

# AIT Build Suite 2010

---

*V2.1.1112.1700*

## Copyright

The tool AIT Build Suite 2010 is provided by AIT Applied Information Technologies AG, Leitzstr. 45, 70469 Stuttgart, Germany.

The content of this document is the intellectual property of the AIT AG. © 2010 AIT AG

## Version History

Version	Comments	Date	Author
2.0.1001.1900	Initial version	19.01.2010	Sven Hubert
2.0.1002.0300	Added Sandcastle documentation	03.02.2010	Thorsten Dralle
2.0.1002.0500	Updated process flow	05.02.2010	Sven Hubert
2.0.1004.2600	RTM release	26.04.2010	Sven Hubert
2.0.1010.3100	Intermediate release	31.10.2010	Sven Hubert
2.1.1104.2200	RTM release with build tracing	22.04.2011	Sven Hubert
2.1.1112.1700	RTM release new versioning mechanism	17.12.2011	Sven Hubert

## Contents

Copyright .....	1
Version History .....	1
Feature Overview .....	3
Release Management.....	3
Versioning.....	4
Upgrade .....	5
Code Documentation .....	5
Tracing.....	5
Deployment.....	5
AIT Build Process Template .....	6
Standard Activities.....	7
Custom Activities.....	7
MSBuild Extension Points.....	7
Custom script.....	8
Extending an Extension Point by additional properties .....	9
Internals of an Extension Point .....	10
Branch structure.....	10
Limitations .....	10
Release Management.....	11
Work Item and Changeset Association .....	11
Versioning.....	11
Sandcastle.....	13
Installation.....	13
Create a SHFB Project.....	13
Configuration of the Build Process.....	14
Appendix – Workflow details .....	16
This document was created by.....	17

## Feature Overview

The AIT Build Suite 2010 helps you simplifying complex tasks during your build process. It enables a better versioning and release management. It also helps with documentation using Sandcastle. Additionally legacy MSBuild scripts can be integrated into the process. The update template delivered by Microsoft is obsolete using the AIT build process template.

A significant amount of project effort, especially at bigger customers such as Nero AG, goes into migrating and optimizing build processes. Build processes exhibit 3 important aspects, which due to their criticality and benefits legitimate project costs:

1. Build processes are implicitly business critical – only a working build process results in deliverable products.
2. Heterogeneous build processes imply high – mostly hidden maintenance efforts without directly visible benefits. Small changes in “the scripts” are long lasting and expensive tasks which can be done by high qualified resources only.
3. Build processes often require manual steps, which can be automated with modern platform. Which saves costs!

With the new AIT Build Suite 2010 complex build processes can be configured easily using centralized builds with Microsoft Visual Studio Team Foundation Server 2010.

## Release Management

An important part of the Build Suite is release management. The standard process associates code changes (Changesets) and tasks (Work Items) using the build status. All changes that happened on a specific branch since the last successful build will be associated with the current one (see figure 1).

