

SVN Source Code Repository

Process

**Table of Contents**

[1. Introduction 4](#_Toc393897001)

[2. SVN Background 4](#_Toc393897002)

[3. Important activities of SVN 7](#_Toc393897004)

[4. Actors SVN 7](#_Toc393897005)

[5. SVN folders for Application code base 8](#_Toc393897006)

[6. SVN folders for SQL code base 8](#_Toc393897007)

[7. SVN Development & Release process 9](#_Toc393897008)

[7.1. Code Check-In & Check-Out Process 10](#_Toc393897009)

[7.2. New Branch Creation 11](#_Toc393897010)

[7.3. Build Creation 12](#_Toc393897011)

[ Single release with one Branch folder 12](#_Toc393897012)

[ Simultaneous releases with multiple Branch folders 12](#_Toc393897013)

[7.4. Sync up Branch with Trunk (Single release with one Branch folder) 13](#_Toc393897014)

[7.5. New Tag Creation from Trunk 14](#_Toc393897015)

[7.6. Code Comparison 14](#_Toc393897016)

[7.7. SVN Pull Command 14](#_Toc393897017)

[8. Appendix 15](#_Toc393897018)

## Introduction

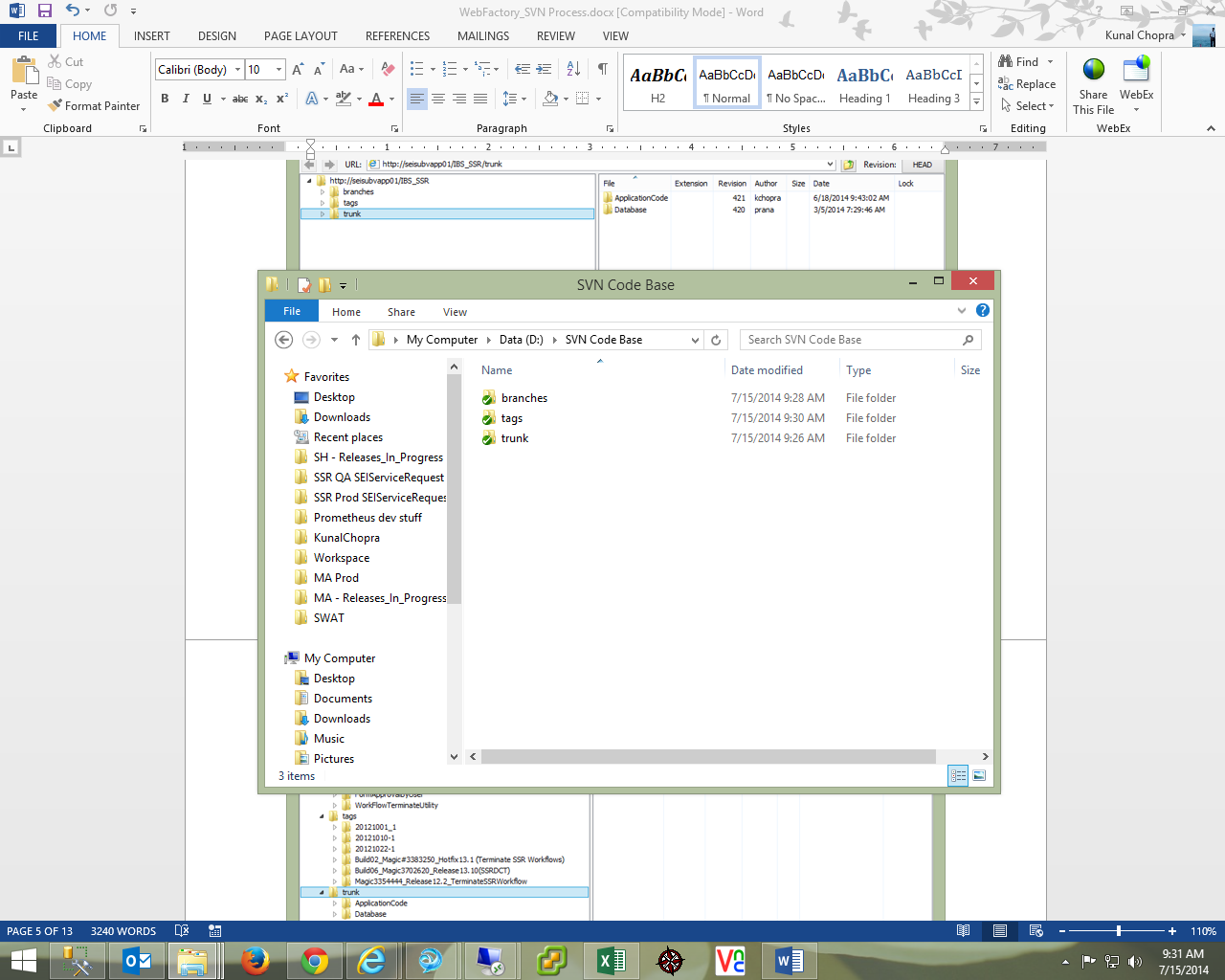
Currently most of the web factory applications maintain their code base in VSS and as per the new SEI guidelines, the code base for both the application & SQL has to be migrated and maintained into the SVN source code repository.

