FTP Automated Build   
& Deployment Guide

Contents

[Document History 2](#_Toc352931331)

[Related Documents 2](#_Toc352931332)

[Document Overview 2](#_Toc352931333)

[1 Environment Update Overview 3](#_Toc352931334)

[2 Updating an Environment 4](#_Toc352931335)

[2.1 Step 1: Edit Build Definition - Software Version 4](#_Toc352931336)

[2.2 Step 2: Queue New Build 4](#_Toc352931337)

[2.3 Step 3: Check Build Success 5](#_Toc352931338)

[3 Fixing Deployments 7](#_Toc352931339)

[4 Deployment Configuration 8](#_Toc352931340)

[5 Managing Build Definitions 10](#_Toc352931341)

[5.1 Locate Build Definition 10](#_Toc352931342)

[5.2 Coping Build Definition 11](#_Toc352931343)

[5.3 Editing Build Definitions 11](#_Toc352931344)

[6 Queue Build 16](#_Toc352931345)

[6.1 Viewing Build Details 16](#_Toc352931346)

[6.2 Deployment Logs 17](#_Toc352931347)

[6.3 Move to Nightly Build 18](#_Toc352931348)

[7 Rig Testing 19](#_Toc352931349)

[7.1 Deployment Verification Testing 19](#_Toc352931350)

[7.2 Automated Smoke Testing 19](#_Toc352931351)

[7.3 Manual Smoke Testing 19](#_Toc352931352)

[8 Common Errors 20](#_Toc352931353)

[8.1 From Build Output 20](#_Toc352931354)

[8.2 From Setup-Deployment 20](#_Toc352931355)

[9 Appendix A: Creating Code Branches 21](#_Toc352931356)

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Changes** | **Changed By** |
| 31/05/2010 | 0.8 | First draft completed | Rik Locke |
| 19/06/2012 | 0.9 | Added Lab Manager sections | Jason Blackford |
| 19/06/2012 | 1.0 | First pre-release review after session with Customer Care team. | Rik Locke |
| 9/9/2012 | 1.1 | Updated for handover to dev teams. | Rik Locke |
| 18/04/2013 | 1.2 | Updates |  |
| 15/10/2013 | 1.3 | Updates on the build definitions | Ebru Cucen |

This document is stored in TFS Source Control at; $/Deployment/Main/Documents.

Any version of this document not from this location is not controlled.

# Related Documents

The following documents can provide more specific details related to this overall process

They are stored and controlled in the same location as this document.

FTP Build and Deployment Dev Guide.docx

FTP Deployment Troubleshooting Guide.docx

# Document Overview

This document is intended to provide guidance for the use of automated functional builds with automatic rig deployment for teams within TfL's FTP project.

The first sections go over the process from a very high level, day to day perspective

# Partitioned Builds

To be able to deploy from the same package, we have partitioned the builds. There are CI builds to create a package so that we can keep them indefinitely to use whenever we want, and there are deployment builds responsible just for deployments. The CI build compiles the application resources defined in the Items to Build section in Process section whereas Deployment build compiles the deployment resources which is mainly Deployment solution of the specified branch. For deployment builds, there are two options: to get last successful CI build or a specific CI build.



|  |  |
| --- | --- |
|  | Step1. Queue the CI build  Check the CI build to see if it meets the requirement for :  Correct branches are defined in Workspace section and in Process section for Items to Build and Tests.  Step2. Edit the Deployment build to get the specific build or the latest successful build.  NB in Camden and before we only support one ‘partitioned build to be specified’  If you want a specific build, Build Number should be filled with this value, otherwise it will take the last successful build.    Step2 Post Camden: We allow multiple partitioned builds to be specified from Deptford. By clicking the ellipsis to edit the ‘Partitioned Build Settings’ you are presented with a dialog where you can specify multiple build and optionally, specific build numbers.    Step3. Queue the Deployment build  Step4. Check the Build Success |

|  |  |
| --- | --- |
| Environment Update Overview for Unpartitioned Deploys | |
|  | The diagram here illustrates the flow of the automated build process.   1. Check the Build Definition  * Software Branch * CI build it will take to deploy * Environment to deploy  1. Queue the Build 2. Check Build Success  * Build results * Post-deployment test results * Logs   Deployment Server:  DevInt: TDC2BLD001  *PreProd: FPDC2MGT001 (Internal Jump Server)*  *For LabManager rigs*  *TS-DB1 is the jump server to do deployment.* |

# Updating an Environment

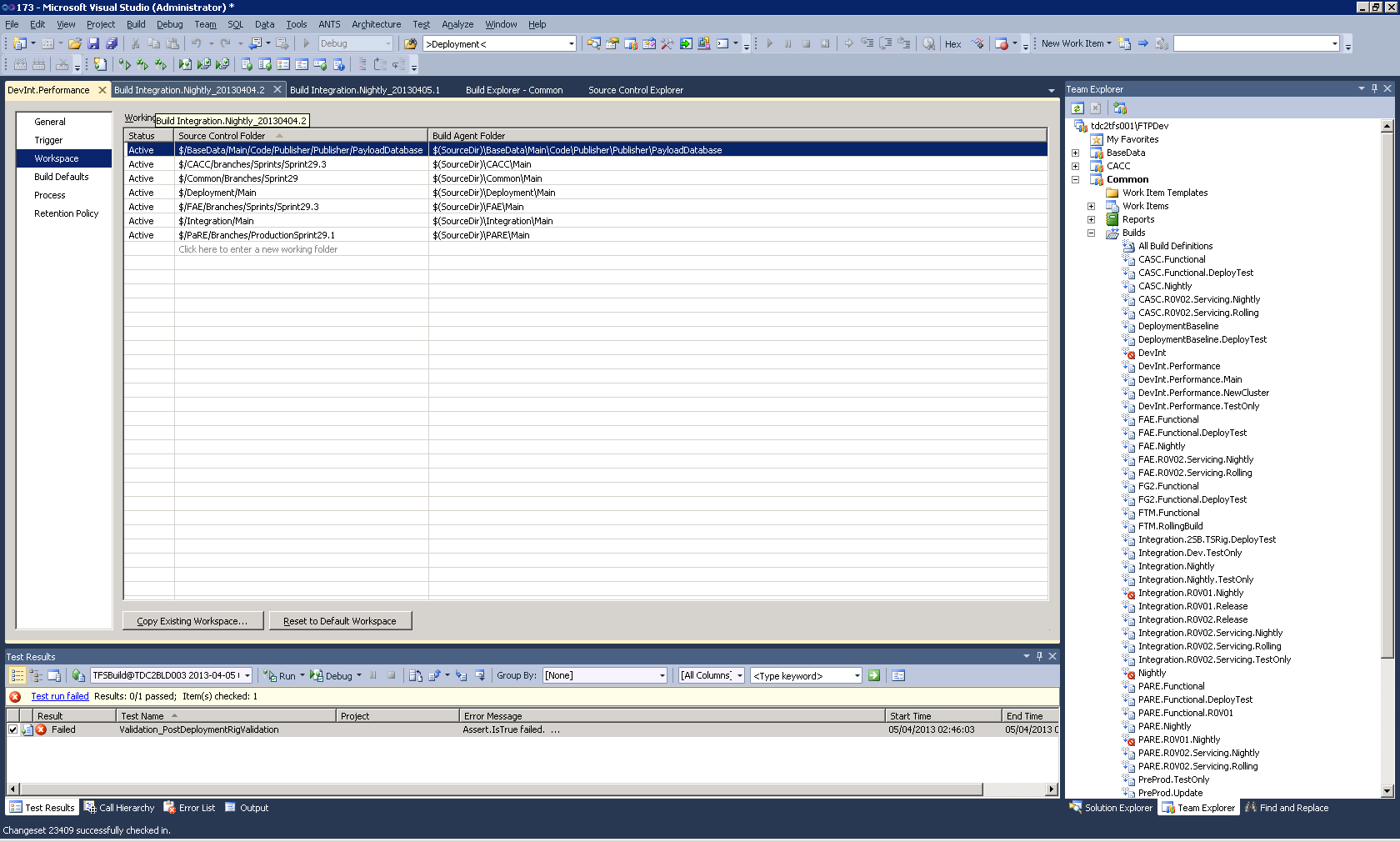
## Step 1: Edit Build Definition - Software Version