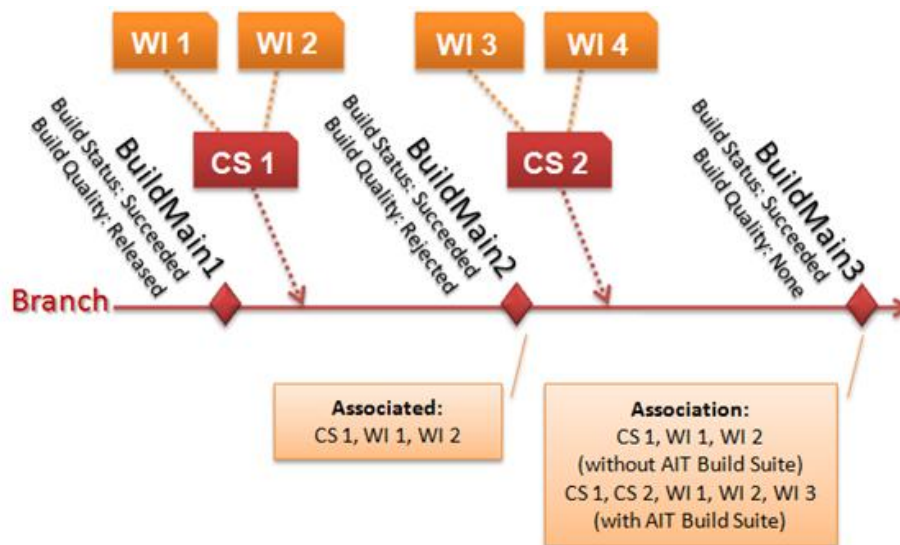


Figure 1 - Build association of Changesets and Work Items

But that's not sufficient. Figure 1 shows the changed behavior using the AIT Build Suite 2010. It additionally instruments the build quality (see figure 2) to generate change lists. With this functionality, all changes back to a released product version can be tracked easily.

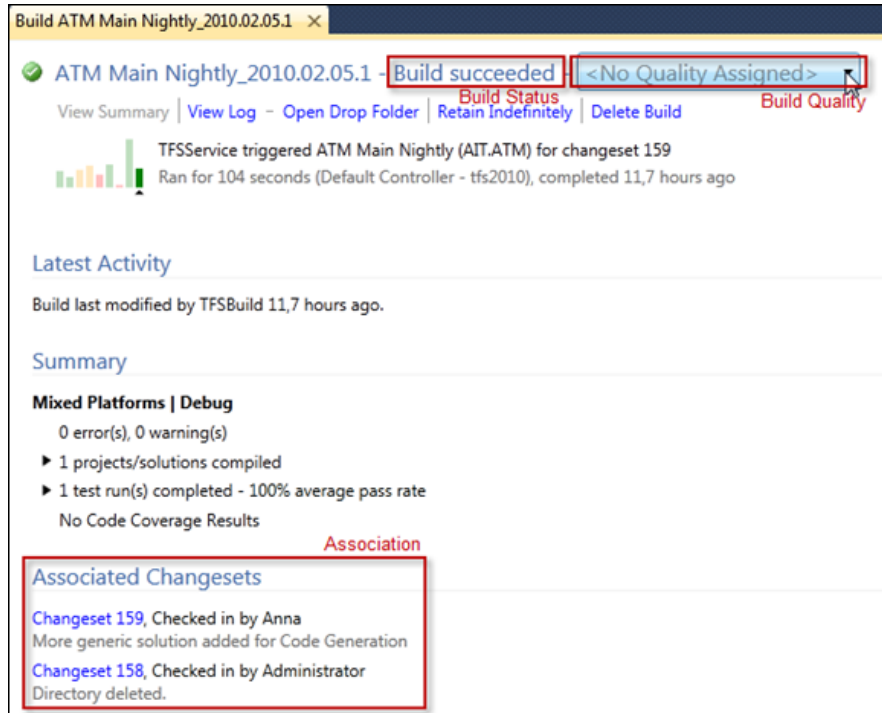


Figure 2 - Build summary

## Versioning

During the creation of product versions it is important to integrate versioning information into the product itself. With the first call of a customer caused by questions or issues, versioning information about the product must be easily accessible for end-users. .NET framework provides options to keep the version number and other details inside an assembly and to show those inside an about box.

AIT Build Suite provides an easy way to integrate this task into the build process and includes an example of how to show these information in your application (see figure 3).

