Use TFS for Gated Checkin and Code Review

# Goal

We want to ensure code can only be checked in when the following criteria are met:

1. Code review: code need to be reviewed by peers
2. Styles must satisfy predefined rules (i.e. FxCop)
3. Unit tests: affected unit tests should still pass
4. Code coverage: code coverage need to be above predefined percentage

## Build Process

When gated checkin is enabled in build definition, TFS automatically creates shelveset and ensure solution is successfully compiled and all tests are passed before shelveset can be checked in. However, TFS do not enforce code style rules and code coverage rules. In order to enforce code style rules and code coverage rules are applied, TFS build needs to be customized.

## Code Review

TFS supports code review by creating two work items (code review request, code review response). Code review process (out of the box) is optional and cannot be easily changed within TFS. Another limitation is that TFS does not provide customization of code review policy, for example, if we want to enable certain rules for code review process (i.e. committer cannot review his/her own code; only a few chosen peers can be code reviewers, etc), we need to write plugin code and deploy it to TFS.

## Acknowledgement

Associated code are downloaded and modified from the following resources:

* TFS Plugins (<http://tfspluginsuite.codeplex.com>)
* TFS Build Extensions (<http://tfsbuildextensions.codeplex.com>)

# Enable Checkin Policies with TFS Plugin

## What is TFS Plugin

TFS plugin allows user to override decision when certain events (i.e. code checkin) happens. This approach requires user to deploy plugin assembly to TFS server under “C:\Program Files\Microsoft Team Foundation Server 12.0\Application Tier\Web Services\bin\Plugins”. TFS service will pick up changes automatically and hookup event handlers defined in assembly.

## TFS Plugin Configuration

This approach can only be applied to TFS on premise, TFS online won’t work.

Policies defined using this approach cannot be overwritten from client (this is different from code checkin policies defined from Visual Studio where user can override with a reason). Therefore, we should carefully define which policy is enabled under with team project. Here is an example of minimal configuration (see Figure 1), policies are disabled by default. As you can see in Figure 2, code review policy configuration allows you specify persons qualified to perform code review, and you can allow same person to be reviewer and committer for the checkin.



Figure : Four policies are enabled for team project "CI"

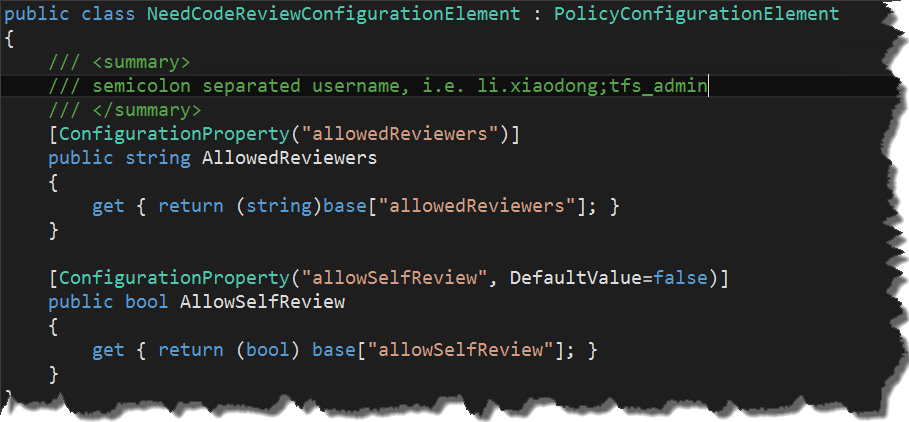


Figure : code review policy configuration

## TFS Plugin Deployment

To deploy custom TFS plugin, drop two files to TFS server under plugin folder (“C:\Program Files\Microsoft Team Foundation Server 12.0\Application Tier\Web Services\bin\Plugins”). To rollback, delete these two files from plugin folder.

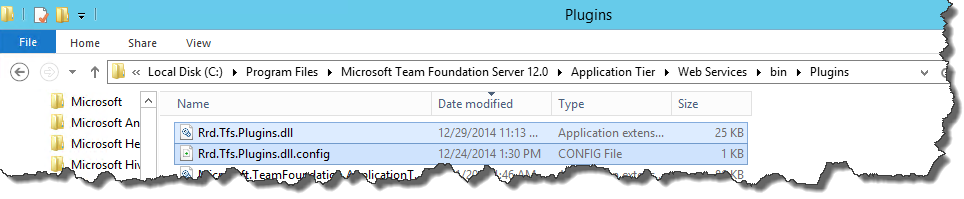
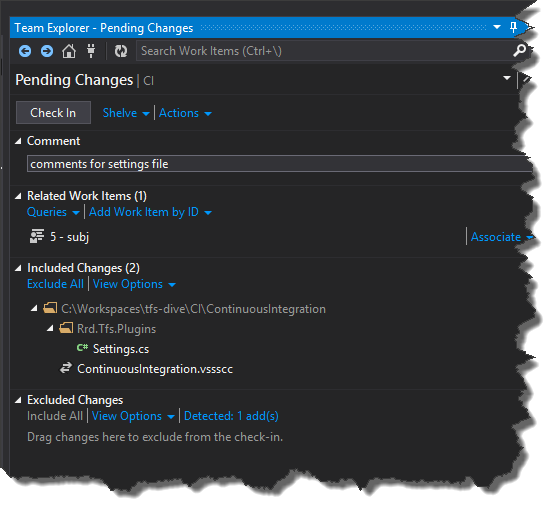


Figure : two files needed for plugin deployment

## Scenario: Code Review

In this scenario, we walk through the process of enforcing code review.