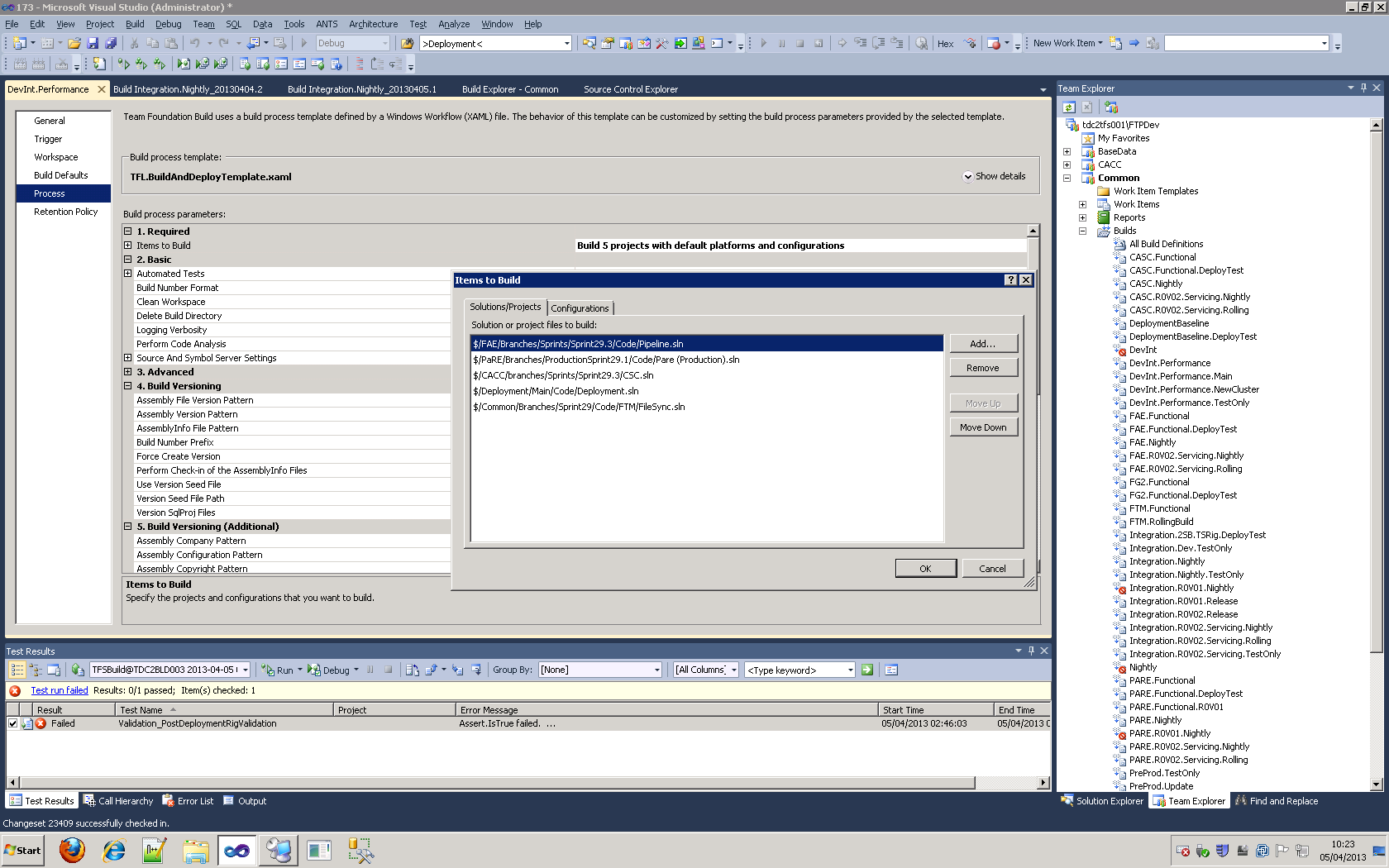
The source code to be deployed needs to be defined is required in 3 places:

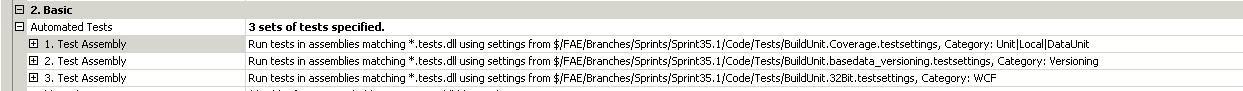
a. Workspace: The source code needs to be defined from the relevant branches.

b. Items To Build: The solutions need to be compiled under Process section should be from the relevant branch.

c. Tests: The Automated tests to be added in the Process







In each case select the desired CASC, FAE and PARE versions and save the Build Definition.

Everything required to build the solutions specified in Process section must be covered by the Workspace otherwise the build will fail.

### Base Data DB

Sometimes there will be changes which mean an update to the BaseData database will be required.

If this is the case then acquire an appropriate backup of the BaseData database and restore it over the existing database in the target environment

See the "FTP Package Deployment Guide" for more specific details on this.

## Step 2: Queue New Build

Exactly as it says on the tin, right click and queue. Some environment deployments are disabled on purpose to control the deployment process. After each change their initial page needs to be updated, queued, and re-disabled.

See the Queuing Builds chapter of this document for more details.

## Step 3: Check Build Success

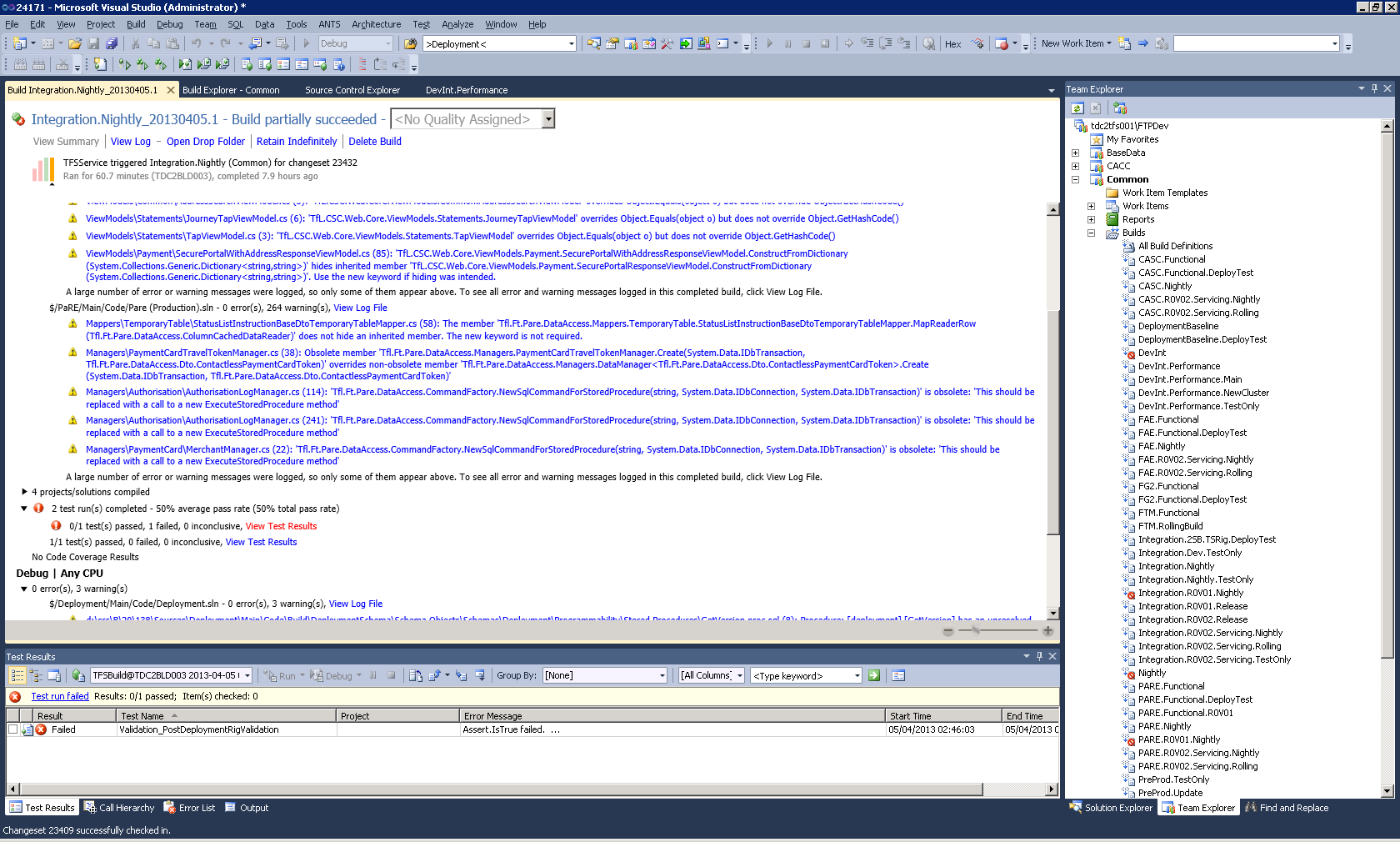
### Build Result

If the build itself has failed then ait will need to be fixed and re-started.