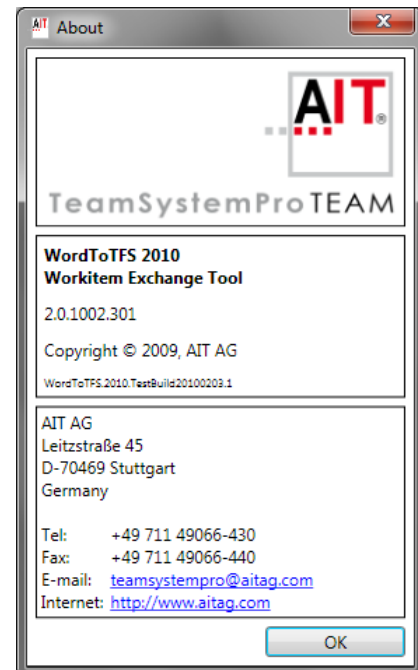


Figure 3 - Assembly information are shown in an about dialog

## Upgrade

With Team Foundation Server 2008 build processes have been executed using a central MSBuild script deployed with Team Explorer or Team Build components. With the new version 2010 Workflow Foundation is used instead. Still, at the core of compilation, MSBuild is used (new version 4.0). Existing build scripts cannot be integrated out-of-the-box with 2010.

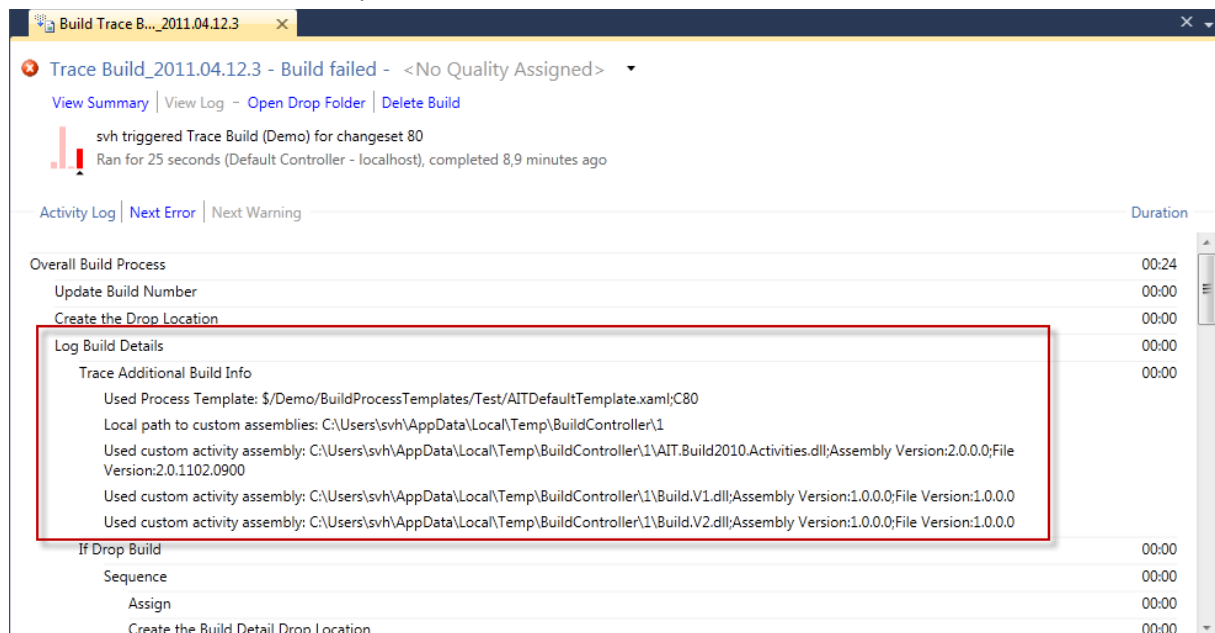
The Microsoft UpgradeTemplate ought to provide an easy to integrate experience. But even for basic customized scripts it is not sufficient. Despite the core compile targets, none of the customizable extension targets like AfterGet or BeforeCompile are instrumented. That's why, AIT delivers a generic Template with Build Suite 2010 which provides MSBuild extension points. This accelerates the migration of build to TFS 2010 dramatically.

## Code Documentation

Customers of AIT use our Sandcastle integration for Team Build 2008. Sandcastle projects can be used to automatically generate code documentation. This can be integrated into the central build as well. The new AIT Build Suite 2010 provides parameterized activities for Sandcastle as well.