1. Visual studio connected to team project “CI”, Changed file “Settings.cs”.
2. From “Pending Changes” pane, click “Request Review” from “Actions” dropdown menu.



1. Pick user “tfs\_admin” for reviewer and click “Submit Request”.

Figure : request code review

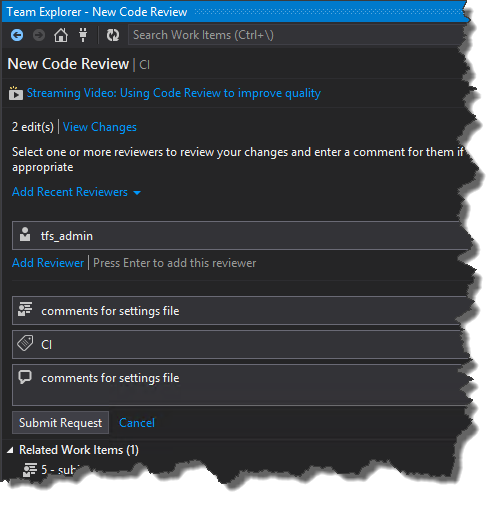


Figure : pick code reviewer

1. Two work items (#25 and #26) are created for code review (see Figure 6).

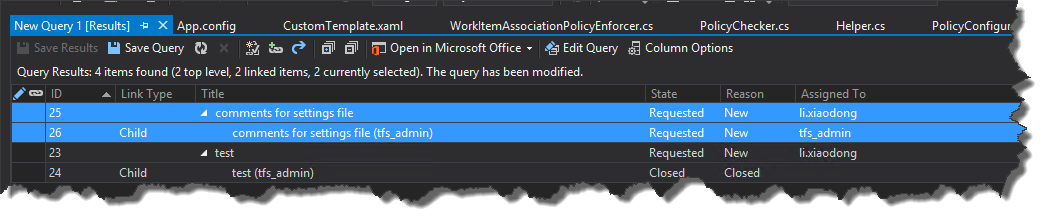


Figure : two work items are created for code review

1. From another server, connect to TFS using user “tfs\_admin”, here is what’s shown within “my work” pane (see Figure 7)