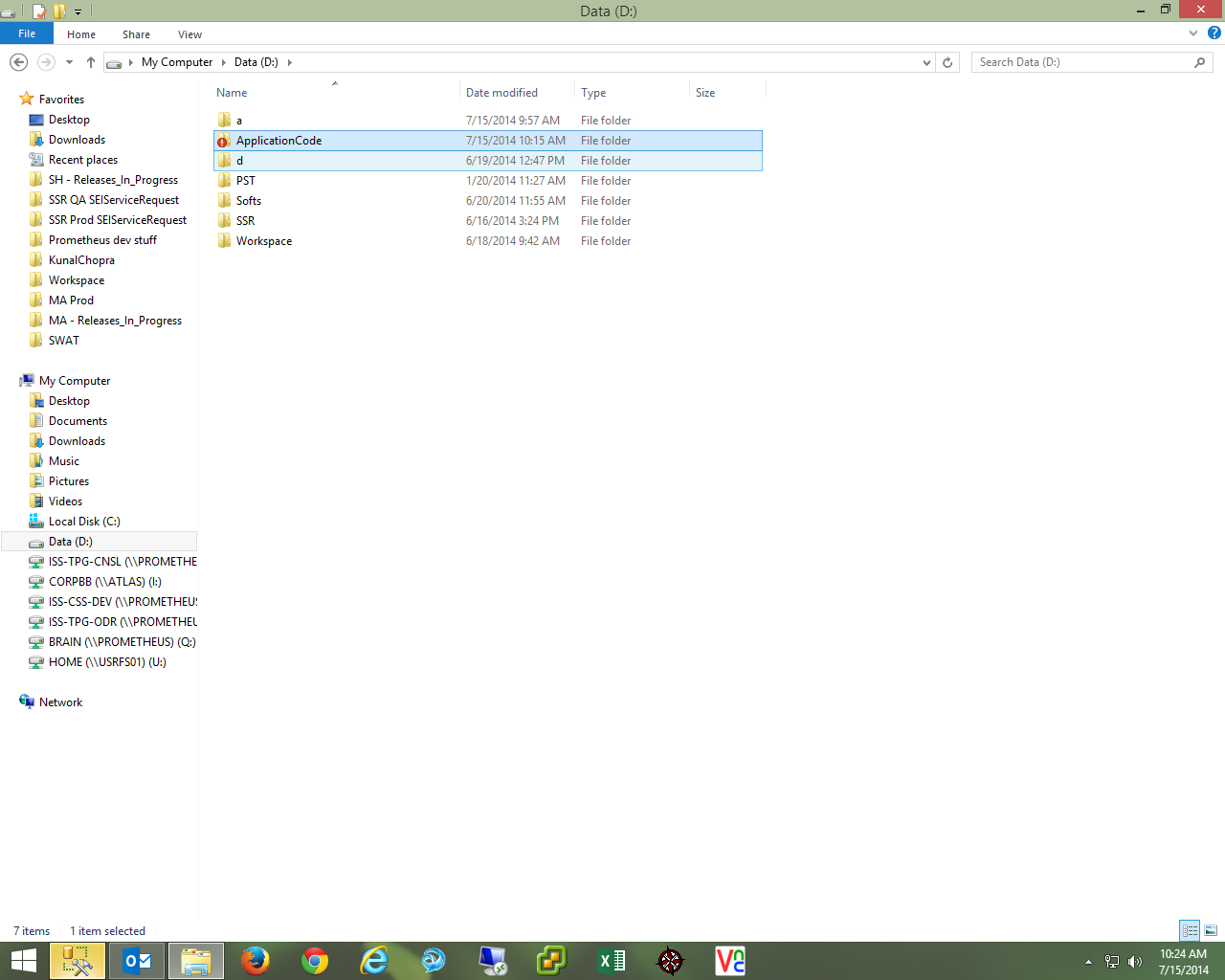
The purpose of this document is to outline and provide an insight on the processes required for maintaining the code base into the SVN repository.

## SVN Background

We all know how the code base is maintained into VSS but SVN works has some different set of terminology and the way it works. The following are some critical differences:

* SVN repository is web based library and can be accessed even using IE.
* The repository is windows authentication protected and users need to have appropriate permission to be able to access the libraries.
* Users will also need a client tool (***[TortoiseSVN](http://tortoisesvn.net/about.html), approved by SEI***) to be installed on your local machine to able to perform operation on the libraries.
* Users can also use Visual Studio plug-in AnkhSVN to work in a connected mode. (***Pending SEI Approval***)
* The most important difference between both applications is that when a user check-out the code base from VSS, the code base on the repository gets locked out with the User’s ID but in SVN nothing gets locked out on the repository as user will always have to work on the local downloaded copy of the code base.
* Revision number is maintained in SVN to quickly track the changes based on the revision number.
* Folder Icons are used to notify user if the local copy of the code is in-sync or out of sync with the source repository. (*Green tick symbol represents that code on local machine and on SVN repository is in SYNC*)





* All SVN repository should always have three different folders:

**Trunk** –

* + This folder will be used to store the most recent copy of the code base.
  + At any point of time this folder will only have one set of code base which will the pristine version of the code running on the production environment.
  + Any work in progress code should not be updated on the Trunk folder until it gets moved to the production environment.