## Tracing

Per default, the AIT Process Templates contain activities that trace the necessary build information during the build process and drop the process template as well as custom parameters of the build definition into the build's drop location.



Activity Log	Duration
Overall Build Process	00:24
Update Build Number	00:00
Create the Drop Location	00:00
Log Build Details	00:00
Trace Additional Build Info	00:00
Used Process Template: \$/Demo/BuildProcessTemplates/Test/AITDefaultTemplate.xaml;C80	
Local path to custom assemblies: C:\Users\svh\AppData\Local\Temp\BuildController\1	
Used custom activity assembly: C:\Users\svh\AppData\Local\Temp\BuildController\1\AIT.Build2010.Activities.dll; Assembly Version:2.0.0.0; File Version:2.0.1102.0900	
Used custom activity assembly: C:\Users\svh\AppData\Local\Temp\BuildController\1\Build.V1.dll; Assembly Version:1.0.0.0; File Version:1.0.0.0	
Used custom activity assembly: C:\Users\svh\AppData\Local\Temp\BuildController\1\Build.V2.dll; Assembly Version:1.0.0.0; File Version:1.0.0.0	
If Drop Build	00:00
Sequence	00:00
Assign	00:00
Create the Build Detail Drop Location	00:00

Figure 4 - Build trace in summary log

## Deployment

The AIT Build Suite 2010 consists of a build process template (an Xaml-file which contains the basic workflow using Windows Workflow Foundation) and a library with custom Workflow activities which are used in the build process template. The build process template can be deployed to \$/TeamProject/BuildProcessTemplates/AITDefaultTemplate.xaml. The additional activity library has

to be deployed using the TF Build deployment features. Just put AIT.Build2010.Activities.dll under source control and specify the folder in the build controller properties:

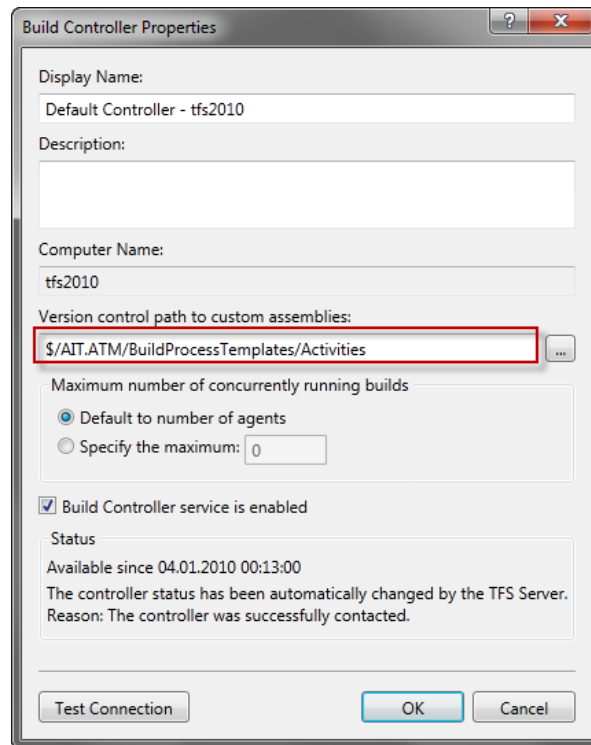


Figure 5 - Build controller properties with path to custom assemblies

In order to edit the process AITDefaultTemplate xaml file, you have to copy the AIT.BuildSuite.Activities dll to your local PrivateAssemblies folder of Visual Studio under:

C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\PrivateAssemblies (64 bit) or

C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\PrivateAssemblies (32 bit).

## AIT Build Process Template

The following figure shows the basic flow of build activities as specified in the build process template.