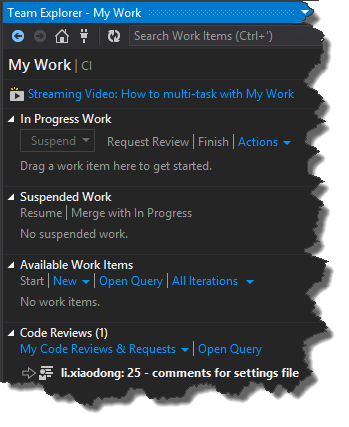


Figure : code request shown under my work

1. Open code review request, and click link “accept” to start code review

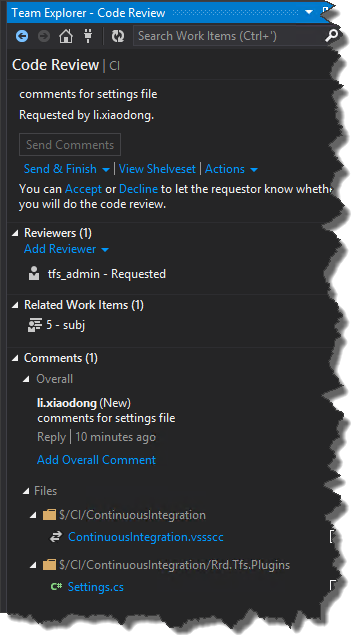


Figure : accept code review request

1. After enter comment, user “tfs\_admin” clicks “Looks Good” from “Send & Finish” dropdown menu.

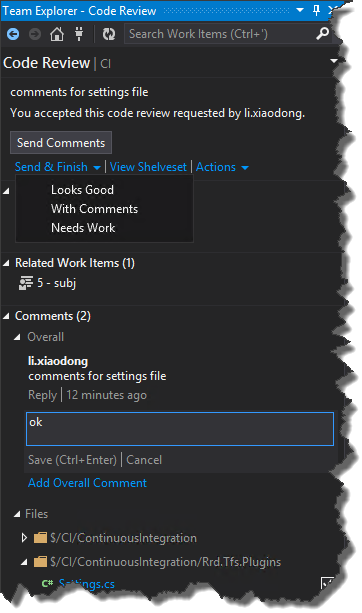


Figure : approve code review

1. Here we can see code review work item response is closed

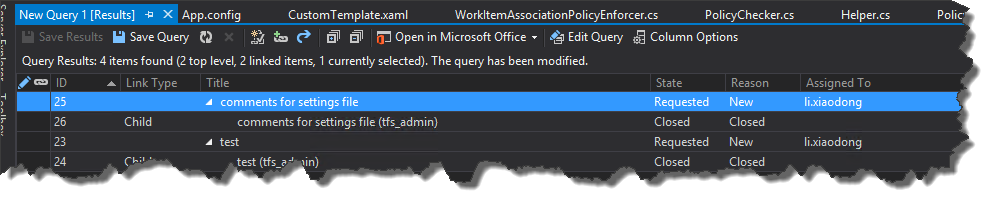


Figure : code review response closed

1. When user “li.xiaodong” looks at code review (open from my work pane), user click link “Complete” from “Close Review” dropdown menu.

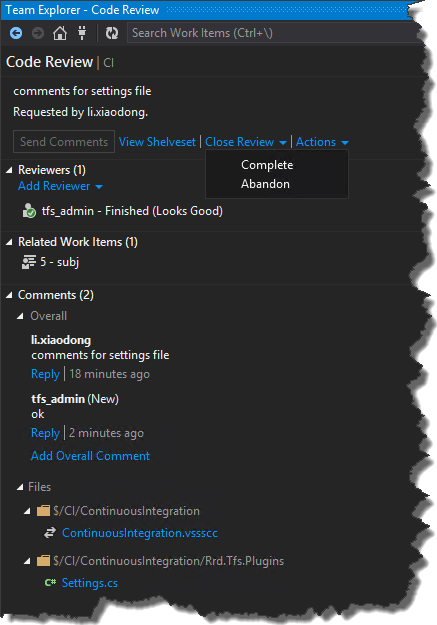


Figure : complete code review

1. User can now check in pending changes.

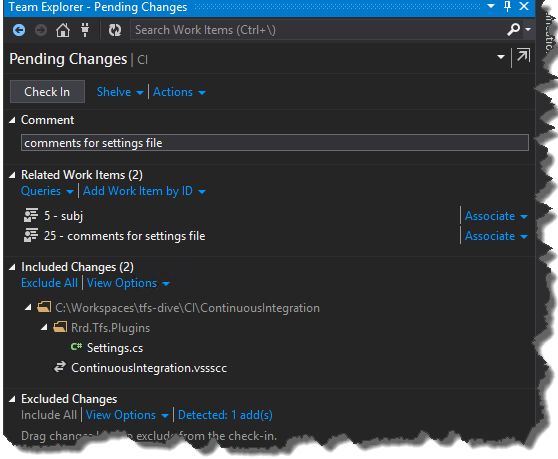


Figure : check in

1. After check in policy validates, you should see a popup window

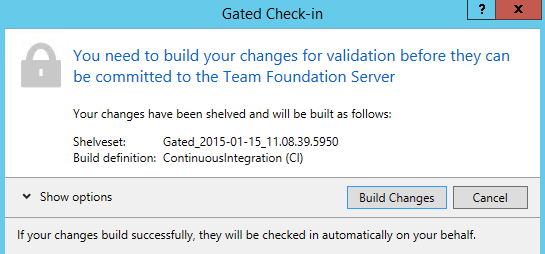


Figure : shelveset created automatically

1. You are notified a build process is triggered by code check in

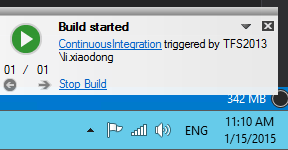


Figure : build started

1. We have customized build definition (see in next section) and enabled code style check and code coverage (>=75%). Since we don’t have any unit tests, build is set to fail.

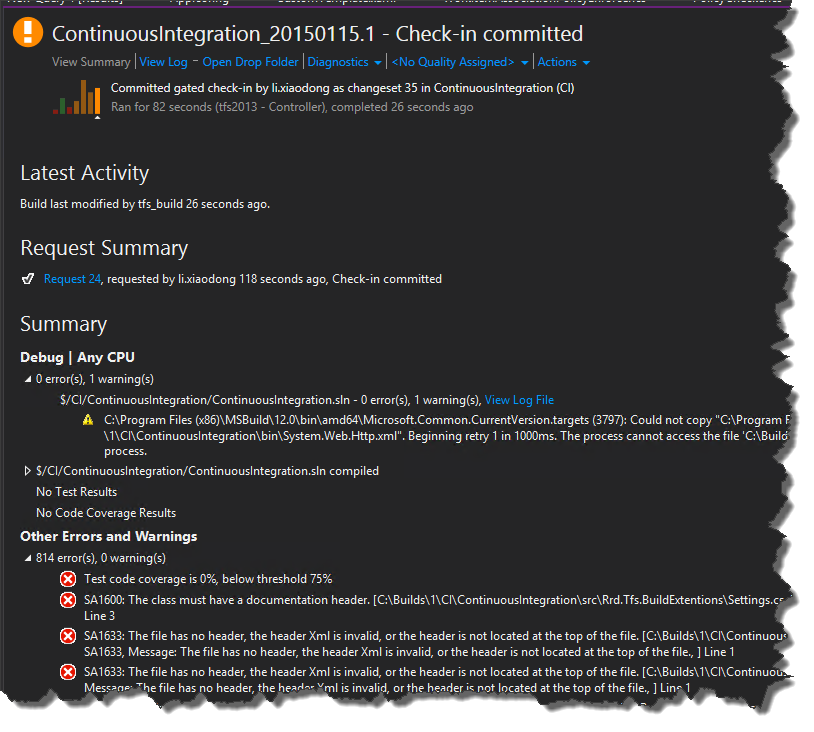
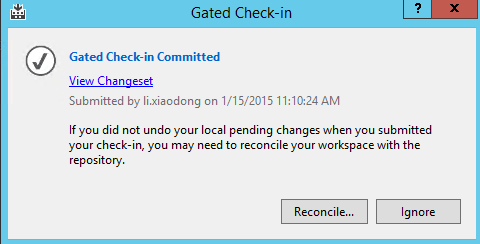