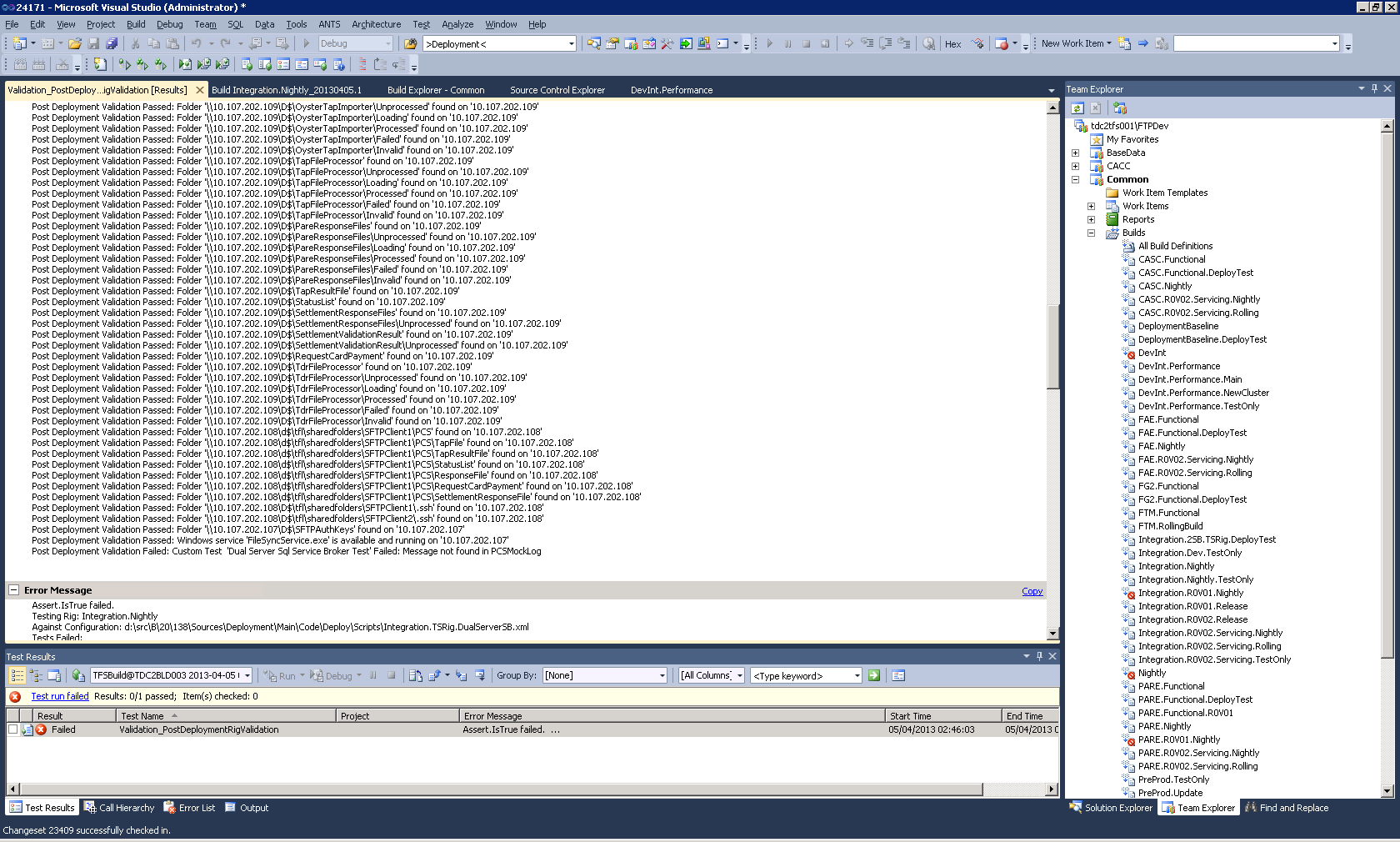
There is a brief section towards the end of this document about troubleshooting or you can refer to the "FTP Build and Deployment Dev Guide"

### Post-Deployment Tests

The build summary shows details about the tests that have been run and which were successful or not.



Double-clicking or right-click, View Test Result Details shows the test log which looks like this;



### Logs

There are numerous sources of information that can be examined to find out what has caused an issue with a deployment.

There are many log files in different locations so it may be helpful to do a search on the whole drop folder for all \*.log files.

#### Deployment Logs

Deployment logs are created by the powershell scripts that execute the deployment of software.

From the build results, select Open Drop Folder and looks for either \Logs\ or \Deployment\Logs\

Deployment Summary Log

The first thing to check is whether any of the software modules failed to deploy properly.

This document is called "DeploymentSummary.<packagename>.log" and is usually found in \Logs\ in the root of the deployment folder.

This simple states whether each 'Server Role' was deployed successfully or not.

#### Deployment Log

This file, "Deployment.<packagename>.log" , is found next to the summary log and contains full details of the output (as would be seen in the console) of the powershell deployment scripts.

When you know what module has failed you can find specific details of the error recorded in here.

#### Role Deployment Logs

These are found in the same log folder as the Deployment Logs above.

#### Software Deployment Logs

Each FTP Server Role deployed produces its own log which can contain more details than the overall deployment logs above.

#### Database Deployment Logs

Each individual Database Role (dacpac project) creates a log on the Database Servers (not the deployment server).

e.g. \\10.107.25.4\D$\DatabaseDeployment\<dacpac>.log

*(10.107.25.4 is one of the DevInt cluster nodes)*

#### FTP Software Logs

Some faults can be caused by the applications themselves rather than by the deployment process.

Often this is due to configuration issues such as connection strings or permissions problems.

If a service has failed to start or has thrown some error causing a fault then there are logs created on the relevant application server under \Logs\ in the folder that the software has been deployed to.

# Deployment Configuration

Each environment has one or more xml files that define the machines in that environment and what software gets deployed to which machine.

A greater level of detail can be found in the "FTP Build and Deployment Dev Guide"

For example;

<drop\_folder>\Deployment\Scripts\DevInt.Internal.Performance.xml

And in source control;

$\Deployment\Main\code\deploy\scripts\DevInt.Internal.Performance.xml

Mostly, these contain include references to each full role definition held in the various Common xml files;

|  |  |
| --- | --- |
| **Common file** | **Contains definitions for:** |
| CommonServerRoles | Server Roles (all non-database roles) common to all environments |
| CommonProductionServerRoles | Contain server role definitions where there are differences between Production and LabManager deployments such as Web Site names |
| CommonLabServerRoles |
| CommonDatabaseRoles | Database roles for lab manager rigs (these have different db instance names that are deployed to) |
| CommonLabDatabaseRoles |
| CommonServiceDeployRolesAutomatic | Server Roles deploying services differentiated by the service start up mode. |
| CommonServiceDeployRolesManual |

<?xml version="1.0" encoding="utf-8"?>

<configuration xmlns="http://tfl.gov.uk/DeploymentConfig" Id="1" Name="DevInt.Internal.Performance" Environment="DevIntPerf">

<CommonRoleFile>CommonServerRoles.xml</CommonRoleFile>

<CommonRoleFile>CommonServiceDeployRolesManual.xml</CommonRoleFile>

<CommonRoleFile>CommonProductionServerRoles.xml</CommonRoleFile>

<CommonRoleFile>CommonDatabaseRoles.xml</CommonRoleFile>

<PostDeploymentTestIdentity>DeploymentAccount.Internal</PostDeploymentTestIdentity>

<machine Id="30" Name="TDC2BLD001" ExternalIP="10.107.197.67">

<ServerRole>Deployment</ServerRole>

</machine>

<machine Id="0" Name="TDC2FAE029" ExternalIP="10.107.203.208" Role="CIS Server 1">

<ServerRole Name="TFL.IISsetup" Description="IIS" />

<ServerRole Name="TFL.WebDeploy" Description="CSC Web Service" Include="CACC.Web.Service" />

<ServerRole Name="TFL.WebDeploy" Description="CSC Lookup Service" Include="CACC.Lookup.Service" />

<ServerRole Name="TFL.WebDeploy" Description="CSC Membership Service" Include="CACC.Membership.Service" />

...

Where environments differ in the deployment of a role they will include a reference to the same definition but include different common files.

These files will only need to be modified if new pieces of software have been added to the deployment.

## Key Configuration Points

See the "FTP Build and Deployment Dev Guide" for more details.

Environment="DevIntPerf"

This value is used for the $Environment SQL CMD variable that governs which versions of app.config, web.config and pre/post-deployment scripts.

<PostDeploymentTestIdentity>DeploymentAccount.Internal</PostDeploymentTestIdentity>

After the software is deployed, a set of post-deployment tests are run.

The account named here is taken from the   
<drop folder>\Deployment\Config\<env>ServiceAccounts.xml file.

The tests are run under the identity of this account.

<machine Id="30" Name="TDC2BLD001" ExternalIP="10.107.197.67">

<ServerRole>Deployment</ServerRole>

</machine>

This specifies to which machine the deployment will be executed from. The software is copied to this server (to D$\Deploy\<package\_Name>\ and the powershell script (D$\Deploy\<package\_Name>\Deployment\scripts\Delpoy-RigFromConfig2.xml) is remotely executed from the build server.

Usually this is done via the environments jump server but in the case of DevInt (which doesn't have an internal jump server) the BLD003 is used.

If new software is required to be deployed, it's usually good to start by duplicating an example of a similar piece of software and editing values accordingly.

### Database Roles

<DatabaseRole Name="FromConfig" Description="FAE Database" Include="FAE.Database">

<ProjectStub>CommonDataAreaDb</ProjectStub>

The name of the dacpac file to be deployed

<TargetDatabase>FAE</TargetDatabase>

<DatabaseInstance>vins001</DatabaseInstance>

<PublishProfile>CommonDataAreaDb.$Environment.Publish.xml</PublishProfile>

The name of the publish profile that provides sql cmd variables.

<PreDeployment>Scripts\Patching\FAE.Patching.sql</PreDeployment>

Deployments that just upgrade existing databases may be required to execute custom scripts for steps that cannot be handles by the dacpac deployment. The patching script is where these sql scripts are added or included and are run before the sql project is

<PostDeployment></PostDeployment>

Dacpac based deployments run their own pre and post-deployment scripts.  
This element instructs the deployment scripts to execute this script after the database project has been deployed.

<AlwaysCreateNewDatabase>false</AlwaysCreateNewDatabase>

If se to true, the target database will be completely deleted and re-created by the deployment.

<TestInfo>

<Sql>

IF (SELECT count(\*) FROM $TargetDatabase.information\_schema.tables WHERE table\_schema ='travel' and table\_name='Journey') = 0

THROW 51000, 'Table [$TargetDatabase].[TRAVEL].[JOURNEY] does not exist', 1;

</Sql>

</TestInfo>

The post-deployment tests for database projects will execute the script specified in order to determine if the deployment was successful.

</DatabaseRole>

### Web Roles

<WebDeploy Name="CSCSupportService" RegistryKey="Software\TfL\FT\PARE\CIS" AssemblyToVersionFrom="Pare.CSCSupportService.dll">

This dll's version is written to the specified registry key.