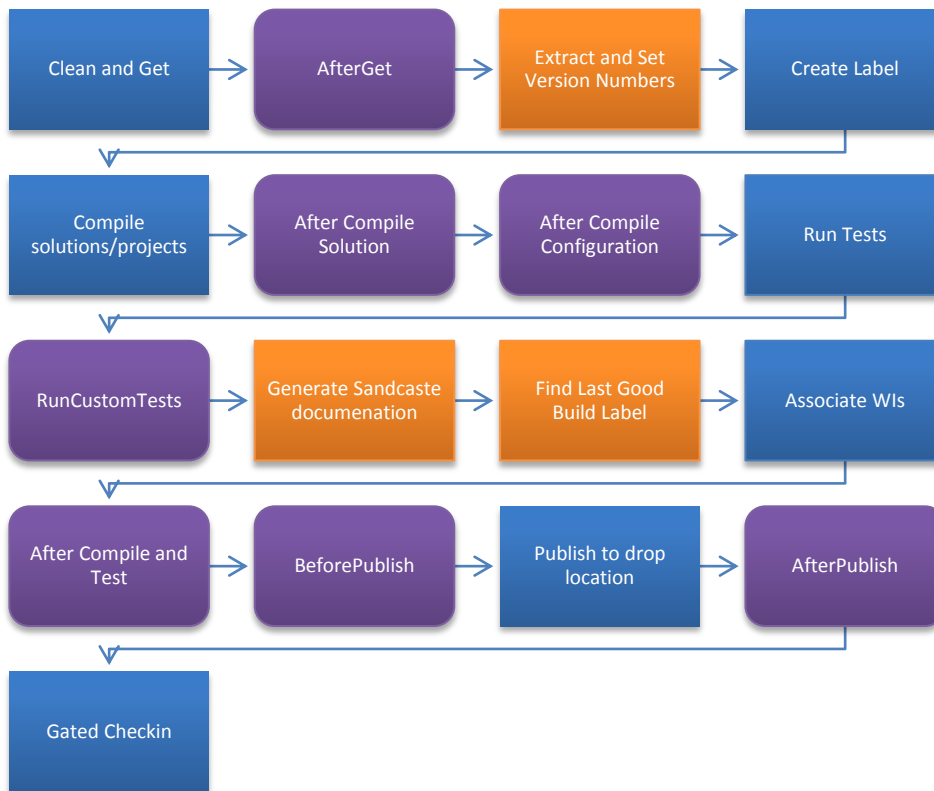


Figure 6 – Basic Build Process



Figure 7 – Legend

## Standard Activities

Standard activities are provided by Microsoft and reside inside the assembly Microsoft.TeamFoundation.Build.Workflow.dll which is deployed when installing Team Explorer or Team Foundation Build components.

## Custom Activities

Custom activities are implemented Code Activities using Windows Workflow Foundation 4.0. They can contain any logic to execute during build.

## MSBuild Extension Points

Extension points can be used to inject custom MSBuild targets into the flow. These can be configured using the properties of a build definition (Process tab).