Figure : build failed because of style rules and code coverage

1. When we use default build definition (removed style check and code coverage check), shelveset is automatically checked in when build succeeds.



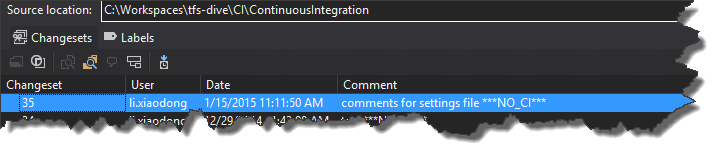


Figure : shelveset is checked in after build succeeds

1. Error messages for code review
   1. When user didn’t request code review and check in code, check in is blocked with message “Check in should have code review requested”
   2. When user requested code review but code review is not finished or accepted or rejected, check in is blocked with message “Associated code review should be accepted and approved before check-in”
   3. When specified code reviewer is not in specified code reviewers list, check in is blocked with message “Code reviewer is not allowed”
   4. When requested code reviewer is same as committer, check in is blocked with message “You cannot assign code reviewer to yourself”

# TFS Build Definition

## Enable Gated Checkin

Gated checkin is enabled within build definition (see Figure 16)

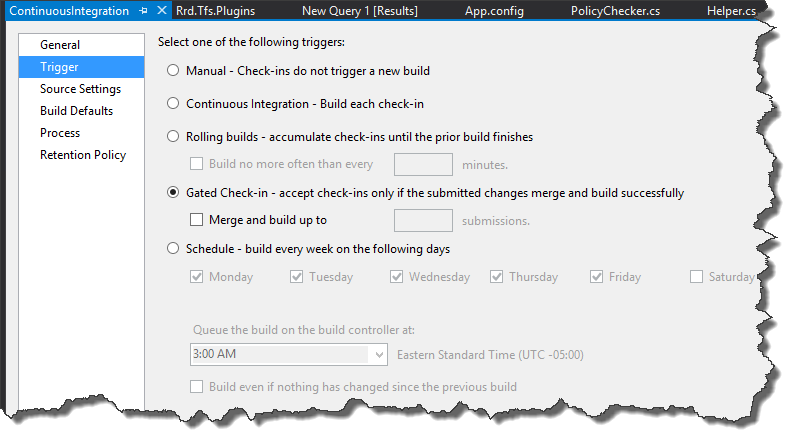


Figure : enable gated check in

## Build Process

We can customize build process (workflow) and add new activities to support retrieve test pass percentage, code coverage percentage, and style check results. New activities are from custom assemblies that must be checked into TFS first.

## Custom Build Activity

We are using two new activities in our build definition: 1) get code coverage; 2) check code styles.

