Toronto, December 2014

Friedrich Brunzema

# TddHelper Visual Studio Extension – Getting Started Guide.

## Introduction

The TddHelper extension is designed to simplify your life if you do Test-Driven Development. It is a helper utility that exposes commands to move to the next tab group, split windows, and jump to test or implementation files (C# only). If a test or implementation file is not found, the utility can create both test/implementation classes and projects. For created projects, automatic references to unit testing framework is supported, and friend assemblies (also strong-named ones) are supported.

The extension tries to support both jumping from Test -> Implementation and vice-versa. This also applies to class and project creation – you can create the test project first and test class, strong-name the test project and add your unit-test framework assembly and then create the implementation assembly just by jumping to the non-existing implementation. When creating test projects, a reference to the test framework is added automatically if so configured.

## Compatibility

Tested on Visual Studio 2013 only for now. Have not tested with Visual Studio 2015. Probably won’t work with VS 2010, VS2012 (not tested).

## Installation

There are three options to install the extension

1. (Preferred) Installation is through the Visual Studio Gallery. Search for TddHelper.
2. Use GitHub .vsix installer. The GitHub project has an Installer folder containing the TddHelper.vsix file. Installing that file may get you a version that is work in progress.
3. Get the source code (use a release tag), build it and then run the output TddHelper.vsix file. You need the Visual Studio Extension SDK (free) and Visual Studio 2013 to build the project.

## Source code

Source code for the Extension lives on <https://github.com/brunzefb/TddHelper>. Feel free to fork. You can also send me a pull request through github if you have bug fixes/enhancements.

## Support/Feedback

If you find a bug, please add it to the issue tracker on github. <https://github.com/brunzefb/TddHelper>. You also can email me at [brunzefb@gmail.com](mailto:brunzefb@gmail.com), please put TddHelper in the subject line. **I will only respond if an issue has been created.** The extension generates comprehensive logs info stored in C:\ProgramData\TddHelper\TddHelper.log, so please attach the log file to the issue along with what you were doing at the time.

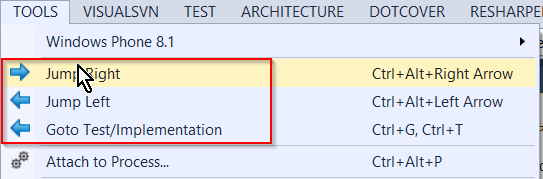
## Version History

Version 1.0 - November 2013, Proof of concept, Jumping Tabs

Version 1.1 - December 2014, first public release. Adds functionality to locate/create tests/projects

## How do I know the install worked?

The Extension adds three new menu items to the Visual Studio Tools menu:



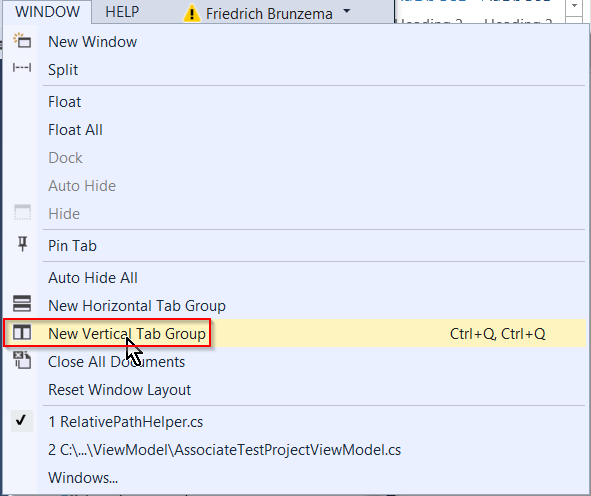
The menu items are a bit lame – they work but it is way better to use the keyboard shortcuts.

**Warning!** The Intel graphics chipset utility maps the Ctrl+Alt+Right Arrow to rotate the screen. That functionality can be disabled in the Intel utility. Ctrl+Alt+Up Arrow restores the orientation if you get stuck. Check the Keyboard mapping section at the end of this document for more information.