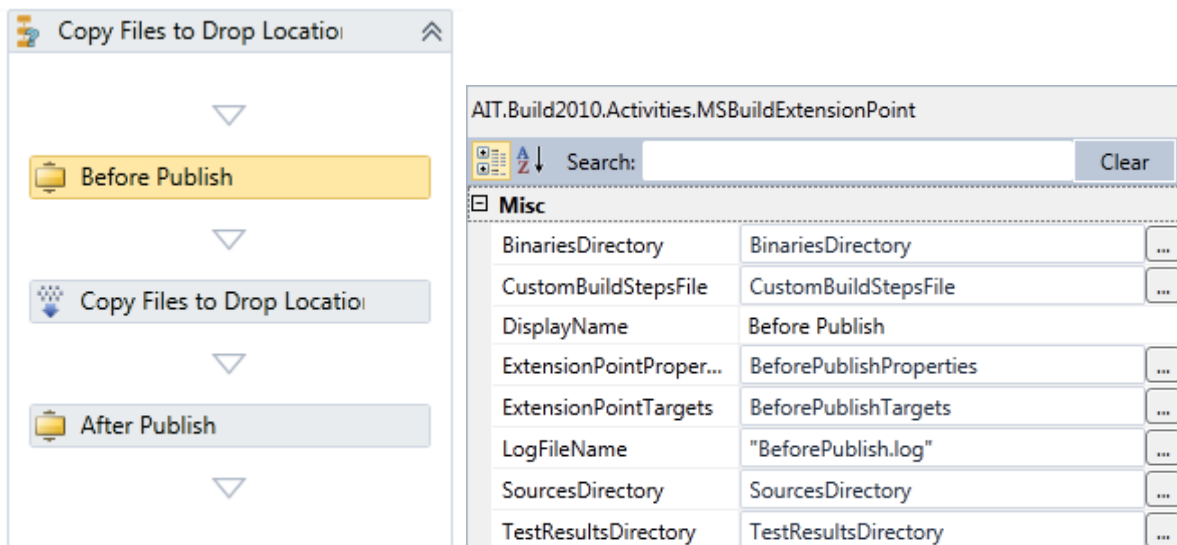


Figure 8 - Sample Extension Point with default initialization using global build arguments

Properties of an extension point:

**CustomBuildStepsFile** – the MSBuild script file which contains the targets to execute (e.g. ComponentName\Build\CustomBuildSteps.targets)

**ExtensionPointProperties** – a string with a semicolon-separated list of custom key-value properties which are available inside the MSBuild script (e.g. "Property1=Value1;Property2=Value2" can be referenced as \$(Property1) and \$(Property2) inside the MSBuild script called by this activity)

**ExtensionPointTargets** – an ordered list of target names to execute (defined in MSBuild script as "`<Target Name="TargetName">`")

**LogFileName** – name of the log file that will be created inside the LogLocation of the build  
The directory parameters are injected into the build by default using the /p command line argument when calling MSBuild.

## Custom script

The MSBuild script CustomBuildSteps.targets contains branch-specific build steps. Team-specific build steps should be held above the branch. This causes an additional entry in the build workspace to download the script during GetWorkspace activity.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- DO NOT EDIT the project element - the ToolsVersion specified here does not prevent the solutions
and projects in the SolutionToBuild item group from targeting other versions of the .NET
framework.
-->
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003" ToolsVersion="3.5">
  <PropertyGroup>
    <!-- Url of the TFS app-tier. -->
    <TeamFoundationServerUrl
      Condition="'$(TeamFoundationServerUrl)' == ''>$(TFSUr1)</TeamFoundationServerUrl>

    <!-- The working directory where Components.targets lies (normally the solution dir) -->
    <WorkingDir
      Condition="'$(WorkingDir)' == ''>$(MSBuildStartupDirectory)</WorkingDir>

    <!-- Used to determine whether we are in desktop build or in central build -->
    <IsDesktopBuild
```

```

        Condition="'$(IsDesktopBuild)' == ''>True</IsDesktopBuild>
</PropertyGroup>

<Target Name="AfterGet">
  <Message Text="AfterGet called." />

  <!-- System properties -->
  <Message Text="TeamFoundationServerUrl:[$(TeamFoundationServerUrl)]" />
  <Message Text="BuildUri:[$(BuildUri)]" />

  <!-- Custom properties - specified in build definition properties -->
  <Message Text="PropertyName:[$(PropertyName)]" />

  <!-- Your custom code here -->
  <!-- e.g. command line -->
  <!-- <Exec Command="c:\run.bat" /> -->

  <OnError ExecuteTargets="OnError" />
</Target>

<Target Name="AfterCompile">
  <Message Text="AfterCompile called." />

  <OnError ExecuteTargets="OnError" />
</Target>

<!-- Equivalent for targets RunCustomTests, BeforePublish and AfterPublish -->
<!-- You can also include custom targets -> Specify in build definition properties -->