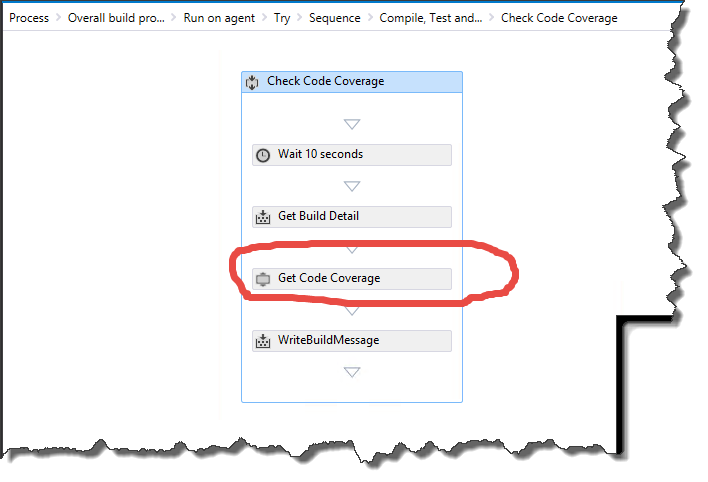


Figure : get code coverage

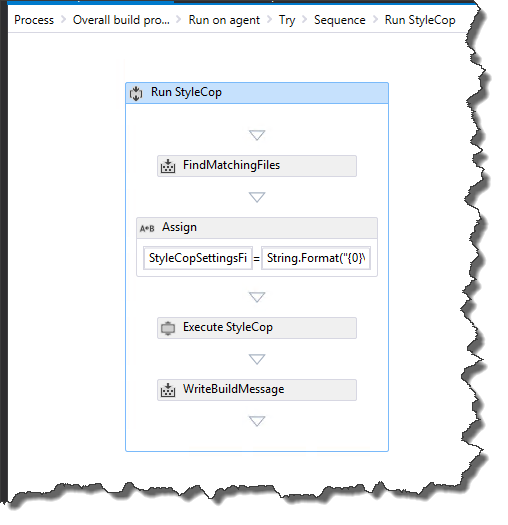


Figure : check code style

## Build Process Deployment

Build process file (xaml) and assemblies (dll) for workflow activities must be