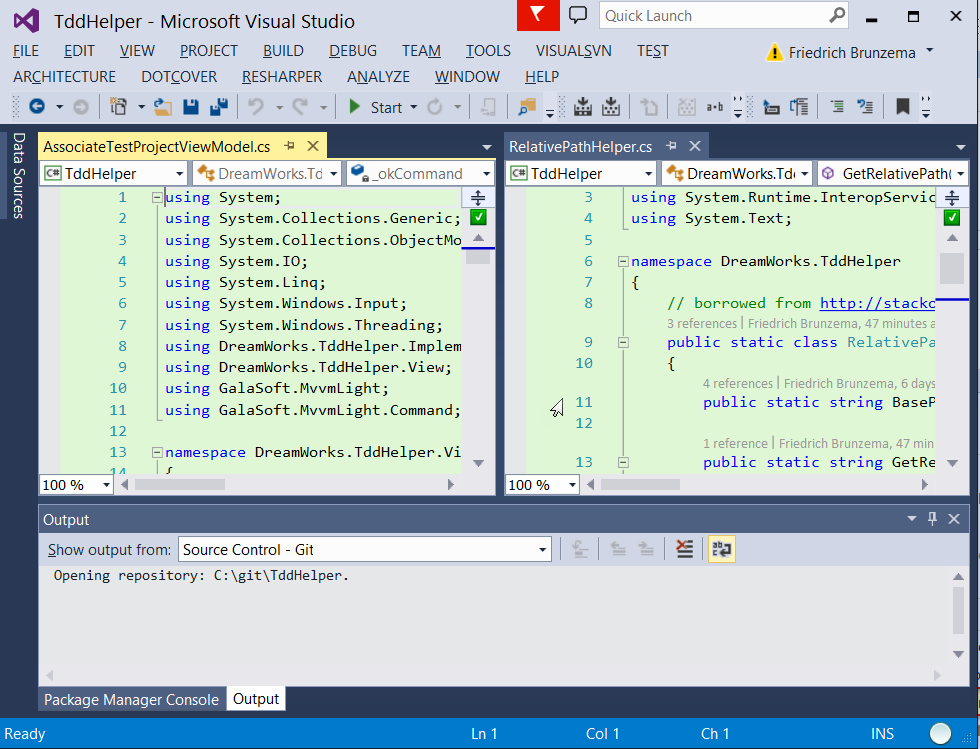
# Jump Left and Right between Tab Wells Using the Keyboard

Unfortunately, Visual Studio does not expose keyboard shortcuts to move between two tab wells. This extension solves this problem. To try it do the following:

1. Load a solution and open two source files
2. Use Window->New vertical Tab Group



1. The resulting screen should look similar to this:

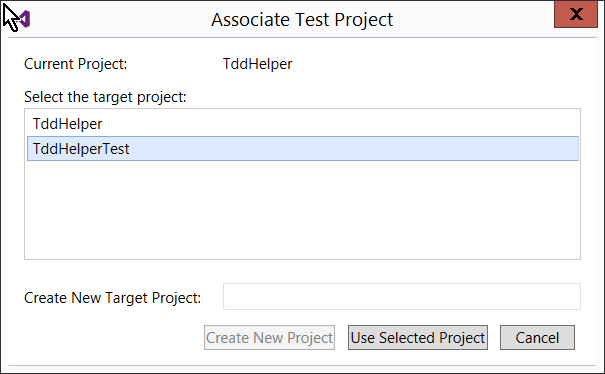


1. Use Tools->Jump Right or Ctrl+Alt+Right Arrow to move the focus between tab wells. Ctrl+Tab will select a different tab in the currently active tab well.

**Tip:** You can assign the command Window.MovetoNextTabGroup and Window.MoveToPreviousTabGroup to move the highlighted tab to the next/previous tab group. My keyboard mapping for this is Ctrl + Shift + Alt + =.

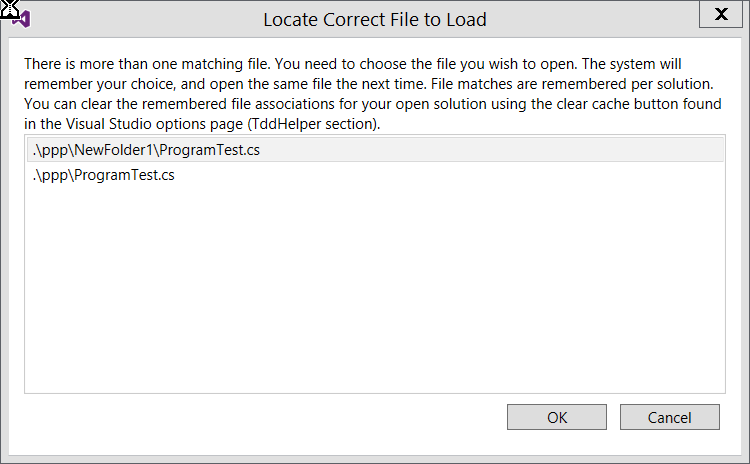
## Goto Test / Implementation

1. Make sure a **.cs file has the active focus** before invoking the command, otherwise nothing happens.
2. If the file ends with Test.cs, the system loads and places the implementation file if a project has been associated.