<AppPool>

<Name>PARECSCSupportService</Name>

<ServiceAccount>PAREServiceAccount</ServiceAccount>

Accounts are named according to sets of credentials stored (with encrypted passwords) service accounts xml file (<drop folder>\Deployment\Config folder)

</AppPool>

<Site>

<Name>Pare.CSCSupportService</Name>

<Port>8722</Port>

<PhysicalPath>D:\tfl\PaRE\CSCSupportService</PhysicalPath>

<ApplicationPool>PARECSCSupportService</ApplicationPool>

</Site>

<Package>

<Name>CSCSupportService</Name>

</Package>

<TestInfo>

<EndPoint>CSCSupportService.svc</EndPoint>

<EndPoint>DenyListService.svc</EndPoint>

</TestInfo>

The post-deployment tests will use the specified endpoint when making an http request to determine if the deployed web site is running and responsive.

</WebDeploy>

### Service Roles

<ServiceDeploy Name="FAEEngineController">

<MSI>

<id>EED8A1A9-891E-4D78-9C56-532939489846</id>

Required when uninstalling an existing service

<name>Controller.msi</name>

<installlocation>d:\tfl\FAE\Controller\</installlocation>

</MSI>

<Services>

<Service>

<Name>EngineControllerHost.exe</Name>

<Credentials>FAEServiceAccount</Credentials>

Again, this is a named reference to a credential set in the \Configs\ServiceAccounts.xml file

<StartUpType>Automatic</StartUpType>

</Service>

</Services>

<Configs>

<config name="EngineControllerHost.exe.config" target="\tfl\FAE\Controller"/>

<config name="Tfl.Ft.Fae.OysterTapImporter.exe.config" target="\tfl\FAE\Controller"/>

</Configs>

Windows service config files must be explicitly copied by the deployment powershell scripts, this list tells the scripts which files are to be copied

</ServiceDeploy>

# Fixing Deployments

## Re-deploying Individual Roles

As the deployment config file defines what gets deployed where, you can repeat parts of deployments by copying and editing the config file.

For example, to re-run the PARE installer on DevInt;

1. Copy the D:\Deploy\PreProd.Update\_20130404.2\_Internal\Deployment\Scripts\PreProd.Internal.xml file.
2. Rename it to (for example) PreProd.Internal.PareOnly.xml
3. Edit it and delete all <machine> elements except the one for the pare server.
4. Execute the deploy command on the new config;

powershell -ExecutionPolicy Unrestricted "D:\Deploy\**PreProd.Update\_20130404.2\_Internal**\Deployment\Scripts\Deploy-RigFromConfig2.ps1" -RigRelativePath **'PreProd.Internal.PareOnly.xml'** -Password 'Olymp1c$2012' -DeploymentLogFolder **'D:\Deploy\PreProd.Update\_20130404.2\_Internal\Logs'** -EnableRemoting 0

*(EnableRemoting is only required for new environments and LabManager rigs)*

## Re-running Post-Deployment Tests

If a deployment problem has been resolved and needs to be re-tested, the DeploymentTool.exe which runs the post-deployment tests can be executed manually on the deployment server

In a DOS prompt, run the following command;

"D:\Deploy\**PreProd.Update\_20130404.2\_Database**\Deployment\Tools\Deployment Tool\DeploymentTool.exe" -Type 'Post' -ConfigFile 'D:\Deploy\**PreProd.Update\_20130404.2\_Database**\Deployment\Scripts\**PreProd.Database.xml**' -ServiceAccountsFile 'D:\Deploy\ **PreProd.Update\_20130404.2\_Database**\Deployment\Config\PreProd.ServiceAccounts.xml' and -Password 'Olymp1c$2012'

The specific folder and deployment config file **in bold** will need to be changed depending on the config being tested.

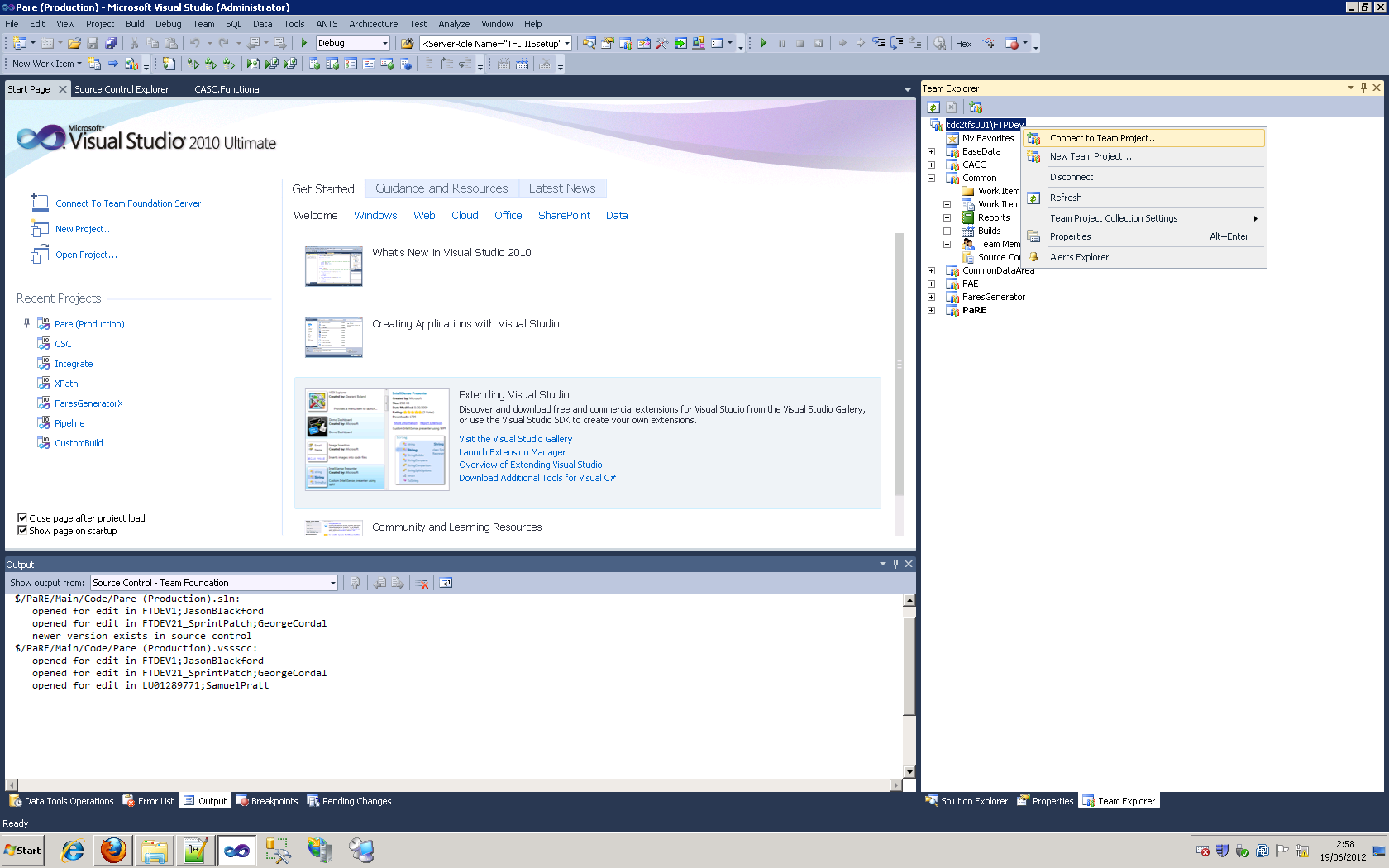
# Managing Build Definitions

## Locate Build Definition

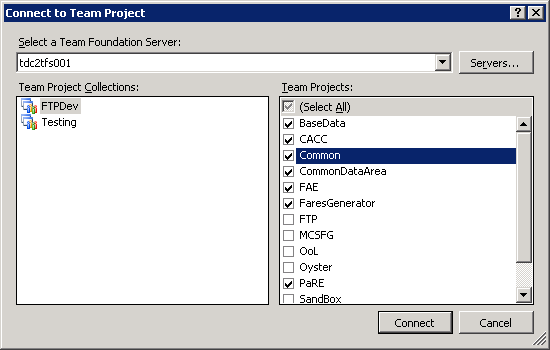
Currently almost all of the build definitions are stored under the Common area;

If you can’t see the Common area, then select it via the Connect to Team Project dialog.

Right-click the root node of the Team Explorer and select “Connect to Team Project...”