checked into TFS. Once checked in, we can pick new process from build definition (see Figure 20), this step can be done by any developer who have contribute write.

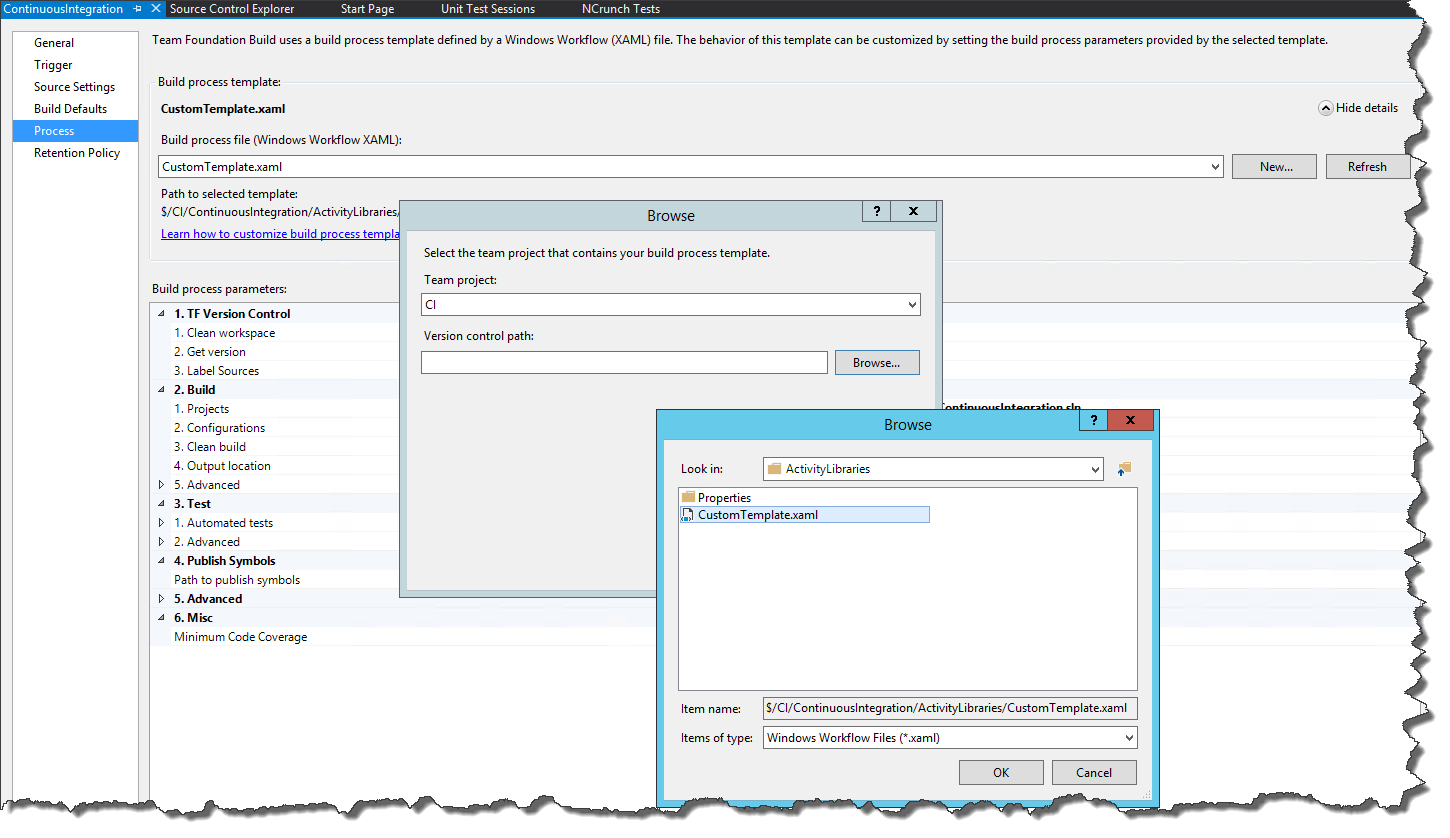
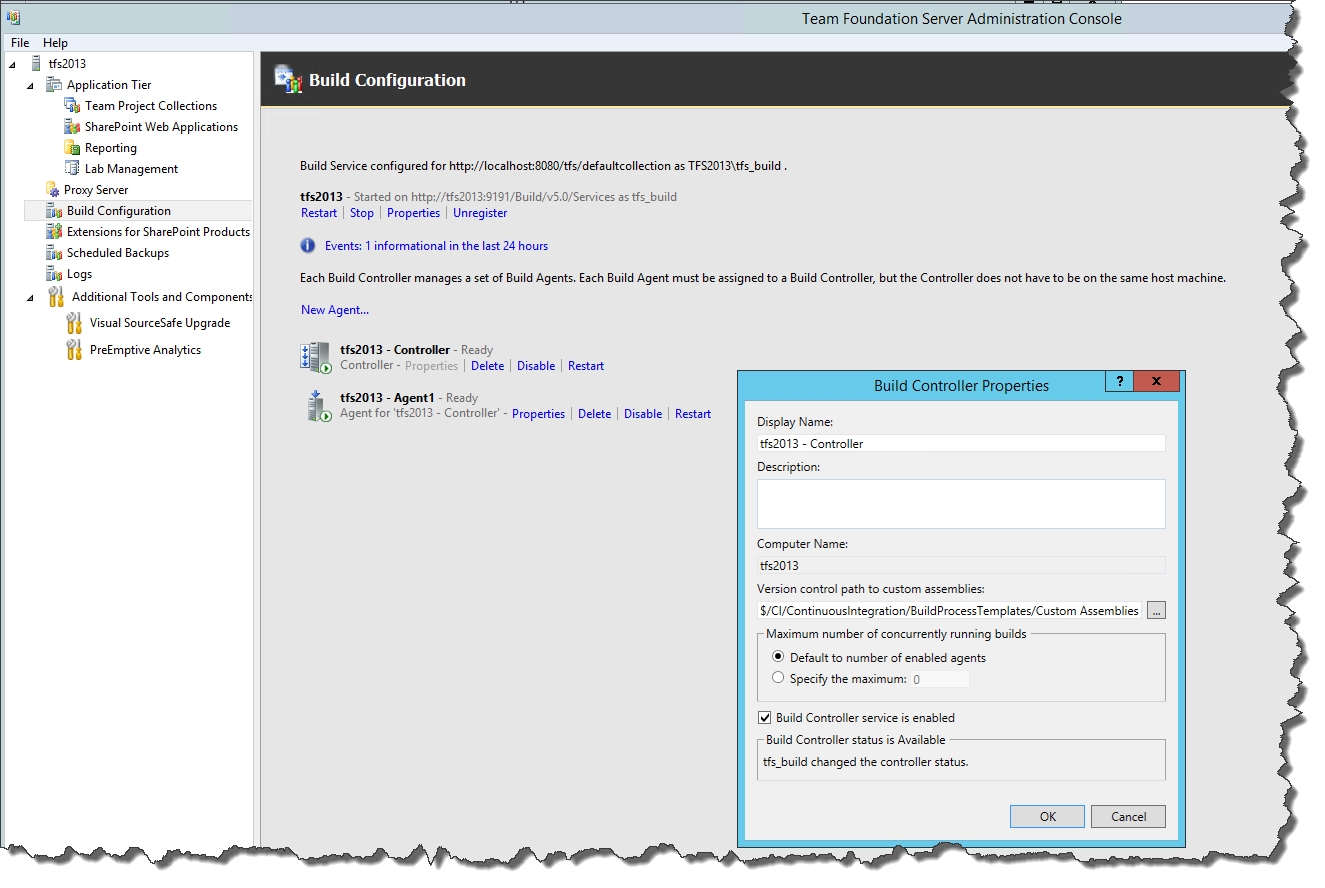