1. If you press “Use Selected Projected”, a test/implementation is created in the selected project.
2. If you give a name to the Target Project and press “Create New Project”, the following happens:
   1. A new class library project is created [SolutionDir]\NewProjectName\NewProjectName.cs
   2. A folder hierarchy corresponding to the source solution folder hierarchy is created in the target project.
   3. A new class file with corresponding name is created, class is named to correspond with file.
   4. A reference to your preferred unit testing framework is added (needs to be configured in the options)
   5. A reference to the implementation project is added to created unit test projects, if configured to do so.
   6. A friendship relationship (allowing testing of internal classes) is created between Test and implementation projects. (AssemblyInfo.cs in the implementation project is enhanced with an InternalsVisibleTo() statement.
   7. If the implementation project had a strong name, the implementation’s key/pair file (.snk) is copied to the test project. The test project is modified to use the new key.

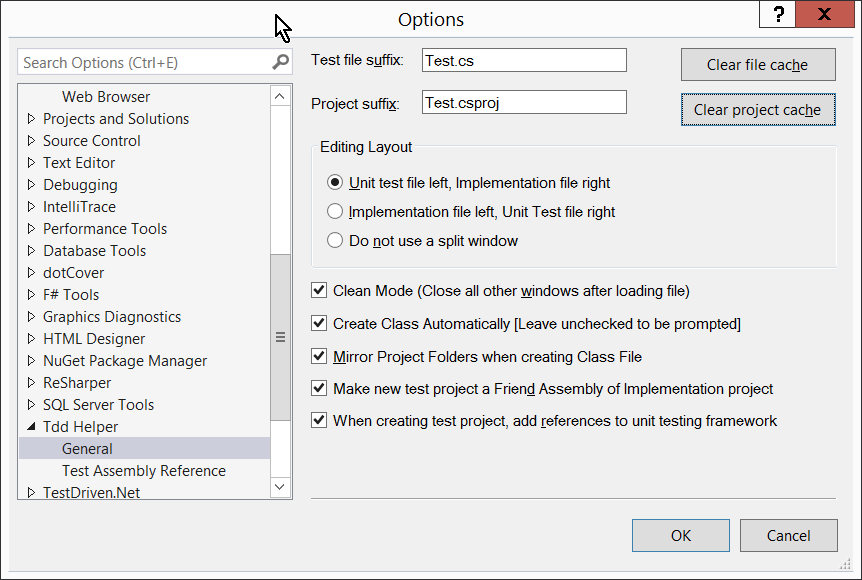
**Tip:** When you have duplicate files a conflict resolution dialog will pop up. This helps you locate the correct file. Once you select the correct file, you choice is remembered – this works on a per solution basis. The dialog is shown below:



# TDD Helper Options Explained

Options can be accessed by going to the Visual Studio Tools->Options and locating the Tdd Helper section:

## General



**Test file suffix:** The Extension works by using a naming convention to map between Unit tests and implementation. If you implementation is File.cs and your test is FileTest.cs, then choose Test.cs for the suffix. If your implementation is File.cs and your test is FileUnitTest.cs, then choose UnitTest.cs for Test file suffix.

**Project suffix:** Not currently used, leave this as is.

**Editing Layout:** Gives options to place your unit tests either left, right (using split window) or just to use a single multi-tabbed window. **Tip:** If you change the layout from left to right or vice versa, and your test/implementation is wrong side, jumping to the implementation/test will switch the windows to the correct layout.

**Clean Mode:** Saves everything, closes all the windows and then just reloads Test and implementation in the correct layout. This is useful as it focuses the view on just the test and the class under test. If this is not selected, Tdd Helper will try to keep the other windows open – it will just close the test and implementation files and re-open them in the preferred location.

**Create Class Automatically:** Tdd Helper maintains an association between Implementation and Test project. The association is created when you try to jump to a non -test or implementation, and is persisted on a solution basis. If Create Class Automatically is checked then missing implementation/test classes are created automatically in the associated project. Unchecking this checkbox will cause the system to prompt you before creating a class.