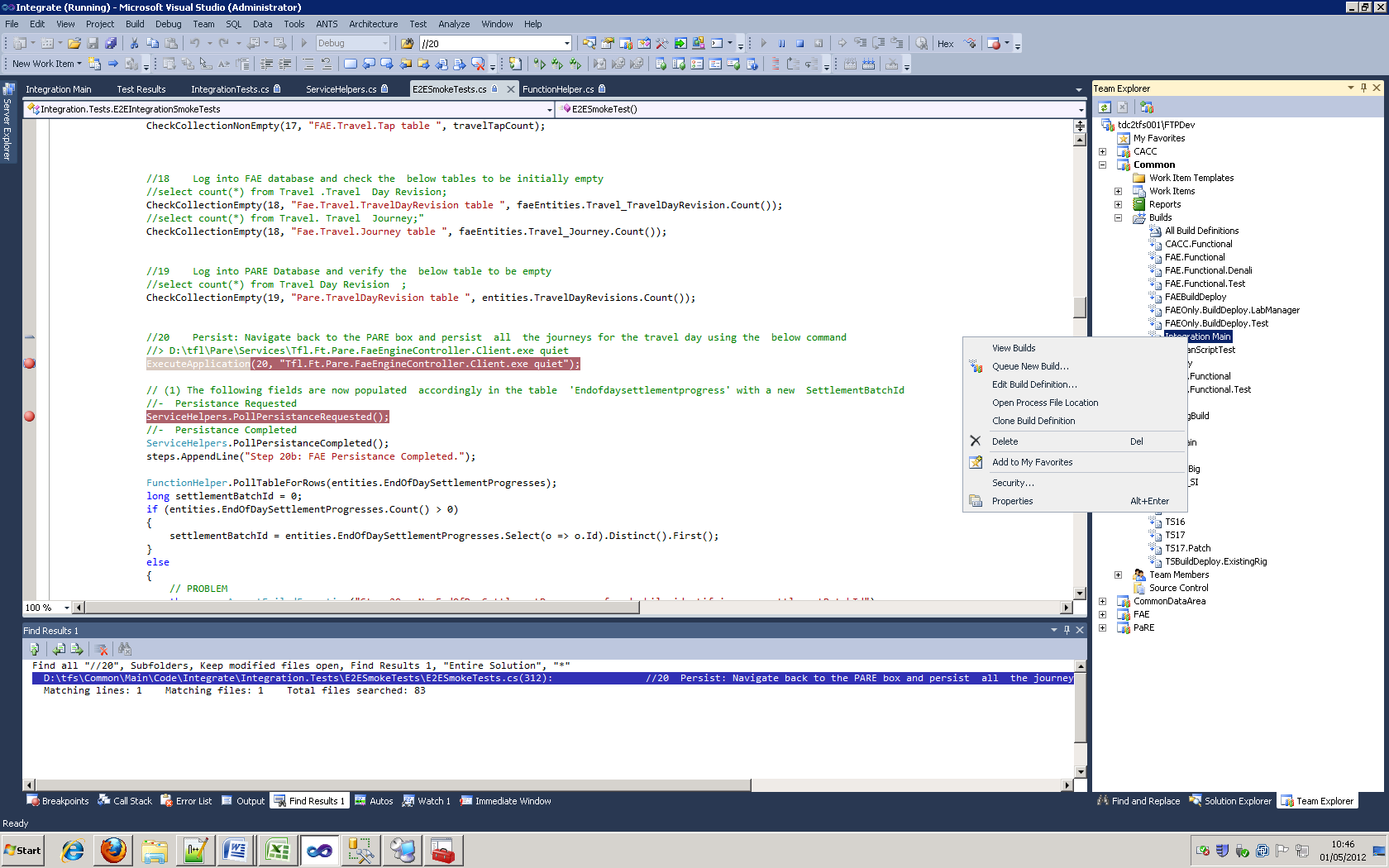
Check the Common entry in the Team Project list and click “Connect”.



## Coping Build Definition

Typically, the team’s functional build will just be updated with the appropriate version references but you can easily make a new definition based on an existing one if desired.

Select 'Clone Build Definition' from the right-click context menu for the build definition to be cloned.....



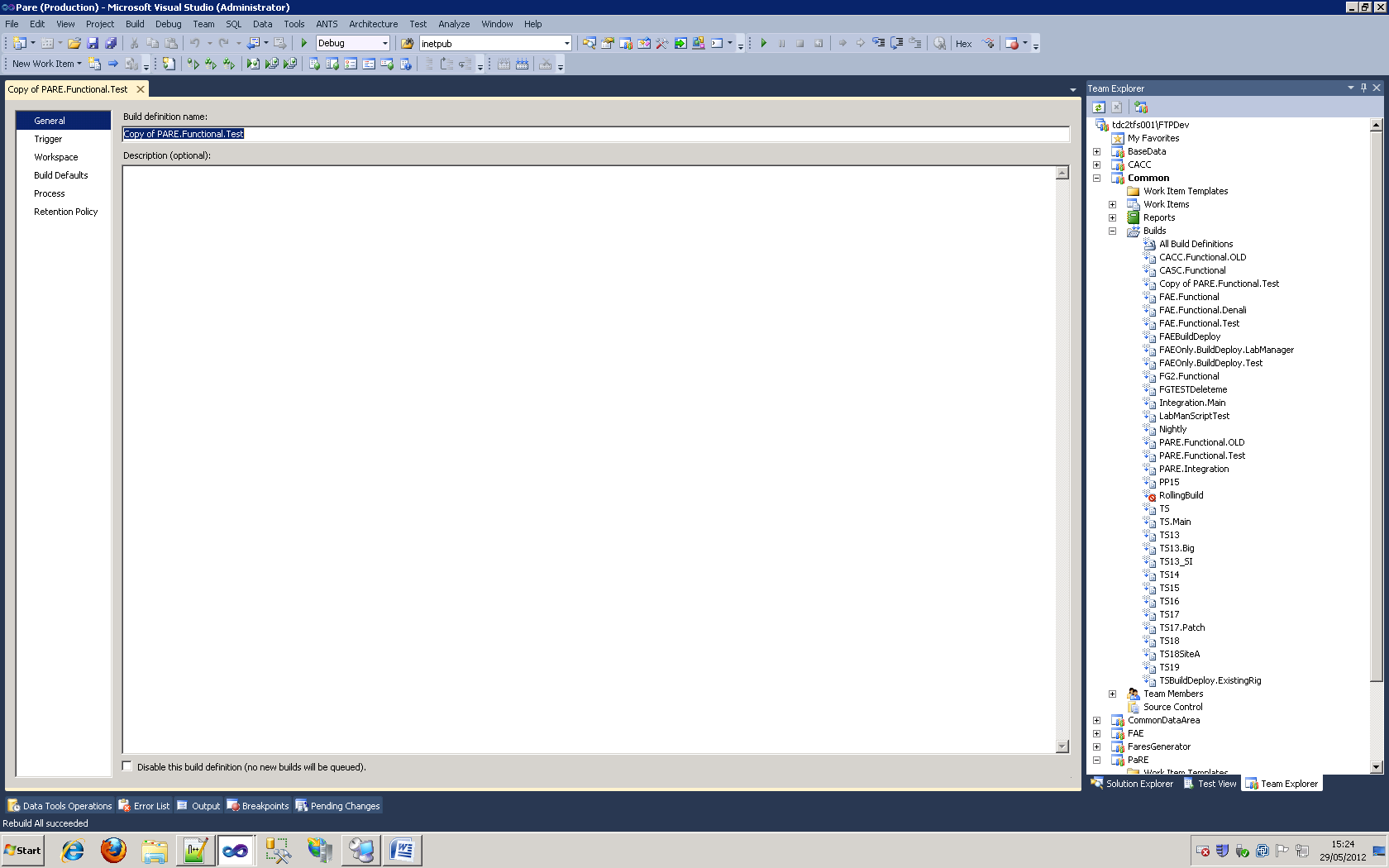
## Editing Build Definitions

Select "Edit Build Definition" from the context menu and edit the each section as required:

### General

If this is a copy of an existing build definition, rename it in the General section.

The typical naming convention for a build definition is <team>.<environment>[.<very\_short\_desc>]