Figure : pick new build process template

## Build Activity Deployment

This step is done once by person who has access to TFS physical server. He/she will need to click build controller property, then points “version control to custom assembly” to TFS path where workflow activity assemblies are located.



## Scenarios

### Enforce Unit Tests

### Integration with Pester and PowerShell Tests

### Enforce Code Styles

TFS support code style check (Microsoft code rules) out of the box when static code analysis is enabled in build definition.

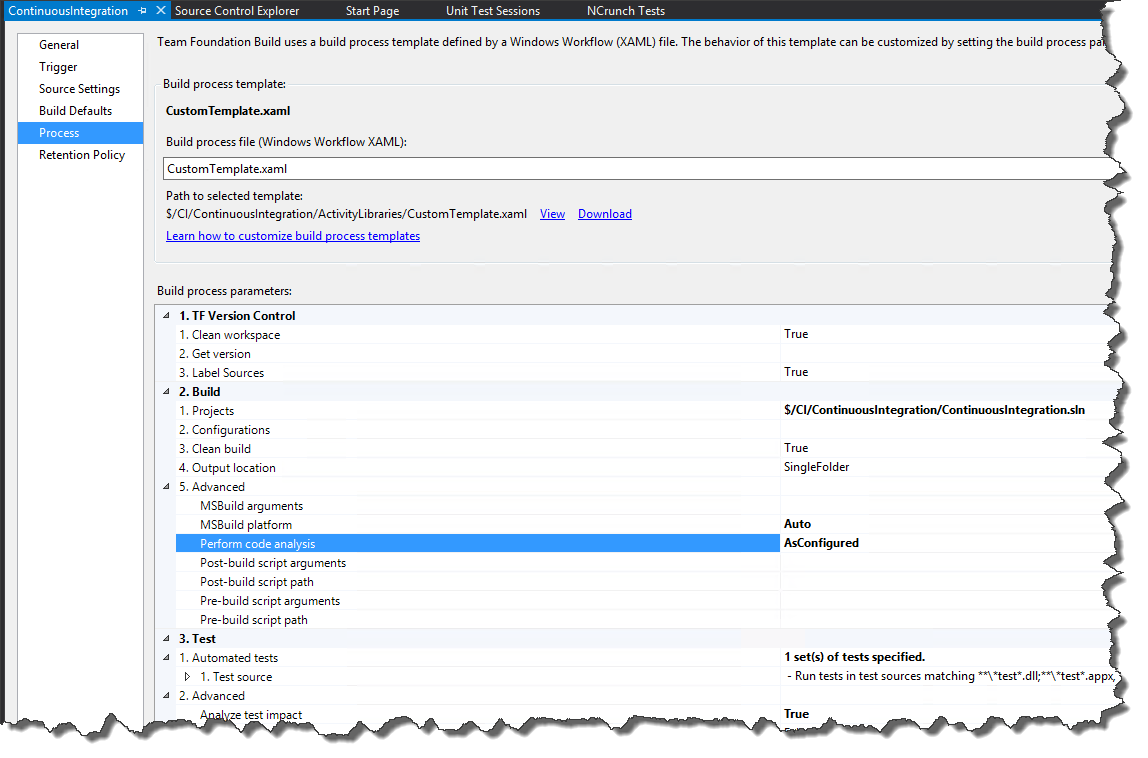


Figure : code analysis within build definition

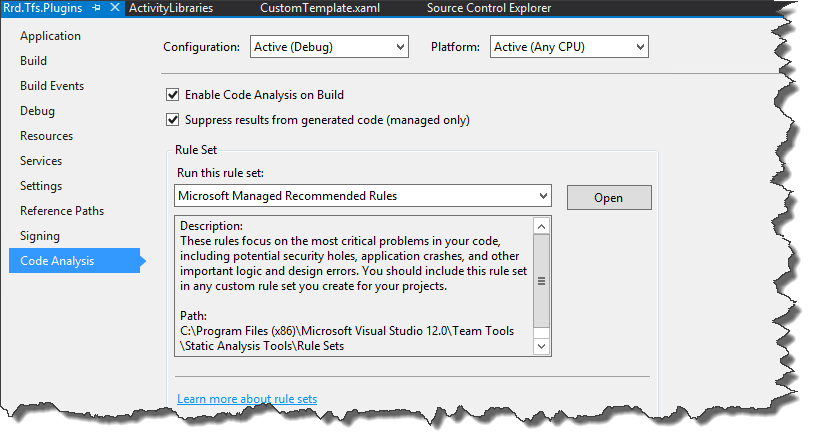


Figure : turn on code analysis at project

### Enforce Code Coverage

Minimum code coverage is provided as workflow parameter (>=75%, see Figure 21). When code coverage is less than specified percentage, the process sets build result to fail to prevent gated check in from committing changes.