**Mirror Project Folders when creating class.** If you check this, Test/Implementation classes are created in the same Solution Folder hierarchy. E.g if . .\project\Subfolder1\Subfolder2\impl.cs is active and you jump to test/implementation, a new \projectTest\Subfolder1\Subfolder2\implTest.cs will be created. Left unchecked, a new \projectTest\implTest will be created.

**Make new test project friend of implementation project.** If jump to a non existing test/implementation, the system will prompt you to pick or create a project. If you create a project and have this option selected, then an [Assembly:InternalsVisibleTo(..)] directive will be added to your implementation’s AssemblyInfo.cs file. TddHelper knows about strong-names – so the proper public key is automatically added in a user-friendly way without running Sn.exe –Tp. Having the box unchecked means that TddHelper leaves your AssemblyInfo.cs files alone.

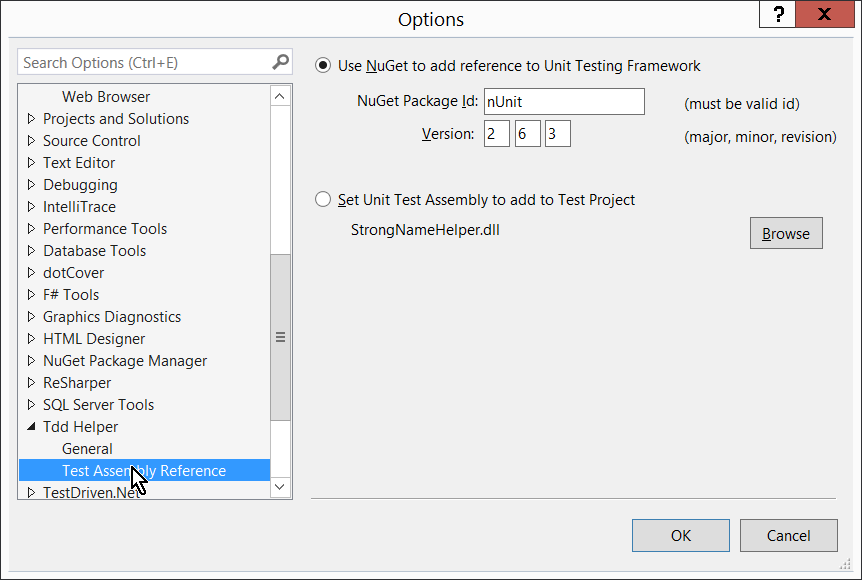
**When creating Test Project, add references to Unit Testing Framework automatically.** Does what it says. The options on how the unit test framework is added (and what framework) are on the “Test Assembly Reference” options page, which is shown below:

**Clear File Cache Button:** The extension maintains a per-solution persisted file association cache. Say you have a file called Program.cs and there are two ProgramTest.cs files in your test project. If you Program.cs is active and you jump to the test file, you will be presented with a dialog to help resolve the conflict, ie to pick one or the other file. Your choice is then persisted and when you jump to the Test file the next time, the remembered choice will be used. If for some reason you picked the wrong file, you can clear the cache with the Clear File Cache Button.

**Clear Project Cache Button:**  In order to create test/implementation files, a project must be specified. The user is presented with a dialog to either pick an existing project or to create a new project where the test/implementation will be created. This choice is persisted on a per-solution basis. If you wish to be prompted again as to where new test/implementation files are stored, hit the clear project cache button.

**Tip:** Internally, absolute paths are used – if you move your project from one location to another on the disk, you may want to clear the caches.

## Test Assembly Reference Options Page



**Use NuGet to add Reference to Unit Testing Framework:** If selected uses the specified package id and version to install (from default package source). Check on NuGet.org to find out the exact Package Id and Version. **WARNING:** If you get this wrong, it will most likely silently fail. Try the Package Manager console window to make sure you get spelling and version right.

**Set Unit Test Assembly to add to Test Project:** Sometimes it is nice not to use the package manager. Just select this option, and then click browse to locate your unit testing framework assembly. A reference to that assembly file is then added, just like if you were to reference it manually.

# Remapping Keyboard shortcuts

Use the Tools->Customize->Keyboard… Dialog to customize the keyboard keystrokes.

|  |  |
| --- | --- |
| **Command ID** | **Default Assignment** |
| Tools.JumpLeft | Ctrl + Alt + Left Arrow |
| Tools.JumpRight | Ctrl + Alt + Right Arrow |
| Tools.GotoToolsImplementation | Ctrl + Shift + F2 |

Are the exposed commands. You may need to disable existing shortcuts to get things to work properly. Check the “Shortcut currently used by” combo and examine the contents carefully.