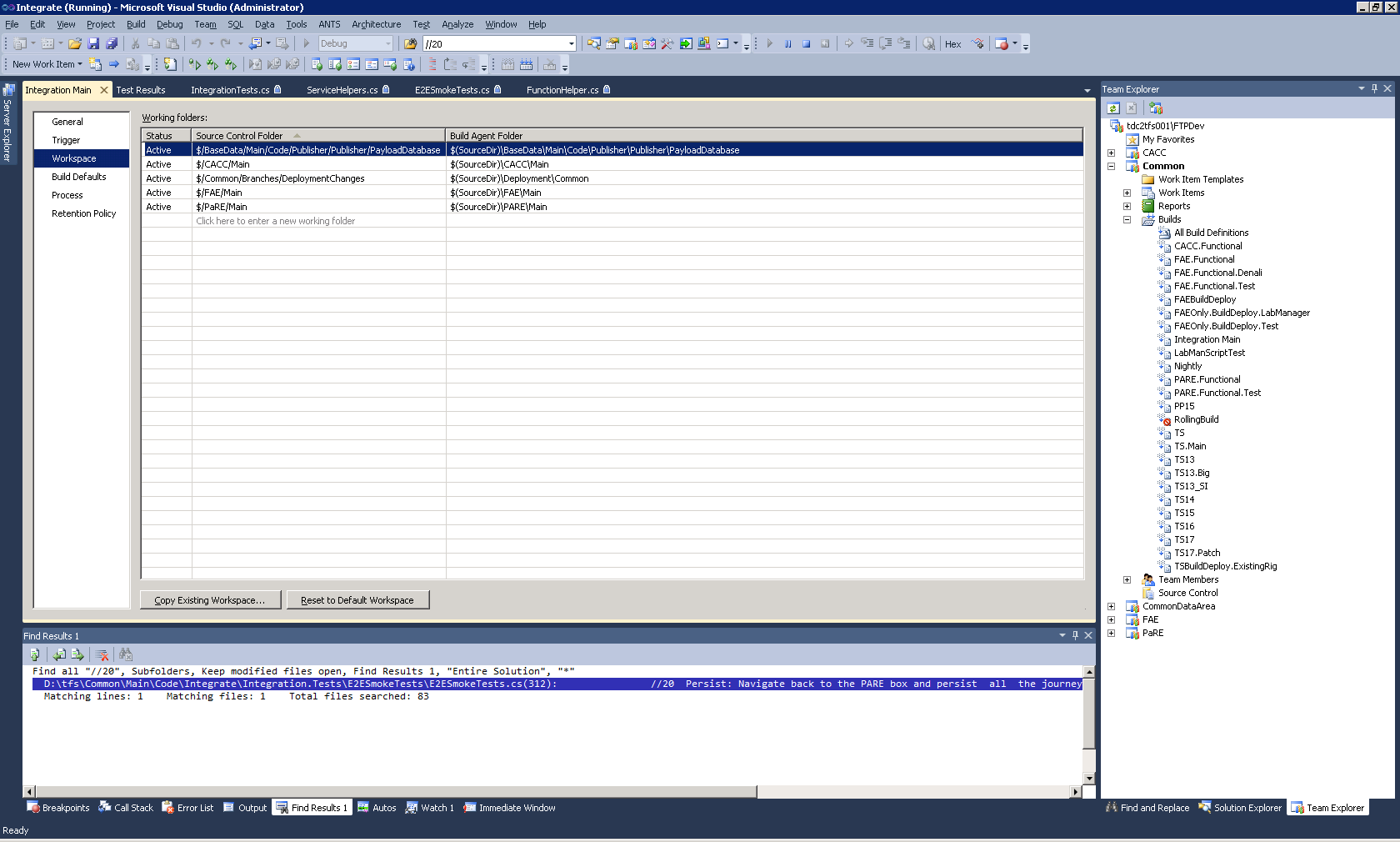
### Triggers

For Functional builds will be left as 'Manual'.

If the build definition has been copied, check that is it is not anything other than manual.

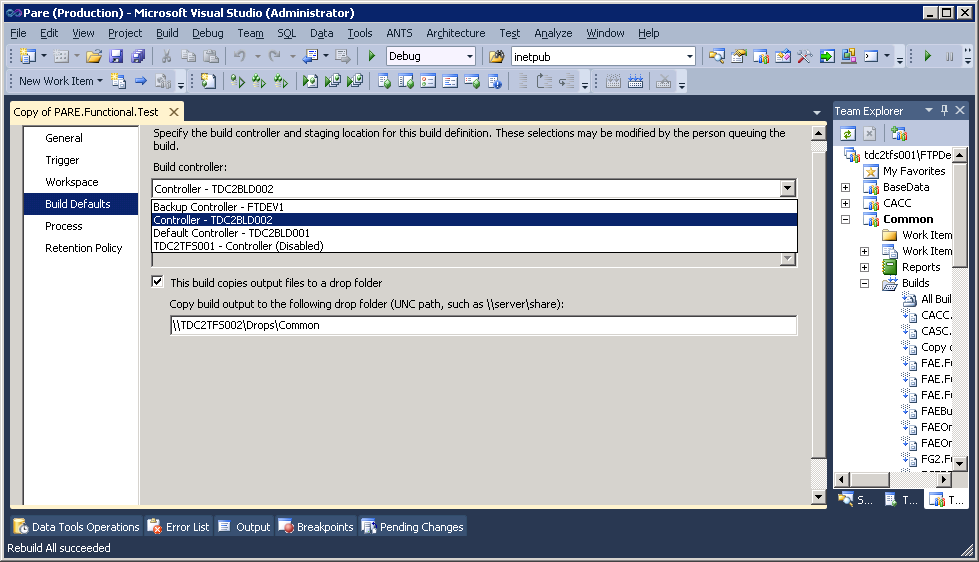
### Workspace

Set the source paths for the code projects to be included in the build



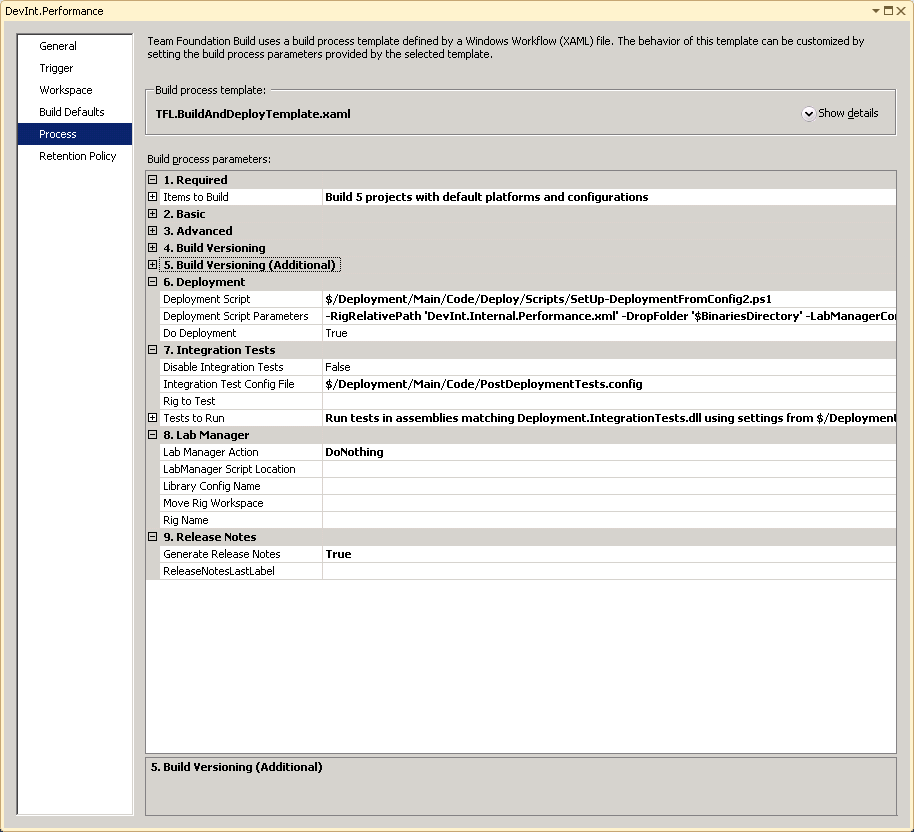
### Build Defaults

Generally speaking TDC2BLD003 is used for rolling builds and 004 for ad-hoc builds.



The output should be copied to the TFS server as shown, to a directory representing the product (PARE, FAE etc)

### Process



Section 1: Items to Build.

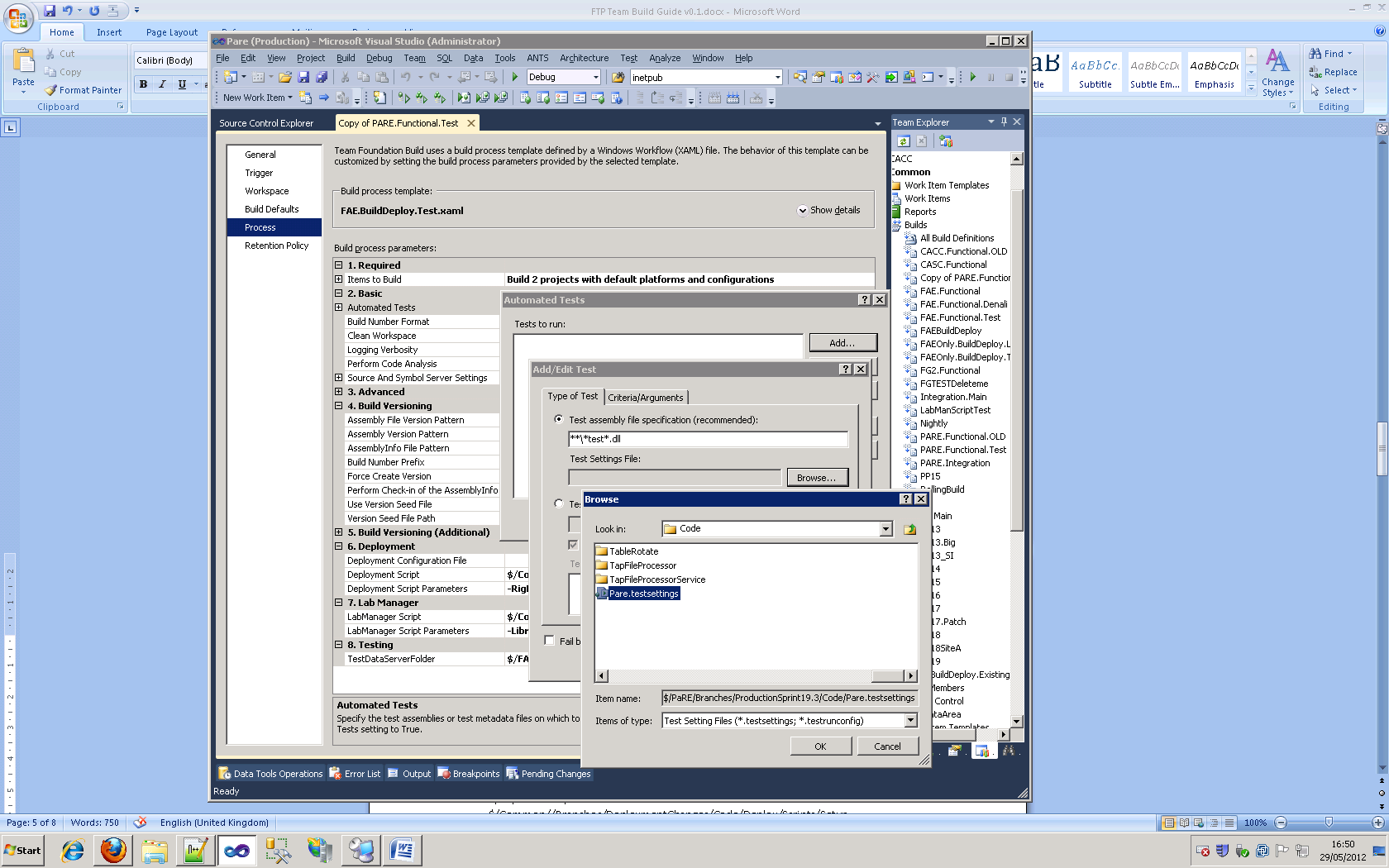
Any code branches that may have changed need to be rebuilt by specifying them here.