**Branches** –

* + This will be the working folder for the development team.
  + As a process, whenever a project gets initiated, a new folder under this repository should be created with the code base from the Trunk folder.
  + Upon project completion, the code base from this repository should be merged with the Trunk and then appropriate Tag should be created.
  + Same rules apply in case of multiple releases as well. (*Make multiple working copies under the Branch folder (using the code base from the Trunk folder*) *as appropriate and then merge it with the trunk folder code base upon completion*)

Tags –

* + As the name suggests, the code base under this folder will have the point in time read only copy of the code base from the Trunk folder.
  + As a process, whenever a change/update is made on the trunk folder, a point in time copy of the same should be made under the tag folder with appropriate release detail. So, for example - if we have rolled out 5 releases, then this folder should have five different folders each one representing each release.
  + We can also say that it works like a label folder of the code base from the Trunk folder.

## http://upload.wikimedia.org/wikipedia/commons/thumb/4/4e/Subversion_project_visualization.svg/650px-Subversion_project_visualization.svg.png

Figure 1: Visualization of a simple Subversion project (Numbers in the figure are Build folders)

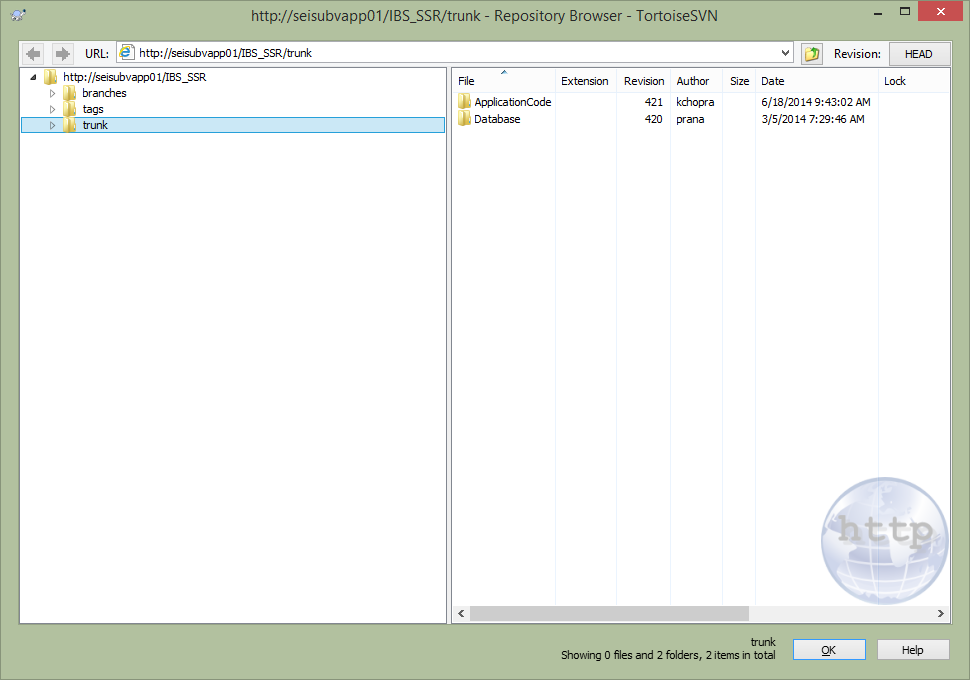


Figure 2: Repository structure for SSR Application

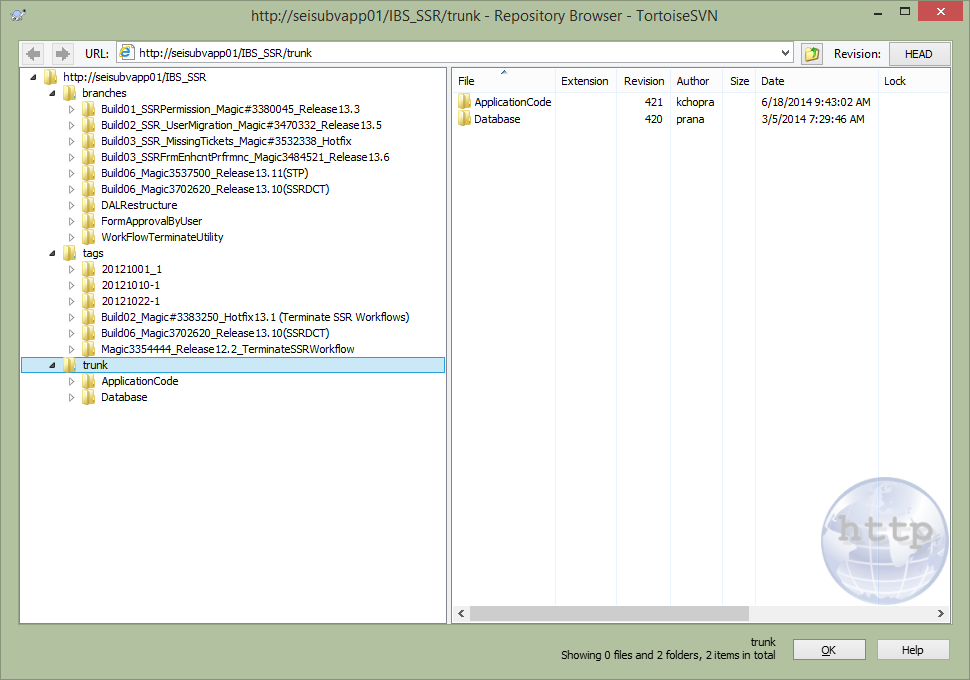


Figure 3: Repository for SSR Application

## Important activities of SVN

Developer should know the usage of the below listed activities:

1. Get Latest Code from SVN
2. Code Check-Out
3. Lock (*Only to be used if multiple developers are working on same object*)
4. Code Check-In
5. Code Merge
6. Code Labeling
7. Build Creation (*Optional step, developers can use existing process of build creation*)
8. Project/File Differences & History

## Actors SVN

**Developer:**

* This set of users will be responsible for performing normal day to day activities for maintaining the code base into SVN.
* These activities will include the following:
  + Code Check-In, Check-Out, Code Labeling
  + Creating working copy of the BRANCH or use the BRANCH through Visual Studio Plugin.
  + Merging delta’s into respective BRANCH folder every EOD. (Will happen automatically if developer use Visual studio plugin)
* Depending on the size of the project, a team can have multiple developers for a project.

**Configuration Manager:**

* Create BRANCH for folders appropriately.
* This set of users will be primarily responsible for merging the code from Branch folder to TRUNK.
* In case of multiple developers working on the same release, this user will also be responsible for creating of final merged Branch and then merging the code from final Branch to TRUNK.
* A team will have only one configuration Manager for the entire duration of the project.

**Build Master:**

* This set of users will be primarily responsible for getting the code from BRANCH folder onto the build server for creating the build.
* A team will have only one Build Master for the duration of the project.

## SVN folders for Application code base

Depending on the application current VSS structure, similar structure should be maintained under SVN repository as well.

* TRUNK - This repository will have the most current version of the entire code base in the same structure maintained in VSS PROD folder.
* TAG - The code base under this folder will have the point in time read only copy of the code base from the Trunk folder.
