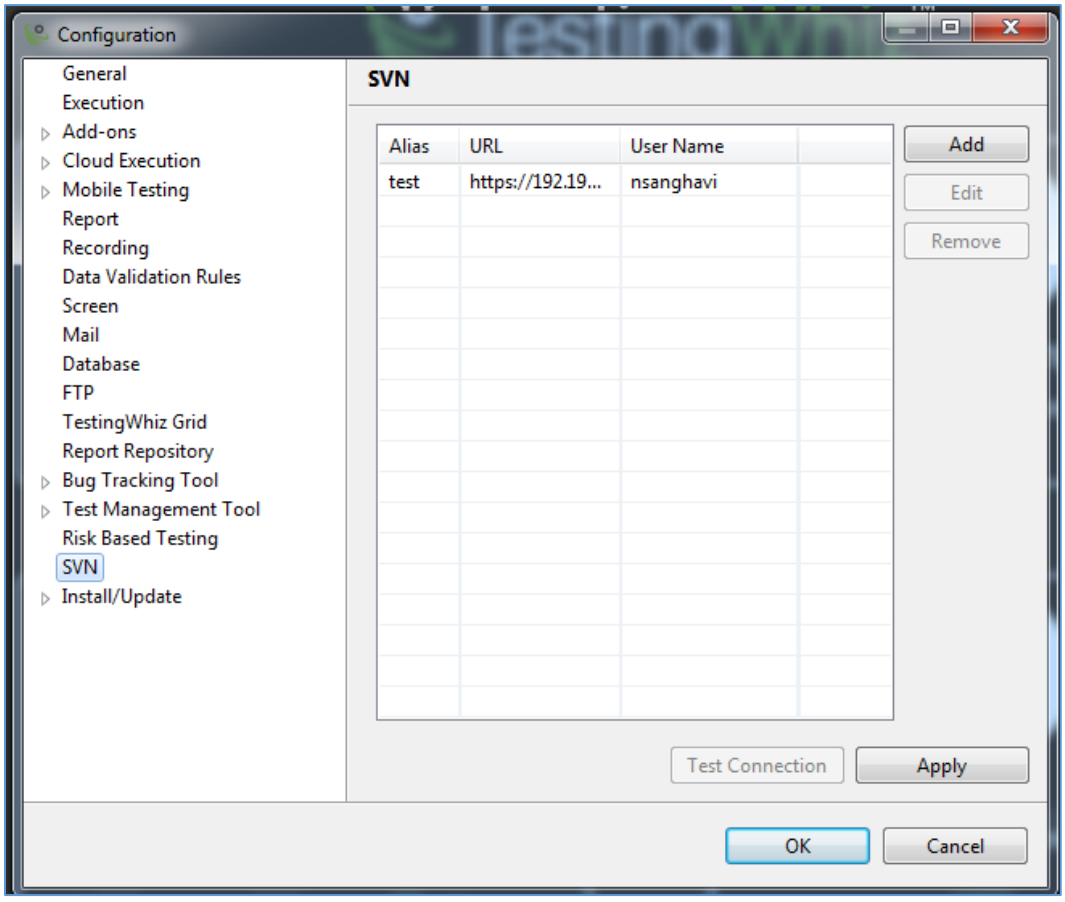


TestingWhiz preserves both the files if they are having same name. User can decide to discard one based on code inside.

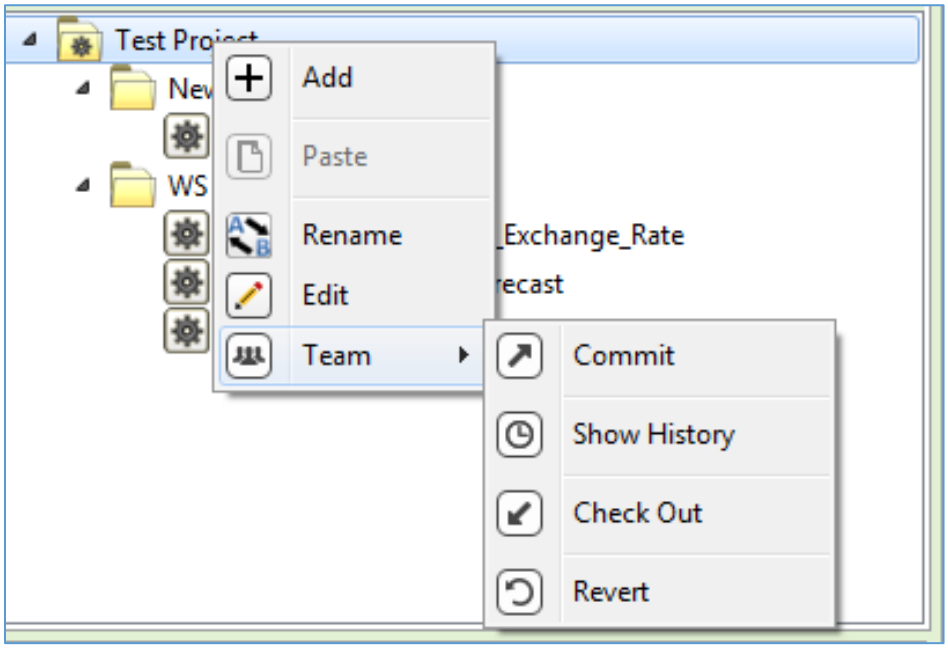
Final Step

TestingWhiz allows integration with versioning tools like SVN. The merged scripts can be checked in to your SVN repository using TestingWhiz IDE itself. Below are the steps:

Setup SVN repository:



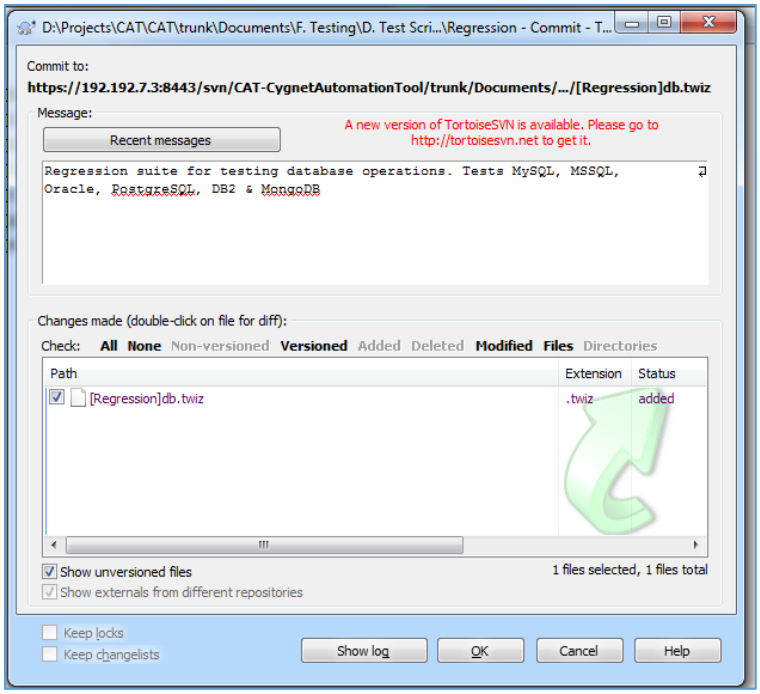
Select the project and perform Check-in, Check-out, Show history or Revert operations.



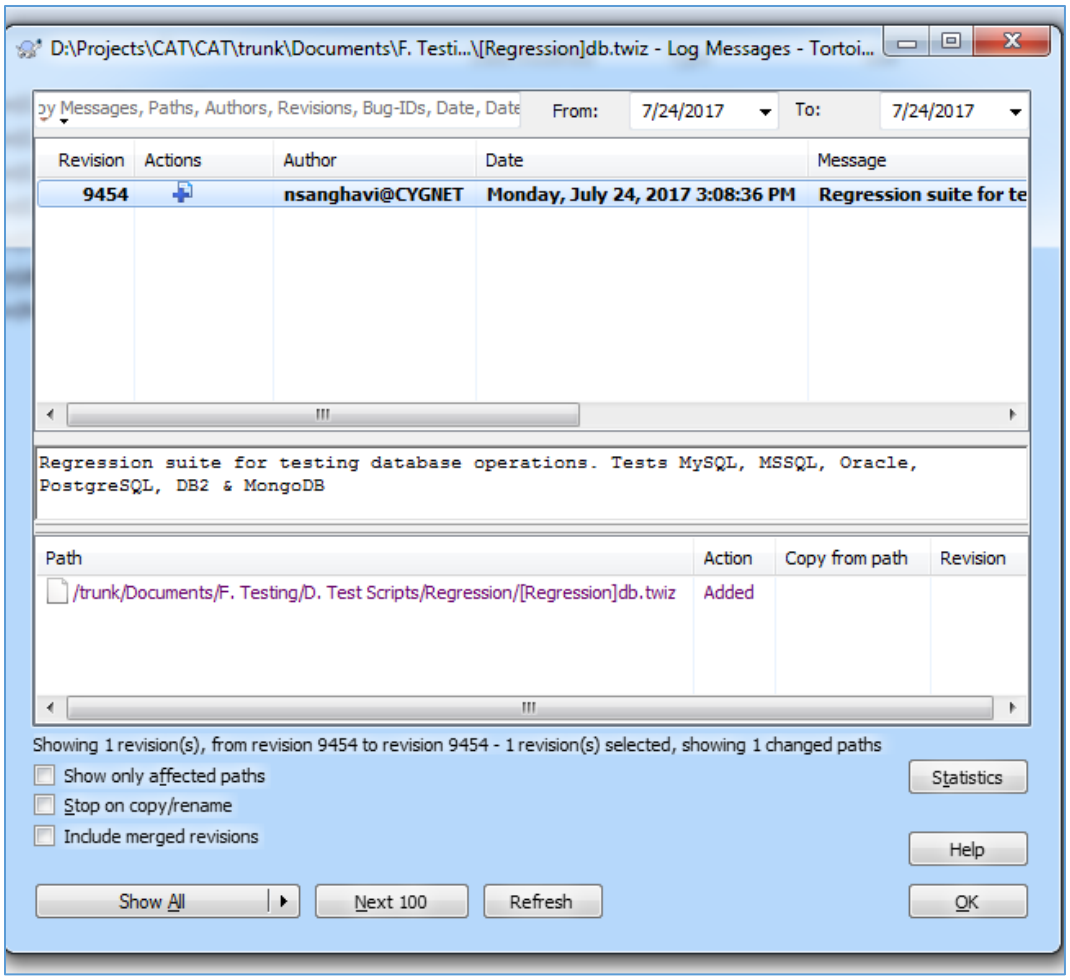
Optionally, once the code is merged, use your existing Version Control System installed on your machine to check-in changes in SVN, GIT or some such and yes, do not forget to put comment as a reason for check-in.

Below are samples for SVN as a repository.

Sample Commit:



Sample Show Log:



End