Any branches being built must exist under a path specified in the Workspace section. So changing sprint branches must be done in the workspace as well as the build items (see below) and the test settings (see next).

Section 2. Basic: Automated Unit Tests

Functional deployments will normally have a series of unit tests in the project which should be run as part of the build process.

To run these select .testsettings files from the project(s) being built.



Open the Automated Tests dialog by clicking the browse button on the Automated Tests line.

Click the 'Browse...' button under Test Settings File.

Locate the testsettings file and click Ok.

Section 6: Deployment

For fully automated build and deployments the deployment script is specified here;

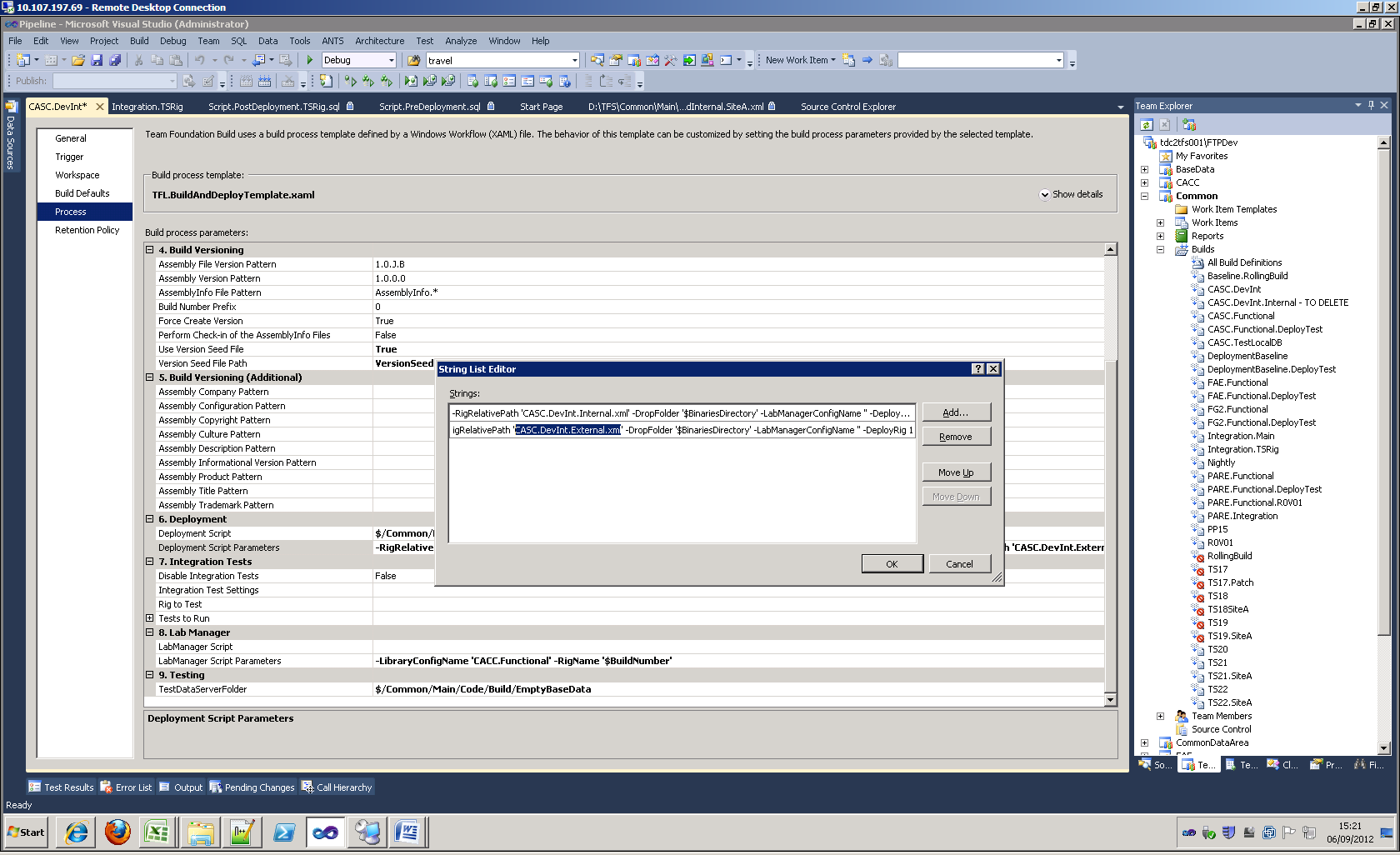
Deployment Script:

$/Common/Branches/DeploymentChanges/Code/Deploy/Scripts/Setup-DeploymentFromConfig2.ps1

Deployment Parameters:

|  |  |  |
| --- | --- | --- |
| Parameter Switch | Example Value |  |
| RigRelativePath | 'Integration.TSRig.xml' | The xml configuration file, relative to the Deploy-RigFromConfig.ps1 location, containing the machine and software configuration to be deployed. |
| DropFolder | '$BinariesDirectory' | The source location of the build output directory containing the software to be deployed |
| LabManagerConfigName | '$BuildNumber' | The name of the LabManager rig to be deployed to. $BuildNumber is of the format <BuildDefinitionName>\_yyyyMMdd.*n* (where *n* is the nth build of the day) |
| DeployRig | 1 | Default 0.  If 1 will execute the Deploy-RigFromConfig.ps1 script to copy and deploy the software to the machine with the Deployment server role in the rig config file. |
| DatabaseDeployment | 'Local' | Optional:  Local: DEFAULT: Copies the database project dacpacs to the target server and deploys them locally.  None: Skips deployment of all database projects. |

For environments that have multiple config files such as DevInt and SiteA's internal and external domains you can specify this multiple parameter sets with different RigRelativeRath values.



The Setup-Deployment script will be run once with each set of parameters.

Section 7: Integration Tests

This section mirrors the out of the box section 2 in allowing the specification of a set of mstests that will be run after the rig and software have been built and deployed.

Disable Integration Tests: As described.

Integration Test Settings: The testsettings file to use in the test run.

Rig To Test: If blank this will run against the rig created for this build

Tests to Run: Specified the test assemblies and category filters to apply as normal.

Section 8: Lab Manager

Lab Manager Script:

If software is being deployed to an existing rig the Lab Manager Script entry is left blank.

Otherwise the Deploy-LabManager.ps1 script should be selected (from the Scripts directory under the common deployment directory included in the workspace.

Lab Manager Script Parameters:

|  |  |  |
| --- | --- | --- |
| Parameter Switch | Example Value |  |
| -LibraryConfigName | 'Preprod\_Denali\_New' | The name of the LabManager rig from the library's collection that will be used as the template for the new rig. |
| -RigName | '$BuildNumber' | This won't need to change. |

9. Release Notes

TBC

# Queue Build

Right Click Build definition and click "Queue New Build...".

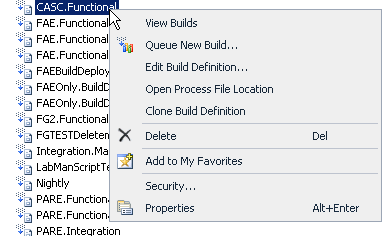
Following a Successful build the deployment log is also available in the TFS Drop folder associated with the build.

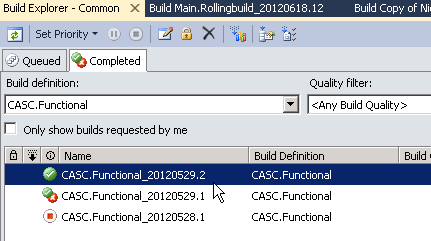
## Viewing Build Details

To view build details:-

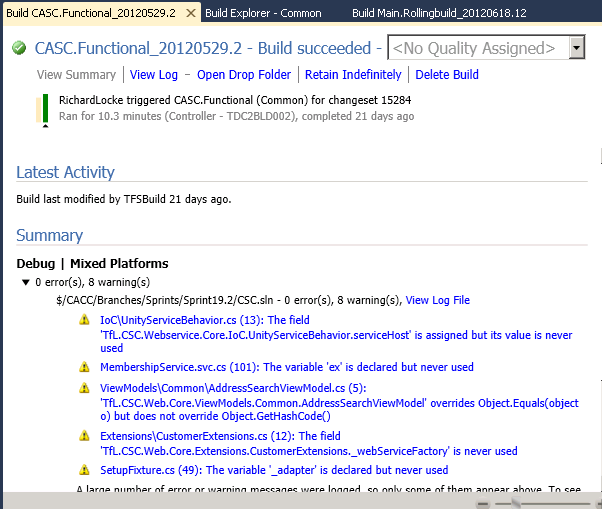
In Visual Studio

In the Team Explorer window

Right Click on the Build Definition, select View Builds  


Locate Build in the Build Explorer  


Double Click on the build

The Build Details windows should be displayed  


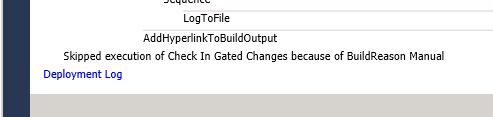
## Deployment Logs

Build output folders deployment log file;

In the Build Details window

Click on the View Log link

Scroll to the bottom of the page

Click on the Deployment Log link  


The powershell build scripts write log files locally on the deployment machine . Because the layered nature of the scripts and the fact that transcription doesn't work properly three files are created;

[D:\Deploy\]deploy-rig.<date-time>.log

[D:\Deploy\]deploy-rig. <date-time>.log

[D:\Deploy]\deploy-roles. <date-time>.log

Additionally, msi installers write their logs to the software's source dropfolder;

[D:\ ...\Deploy\Software\DropFolder\logs\

## Move to Nightly Build

Upon a successful build a new LabManager rig will be automatically un-deployed and moved to the workspace specified in the Lab Manager section of the build definition.