* BRANCHES - This will be the working folder for the development team, only impacted components per release will be moved under this repository.

## SVN folders for SQL code base

Following are the folders that will be maintained for every release from the SQL code standpoint:

1. **Build –** This folder will store all the database build structure folders with underlying customized files/scripts. The structure for the same will be as follows:
   * BuildName <Build44\_SSR Permissions\_Magic#####\_Release##> (Refer to the [appendix](#_Appendix) for Naming Conventions)
     + App
     + CMDScripts
     + Docs
     + Logs
     + RollbackScripts
     + SQLScripts
     + ReadMeFirst.txt
     + Whatsnew.txt

Only customized/one time scripts (like Input file, Rollback script, Alter table script, one time upload scripts, readme) should be included under these folders. Scripts for procedures, Index, Functions & etc. should be copied under the appropriate folders. So every time, whenever we have a new release, a new folder will be added under the Build folder with the appropriate build name.

1. **Cleanup –** We don’t have to maintain this folder and will be removed from Trunk.
2. **ForeignKeys -** This folder will store create SQL script for all the Foreign Keys objects available on the server and any new similar object script will be added to this folder with appropriate naming convention. File with the same naming convention should be delivered as part of build.
3. **Functions -** This folder will store create SQL script for all the Functions available on the server and any new similar object will be added to this folder with appropriate naming convention. File with the same naming convention should be delivered as part of build.
4. **Indexes -** This folder will store create SQL script for all the indexes available on the server and any new similar object will be added to this folder with appropriate naming convention. File with the same naming convention should be delivered as part of build.
5. **Procedures -** This folder will store create SQL script for all the Stored Procedures available on the server and any new similar object script will be added to this folder with appropriate naming convention. File with the same naming convention should be delivered as part of build.
6. **Roles -** This folder will store create SQL script for all the Roles available on the server and any new similar object will be added to this folder with appropriate naming convention. File with the same naming convention should be delivered as part of build.
7. **Schemas -** This folder will store create SQL script for all the Schemas available on the server and any new similar object script will be added to this folder with appropriate naming convention. File with the same naming convention should be delivered as part of build.
8. **Synonyms -** This folder will store create SQL script for all the Synonyms available on the server and any new similar object script will be added to this folder with appropriate naming convention. File with the same naming convention should be delivered as part of build.
9. **Tables -** This folder will store create SQL script for all the tables available on the server and any new similar object script will be added to this folder with appropriate naming convention. **Any time if we alter any table, its new create script (with updated schema) should be updated under this folder**.
10. **Views -** This folder will store create SQL script for all the views available on the server and any new similar object script will be added to this folder with appropriate naming convention. File with the same naming convention should be delivered as part of build.

