Sunday 22 August 2010

# Getting Started with the ASCOM Development System

## System Requirements

You’ll need some software installed on your development workstation to contribute to the ASCOM Initiative. To checkout source code or commit changes you will need a Subversion (SVN) client. We recommend the following:

* TortoiseSVN – command line and integration with Windows Explorer, free (GNU GPL). Available from <http://tortoisesvn.net/downloads>
* VisualSVN Client: *highly recommended* – integration with Visual Studio 2003/2005/2008/2010 (*not express editions*). Free trial and we have a free full license available for ASCOM developers and Microsoft MVPs. Available from: <http://www.visualsvn.com/visualsvn/>

## Accessing the Server

There are actually several servers, but the one you’ll need to know about to get started is our version control system, which runs VisualSVN Server and provides version control based on Subversion. The server is at the following URL:

The correct Internet addresses for all ASCOM servers can be found at: <http://teamserver.tigranetworks.co.uk>

Read-only guest access to the subversion repository is available using the username guest and password guest. To commit changes, you will need a personal user name and password issued by the server [administrator](mailto:Tim@tigranetworks.co.uk?subject=VisualSVN%20server).

Most developers will want to checkout the trunk folder (trunk always contains the most up to date main stream of development).

* Do not attempt to checkout the entire repository starting from the root. If you do this, you’ll actually get dozens of duplicate copies and it’ll take many hours to complete, wasting a lot of time, bandwidth and gigabytes of disk space. You should normally checkout only the /trunk folder.*

Platform trunk: /ASCOM/trunk

There is also a test repository that you can experiment on without fear of messing anything up. It’s at:  
/Test

## Do’s and Don’ts

1. Do not redistribute any of the source code or executables you find here. Any source code you get from this server should be treated as confidential.
2. Most of the source code has a very liberal license, but each developer is at liberty to vary from the standard, so you must respect the license of each individual part of the Platform. Where no license is specified, the *Creative Commons Share-Alike Attribution Required* license is assumed.
3. Try to avoid committing code that doesn’t compile. Breaking the build is considered bad etiquette[[1]](#footnote-1).
4. If you do commit code that breaks the build[[2]](#footnote-2), then our build server will make it clear what caused the breakage and you will be expected to take responsibility and fix the build as quickly as possible.
5. Do commit your changes often. Daily commits would be perfectly acceptable. This helps to keep other developers up to date and safeguards your code against accidental loss.
6. Don’t commit build artefacts[[3]](#footnote-3). As a general rule, you should never commit anything that was created by the compiler or a tool. This is acceptable under a few very specific circumstances, but in general, only commit source code. If you are able to use the VisualSVN client from within Visual Studio, that will automatically commit the right files for you and is our preferred tool – however it does not work with Express versions of Visual Studio.
7. Do keep your working copy up to date using the Update command. We recommend that you use Update at least daily, or at the start of each editing session and immediately prior to any commit. The update command merges edits from other developers into your working copy and helps to ensure we are all working on the same codebase.

Let me pull out one point and reiterate it. This is particularly important when adding new projects to the repository:

***Please be careful to commit only source code***

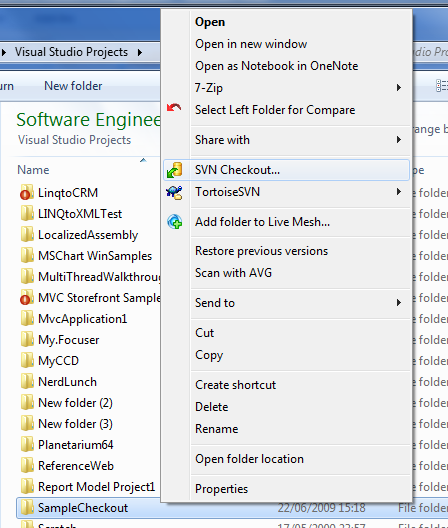
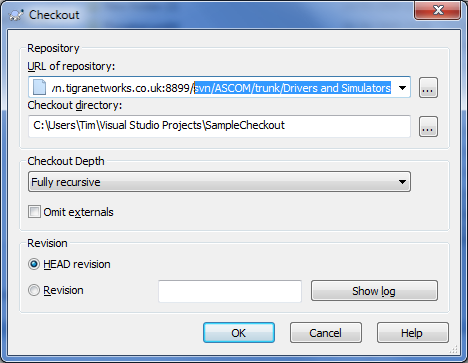
This is important, because once committed, *a file can never be deleted from the Subversion repository* (files can be removed from *revisions*, but the old revisions will always contain the file). So, it is important to be careful what you commit to avoid repository-bloat that can never, ever be cleaned up! Examples of items that should NOT be added to the repository are: executables (.exe and .dll files); anything in obj/ and bin/ folders; user-specific settings from Visual Studio and other tools (.suo, .sccs files); duplicate copies of code – specifically, don’t make backup copies of projects/folders then commit the backups, this duplicates the version history in the repository.

## Workflow

For in-depth documentation on how to use Subversion, please see the [online reference manual](http://svnbook.red-bean.com/) (see page 41 for an overview of a typical work cycle). However, as a lightning fast 30,000 foot overview, here’s the normal developer workflow for ASCOM:



Your first task is to create a working copy of the source code by performing the following steps:

1. Create a folder on your hard drive to hold your working copy. You can call it whatever you like, but “ASCOM Platform” is a good choice.
2. Right-click the folder you just created, look for the TortoiseSVN context menus and select Checkout.  
   
3. Paste in the repository URL that you have been supplied with (normally, this will be <http://svn.tigranetworks.co.uk/svn/ASCOM/trunk/>):  
   
4. Click OK. TortoiseSVN will pull a copy of all the source files into your working copy. This may take some time for a clean checkout. You can now start developing.

1. You can avoid breaking the build by using *pre-tested commits*. There’s a plug-in for Visual Studio that you can install that allows you to submit a *personal build* to the build server. The plug-in gathers up your changes, sends them off to the build server and asks it to build the code with your changes. If (and only if) that build succeeds, your changes will be committed to the subversion repository. This ensures that you never commit code that breaks the build. [↑](#footnote-ref-1)
2. Sometimes, you can successfully build everything locally, then commit your changes, only to discover that you still broke the build. There are many reasons why this can happen – for example, perhaps you referenced a file that only exists on your local hard drive that you forgot to add to the repository. The build server builds under carefully controlled conditions and is designed to catch this type of problem, so it is expected that everyone will, occasionally, break the build. Just don’t do it too often! [↑](#footnote-ref-2)
3. A build artefact is any file that is created by the build process. This includes program executables and DLLs, XML help files, installer packages and so on. [↑](#footnote-ref-3)