If a rig has been redeployed or retested and needs to be moved manually follow these steps;

In Lab Manager:-

Find the configuration in the **TFSWorkspace** workspace

Hover over the configuration until the menu appears

Select Shut Down

Once the configuration had shut down (all configuration consoles should be blank)

Hover over the configuration until the menu appears

Select Undeploy

Once the configuration is undeployed, Move the Configuration to the NightlyBuild Workspace

# Rig Testing

## Deployment Verification Testing

The deployment tool which the build uses to run the post-deployment tests can be executed manually.

<dropfolder>\Deployment\Tools\Deployment Tool\DeploymentTool.exe

Arguments:

-RigRelativeConfig the name of the

## Automated Smoke Testing

## Manual Smoke Testing

Following a successful build and deployment a new rig will be available in Lab Manager in the **TFSWorkspace** workspace.

In order to access the Rig ask an administrator to make you the owner of the Rig in Lab Manager.

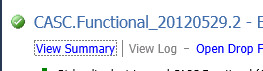
Once you have access you should run through some smoke tests to ensure the deployment was successful.

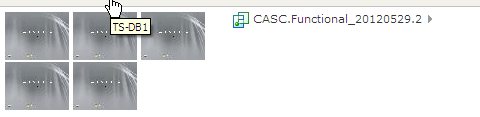
### Rig Naming

By default the deployed Rigs will be named using the TFS Build Number ($BuildNumber).

This is of the format "<BuildDefinitionName>\_yyyyMMdd.*n*". For example;

**In Team Foundation**



**In Lab Manager**

# Common Errors

## From Build Output

1. X is not in build workspace

Either something in the Build items list or the Test Data Server Folder is outside of the paths specified in the Workspace section.

2. Build Targets ... Object not set to a reference of an object...

Periodic build error, just restart the build.

3. Lab Manager cannot find rig

The rig owner needs to be TFSBuild in order to be seen by the build process.

The name specified in the command arguments is wrong (check for spaces at start/end of string)

4. The Directory is not empty.

Try deleting the Build Servers build dir TDC2BLD00x\D$\Builds\<sprint>\<team>\<build\_definition\_name>\\*

## From Setup-Deployment

4. Time Discrepancy between servers on file copy

Redeploy rig OR

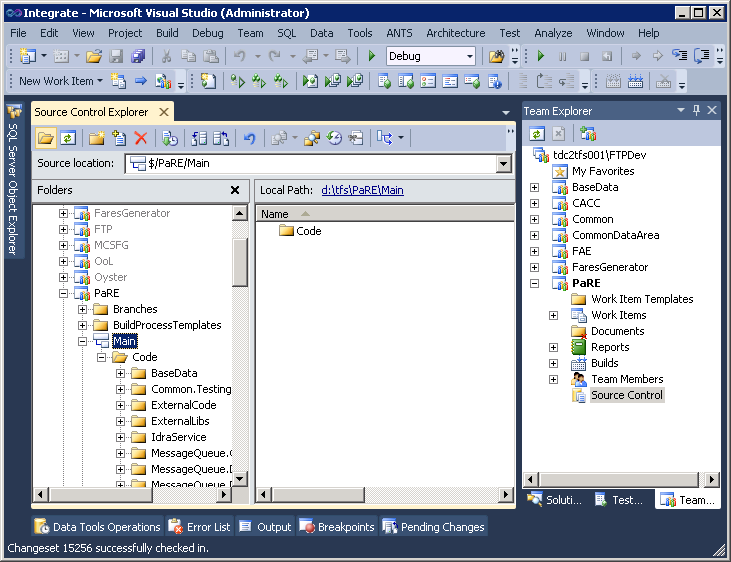
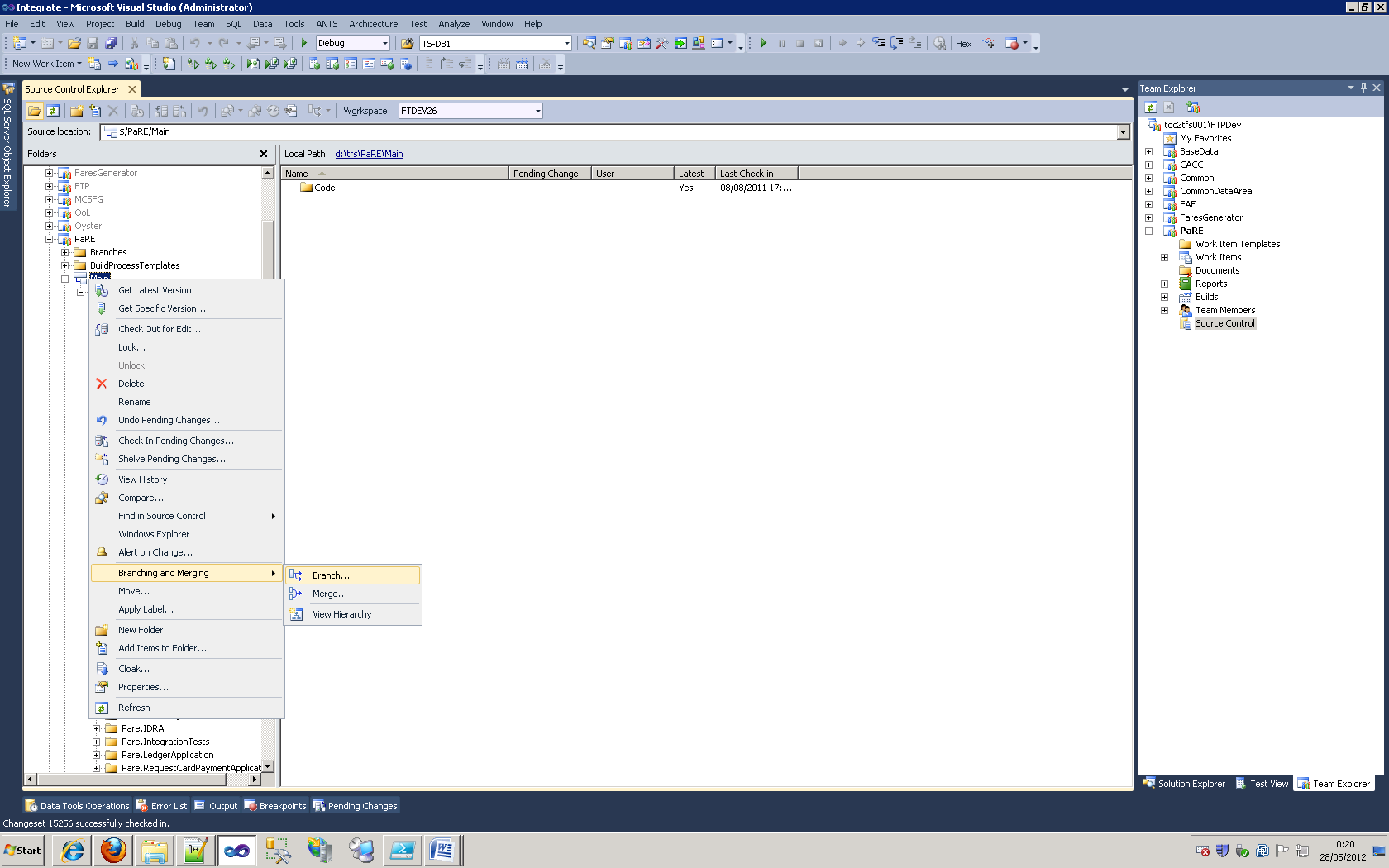
execute 'w32tm /resync' on each server and reset.

5.

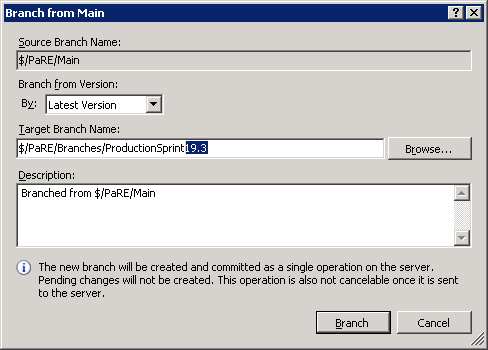
# Creating Code Branches

A distinct branch of the code should be used for all functional deployments to ensure it is possible to go back to the exact code base that a build implements.

1: In the Source Control Explorer for the project to be branched, right click the main branch being forked and select 'Branch...' from the 'Branching and Merging' option in the context sensitive menu.



2: Select the highest branch version in the Target Branch Name and rename it upwards by .1 if it's within the same sprint or to N+1.0 if it is the first release for a new sprint and click 'Branch'



3: Update the VersionSeed.xml file. This is, typically, stored at in the solution root and should have the major and minor versions updated to reflect that selected for the new branch name.