## SVN Development & Release process

As stated in one of the above section, any SVN repository will always have three folders – TRUNK, TAG and BRANCH. Code Base within all these three folders will be updated throughout the life cycle of the release. As a starting point for any development and release activity, a new BRANCH will have to be created out of the TRUNK folder. All the developers will individually checkout the new BRANCH on their local machine and will start working on the code base. Once the development cycle is completed, all users will have to check-in all the code base into the BRANCH and upon Production rollout, the changes from the BRANCH will have to be merged to the TRUNK folder and a TAG will be created out of it.

Following are few scenarios with detailed steps to be followed for maintaining the application and SQL code base

**Small Release with single Developer:**

1. Configuration Manager will Check-out latest code base from the Trunk repository to a local system. (*This will be called as Trunk Working copy*) And create a [NEW BRANCH](#_New_Branch_Creation) under the Branch repository from the code base retrieved. Refer to appendix section for BRANCH naming convention.
2. Developer will create a copy of the code base from the Branch repository to a local system. (*This will be called as Branch Working copy and developer will use the same to make changes*).
3. **If developer is using Visual studio with SVN Plugin then user can skip step 2.**
4. **Based on the developer preference, user can apply lock on the particular file if multiple developers are working on the same file.**
5. Once the release/build changes have been completed, merge the changes under the Branch working copy and commit the same with the Branch repository. **Developer will have to update the Branch repository every EOD. (*Ignore this step if using Visual studio with SVN Plugin* )**
6. Create Build from the Branch folder. **Build Master should perform this activity.**
7. Once the changes have been committed to Branch repository and release has been moved to PROD environment, merge the changes with the Trunk working copy. **Configuration Manager should perform this activity.**
8. Commit the Trunk working copy into the SVN and then create an appropriate TAG.
9. Finally, delete the Branch folder created for the release.

**Note**: *At any given time, configuration Manager will have two working copies, first – a Trunk working copy which he will only use to create a branch (step 2) and then finally when he will sync Branch and Trunk (step 7). And second – a Branch working copy on which developer is supposed to make all the proposed changes.*

**Big Release with Multiple Developers:**

1. Configuration Manager will Check-out latest code base from the Trunk repository to a local system. (*This will be called as Trunk Working copy*) And create [NEW BRANCH](#_New_Branch_Creation) under the Branch repository from the code base retrieved. **All developer will share the same BRANCH and will lock the content appropriately.**
2. Developer to create a copy of the code base from the Branch repository to a local system. (*This will be called as Branch Working copy and developer will use the same to make changes*)
3. **If developer is using Visual studio with SVN Plugin then user can skip step 2.**
4. **Based on the developer preference, user can apply lock on the particular file if multiple developers are working on the same file.**
5. Once the release/build changes have been completed, merge the changes under the Branch working copy and commit the same with the Branch repository. **Configuration Manager will have to create a new BRANCH and merge the code from different Branches.**
6. Create Build from the new merged Branch folder. **Build Master should perform this activity**
7. Once the changes has been committed to Branch repository and release has been moved to PROD environment, merge the changes with the Trunk working copy. **Configuration Manager should perform this activity.**
8. Commit the Trunk working copy into the SVN and then create an appropriate TAG.
9. Finally, delete the Branch folder created for the release.

**Refer to the below detailed steps for performing all the above steps:**

Pre-requisites:

1. SVN should be installed on the local machine with **command line tools**.
2. Visual Studio and SVN plugin should be installed on the local machine.
3. Local user should have read/write access on SVN repository.

## Code Check-In & Check-Out Process

**Refer to the below steps for the code “Check – Out” process:**

Code Checkout steps:

1. Create a folder on the local machine. (This folder will be called as working folder)
2. Right click on the folder and click on SVN Checkout and specify the code location you want to check out. By default, it will check out code from all underlying folders (recursive) and with the head revision version (most latest).

**Note**: *In SVN, working folder doesn’t mean that you have checkout the code from repository. It is just a local folder with a mapping to the SVN repository so that you can commit your changes to the repository using this folder. Also, if you have checked out any code from SVN it doesn’t mean that only you can commit the code into SVN. Any other user at the same time can also create a working folder and commit the changes to the repository*.

**Refer to the below steps for the code “Check – In” process:**

Code Check-In steps:

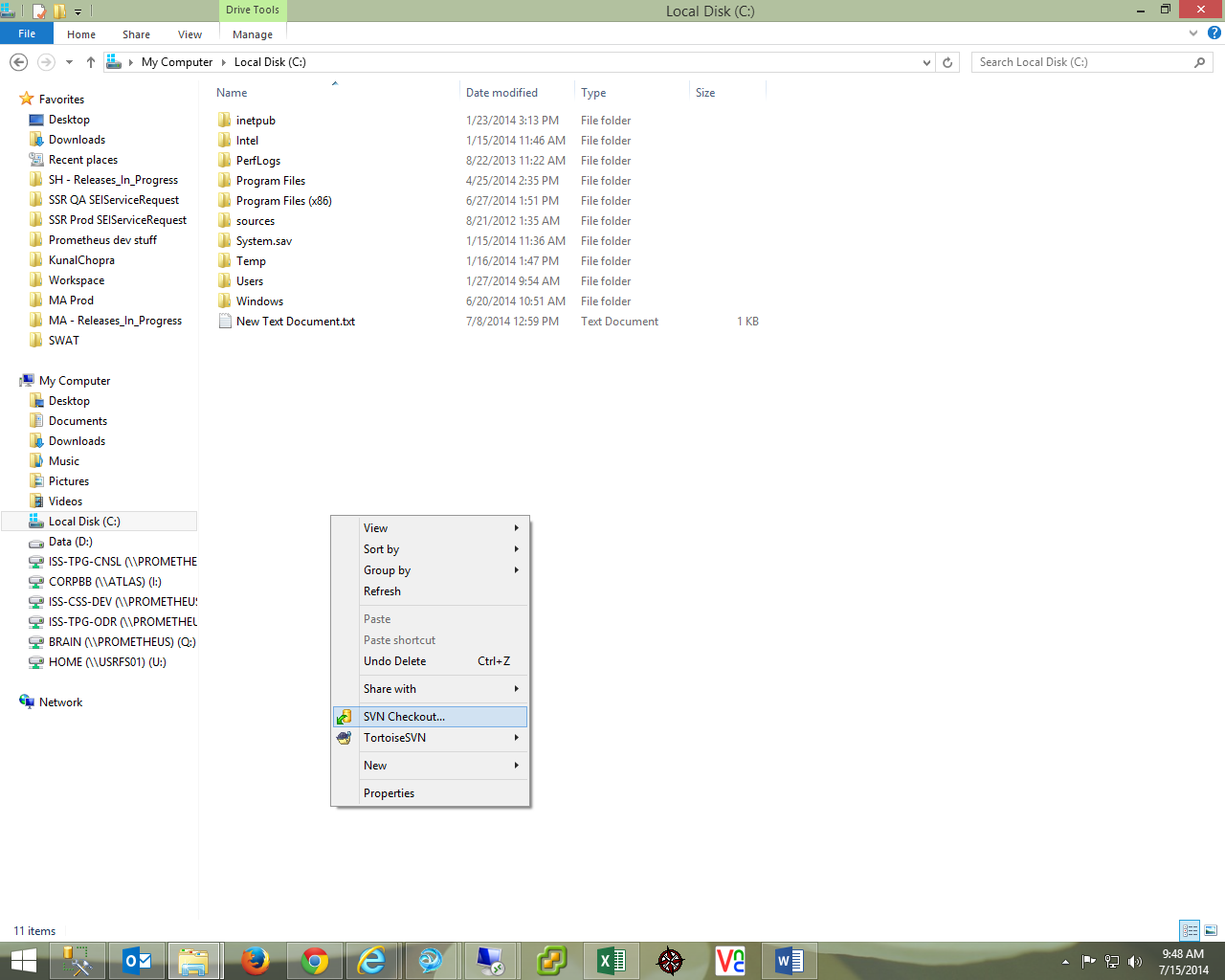
1. Navigate to the working folder created earlier during code checkout.
2. Update the changes within the underlying folders. You can also add new folders.
3. Once the changes have been completed, right click on the working folder and click on SVN Commit… option.
4. Specify the change details as **label** (Refer to the [appendix](#_Appendix) for the label text format) in the pop up window and also verify if SVN is able to identify your changes appropriately.
5. Once verified, click on OK.