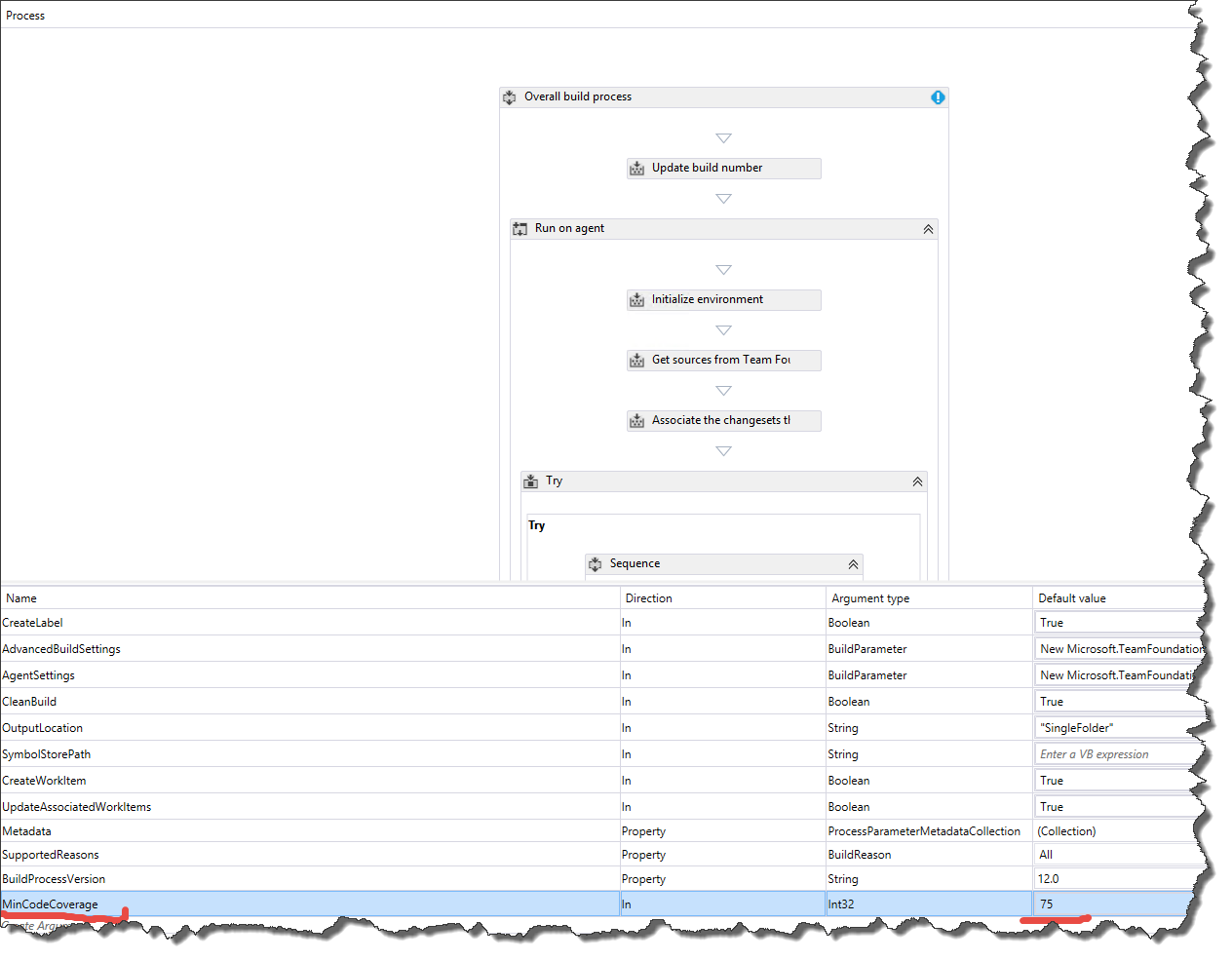


Figure : build definition parameter for code coverage

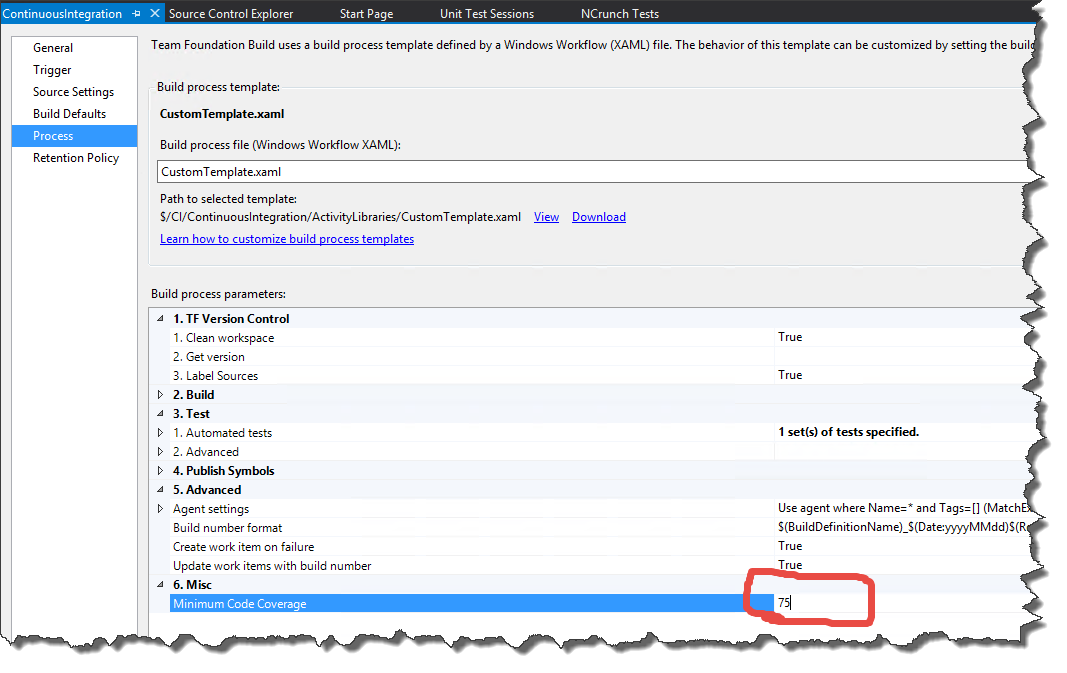


Figure : build definition argument