## New Branch Creation

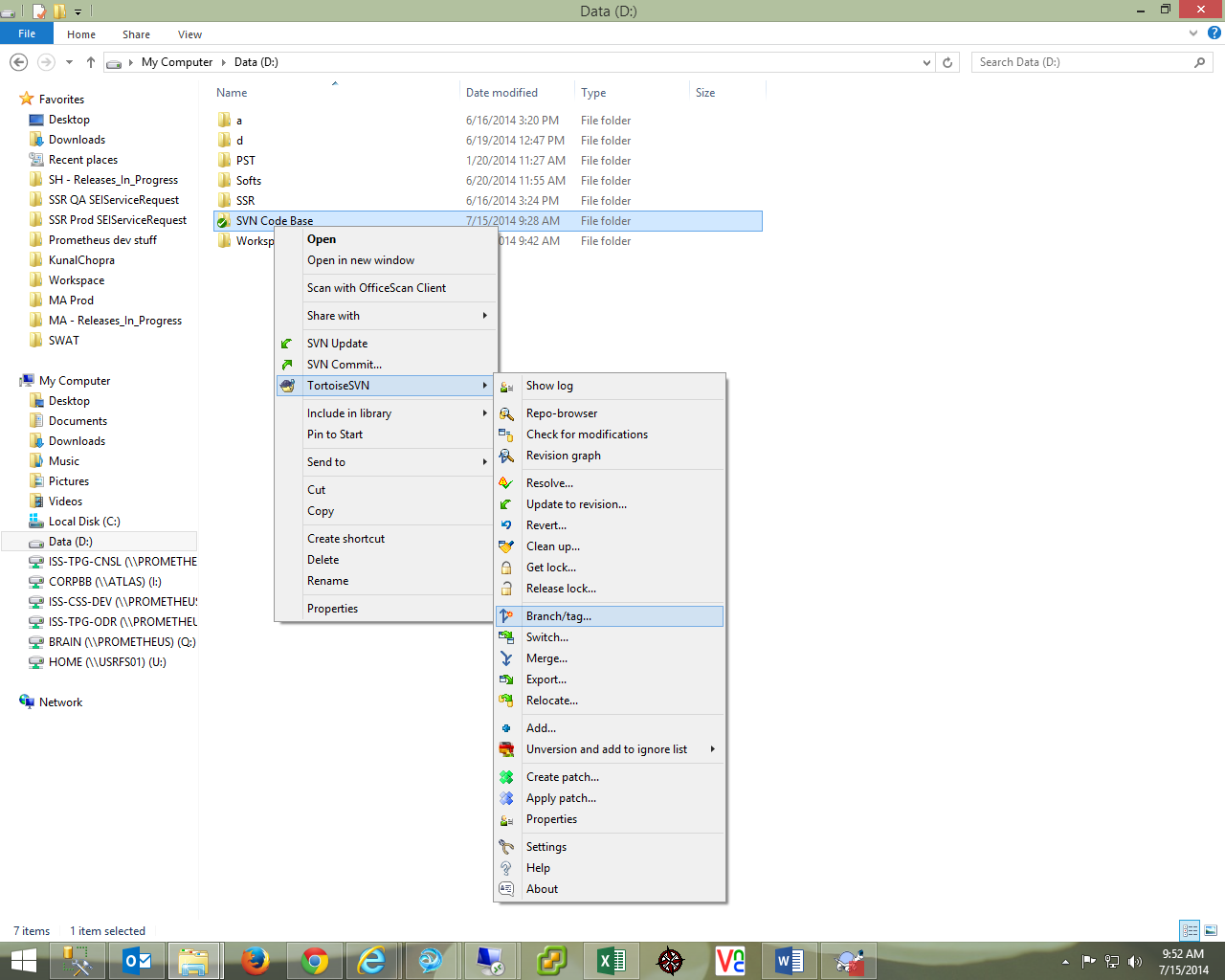
**Refer to the below steps for creating a new Branch from the Trunk Folder:**

Steps:

1. Create a new folder on the local machine to get the code from the TRUNK folder.
2. Check-Out the code base from the Trunk folder to the folder created in step 1.



1. Now, right click on the folder created on step 1 and navigate to the TortoiseSVN submenu and click on Branch/Tag.



1. Specify the new Branch path you want to create. *Eq: /branch/SSR 13.3 Permissions release.*
2. Provide an appropriate message for the labelling purpose. And choose the “head revision” radio button from the create copy option and click on OK.
3. Once the new Branch is created on the repository, Checkout and create a working Branch folder on the local machine.

## Build Creation

## Single release with one Branch folder

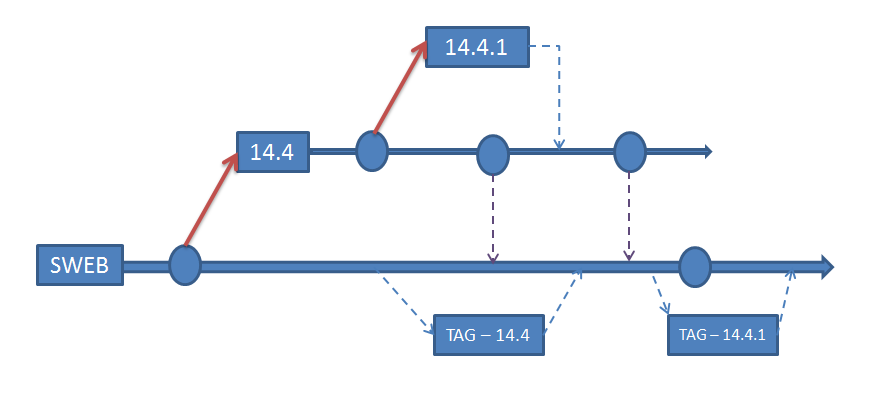
**Refer to the below steps for creating a build from the Branch Folder:**

Steps:

1. Merge the release/build changes on the Branch working copy folder.
2. Commit and Label the changes with the Branch repository.
3. Create the build as appropriate.

## Simultaneous releases with multiple Branch folders

Let’s take a project scenario to understand simultaneous releases. In this scenario we have taken a project “SWEB” having a release ‘14.4’ also has a production issue and for the same a branch ‘14.4.1’ will be created of branch “14.4”. Please refer below screenshot.



[Figure – Sample diagram to depict the simultaneous releases]

**Refer to the below steps for creating a build from the Branch Folder to execute above mentioned scenario:**

Steps:

1. Identify the Branch created for the build/release say 14.4 and retrieved the latest revision code for the same.
2. Right click on the branch folder local copy and navigate to the “Branch/Tag” submenu under TortoiseSVN menu to create a new branch say 14.4.1 of Branch 14.4.
3. Complete the development for release 14.4.1 and commit the changes in corresponding svn repository.
4. After completion of release 14.4.1, merge the appropriate code from respective Branch 14.4.1 to Branch 14.4 using the following steps to make the final build.
   1. Right click on the branch folder local copy and navigate to the TortoiseSVN submenu and click on Merge.
   2. Choose the second option “Merge a range of revisions” and click on the next.
   3. Now, choose the From URL (Which will be latest Branch URL in this case).
   4. Provide the revision range you want to merge. (Example: 4-7,9,11,15-HEAD@pegrevision)
   5. Click on “Show Log” button to validate that you are merging the code base with the latest Branch code.
   6. Once validated, click on OK and then then on next button to proceed with the merge step.
   7. Click on the Merge Button. Now, your local working copy of the Trunk will be merged with the code base from the Branch folder.
5. Commit and Label the changes with the Branch repository.
6. Create the build as appropriate.

## Sync up Branch with Trunk (Single release with one Branch folder)

Pre-requisites:

1. Code changes done for the release should have been promoted to the production environment.
2. All the changes should have been merged with the Branch folder created on the local machine and committed to the Branch repository.
3. User should have a local working copy of the Trunk folder.

**Refer to the below steps for creating a new Branch for the Trunk Folder:**

Steps:

1. Ensure that the Branch has the latest code base to be merged with Trunk.
2. Right click on the branch folder local copy and navigate to the tortoiseSVN submenu and click on Merge.
3. Choose the second option “Reintegrate a Branch” and click on the next.
4. Now, choose the From URL (Which will be latest Branch URL in this case).
5. Click on “Show Log” button to validate that you are merging the code base with the latest Trunk code.
6. Once validated, click on OK and then then on next button to proceed with the merge step.
7. Click on the Merge Button. Now, your local working copy of the Trunk will be merged with the code base from the Branch folder.
8. To commit the same on the Trunk repository, right click on the working folder and click on SVN Commit… option.
9. Specify the change details as label in the pop up window and also verify if SVN is able to identify you changes appropriately.
10. Once verified, click on OK.

## New Tag Creation from Trunk

**Refer to the below steps for creating a new Tag from the Trunk Folder:**

Steps:

1. Create a new folder on the local machine with appropriate name (Folder name can be based on the release number/magic).
2. Check-Out the code base from the Trunk folder to the folder created in step 1.
3. Now, right click on the folder created on step 2 and navigate to the TortoiseSVN submenu and click on Branch/Tag.
4. Specify the new Branch path you want to create. *Eq: /Tags/SSR 13.3 Permissions release.*
5. Provide an appropriate message for the labelling purpose. And choose the “head revision” radio button from the create copy option and click on OK.

## Code Comparison

**Refer to the below steps for comparing code base from two different folders:**

Steps:

1. Navigate to the first folder you want to compare and right click on the same and choose the “Mark for Comparison” option.
2. Now, navigate to the second folder you want to compare it with and right click on the same and choose the “Compare URLs” option.
3. A popup window will open and will give the result set.

## SVN Pull Command

Pre-requisites:

1. Make sure that you have SVN installed on the local machine with command line tools. (you would have to install this feature specifically while installing SVN by default this feature is turned off)
2. Connect to the SVN repo browser and save your credential while connecting to repo browser. (This is required only once as command line will use your saved credentials to pull the code base)
3. Verify that SVN\_Rollback\_File\_List.txt populated with appropriate file name to be pulled from SVN.
4. Verify that SVN pull CMD file is available under the CMDscripts folder.

**Refer to the below steps for pulling the SQL script using the SVN pull command:**

Steps:

1. As per the readme, update the DB build config file. (make sure the that you doesn’t have spaces in your root folder location)
2. Execute the SVN pull CMD from the CMD script folder.

**NOTE:** Development team will not use the SVN pull command/functionality for creating the build as they will still use the existing process for build creation.

## Appendix

1. **Some Important Locations:**
2. SVN Tool:

[\\atlas\SHUTTLE\Web\_Factory\b\_PRODUCT\_AND\_PROJECT\aa\_SharePoint\Softs](file:///\\atlas\SHUTTLE\Web_Factory\b_PRODUCT_AND_PROJECT\aa_SharePoint\Softs)

1. SVN Book:

[\\atlas\SHUTTLE\Web\_Factory\b\_PRODUCT\_AND\_PROJECT\aa\_SharePoint\Softs\svn-book.pdf](file:///\\atlas\SHUTTLE\Web_Factory\b_PRODUCT_AND_PROJECT\aa_SharePoint\Softs\svn-book.pdf)

1. **Naming Conventions:**
   1. Build Naming Convention:
      * + **Single Developer** - Build<Auto incremental Number>\_<15-20 character release Name>\_Magic#######\_Release<Number>
        + **Multiple Developers** - Build<Auto incremental Number>\_<15-20 character release Name>\_Magic#######\_Release<Number>\_**LanID**
   2. Build Labeling Convention:
      * + Uploaded by <SEILanID> on <DateTime> for Build<Auto incremental Number>\_<15-20 character release Name>\_Magic#######\_Release<Number>
   3. SQL object files Naming Convention:
      * + SP - IBS\_APP.<SP Name>.proc.sql
        + Table - IBS.<Table name>.table.sql