<Target Name="OnError">
  <Message Text="Error occured!" />
</Target>
</Project>

```

Figure 9 - Custom script sample

## Extending an Extension Point by additional properties

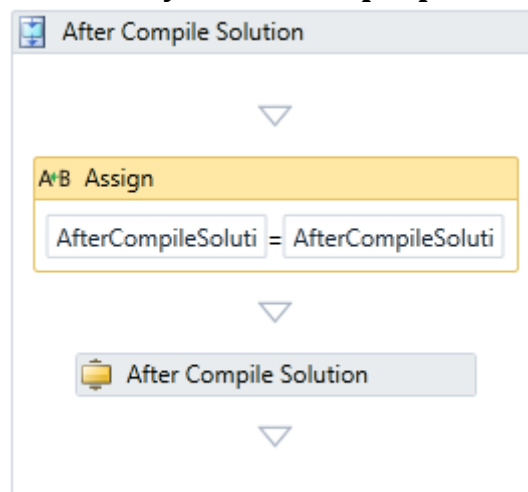


Figure 10 - Sample Extension Point with additional properties assigned

The extension point activity is a sequence. It will first initialize some variables used to configure the call to MSBuild such as Script name to call, Targets to execute, Properties to transfer.

The Properties variable can be set inside the build definition and extended inside the process template like this (see Figure 10):

```
AfterCompileSolutionProperties + ";" + Project + ";" + Platform + ";" +
platformConfiguration.Platform + ";" + Configuration + ";" +
platformConfiguration.Configuration
```

## Internals of an Extension Point

The call to MSBuild is configured to log into a specific log file with the same name as the extension point:

AdditionalVCOverrides	<i>If GenerateVsPropsFile is true, the contents of this string will</i>	...
CommandLineArguments	[Properties]	...
Configuration	<i>The (optional) configuration to be built for the specified proj</i>	...
DisplayName	Execute MSBuild script	
GenerateVSPropsFile	False	...
LogFile	"AfterGet.log"	...
LogFileDropLocation	BuildDetail.DropLocation + "\\logs"	...
MaxProcesses	1	...
OutDir	<i>Specify the directory to which outputs will be redirected.</i>	...
Platform	<i>The (optional) platform to be built for the specified project/s</i>	...
Project	Script	...
ResponseFile	<i>Specify the response file to use.</i>	...
RunCodeAnalysis	<i>Specify whether code analysis should always be run, should</i>	...
Targets	Targets	...
TargetsNotLogged	<i>Specify the targets for which project started events should m</i>	...
ToolPath	<i>Specify the path to the tool. This value is optional.</i>	...
ToolPlatform	Microsoft.TeamFoundation.Build.Workflow.Activities.ToolP	...
Verbosity	Microsoft.TeamFoundation.Build.Workflow.BuildVerbosity.l	...

Figure 11 - Default Extension Point Activity Property

Since this activity is encapsulated in the MSBuildExtensionPoint activity inside AIT Build 2010 activities dll, you don't have to create it yourself.

## Branch structure

The branch to build should follow the following folder structure to have support using default values:

```
/ComponentName
  /Build
    CustomBuildSteps.targets – MSBuild script
  /src
    /SolutionName
      SolutionName.sln – Solution to build
```

## Limitations

An extension point for BeforeGet is not available, since the custom script file is part of source control, it has to be downloaded to the build agent during activity Get Workspace. As such, extension points before the Get activity (e.g. BeforeGet) cannot call the custom script file.

## Release Management

5. Release Management	
Explicit Version Info	
Required Build Qualities	String[] Array
[0]	3 - Merged
[1]	3 - Released
Required Build Status	Succeeded
Update Assembly Description	True
Update Assembly Version	True
Update Build and Revision Only	False
Use Build Number as Version Info	True

Figure 12 - Release Management Properties

## Work Item and Changeset Association

The BuildSuite lets you change the default association behavior. The following properties can be set inside your build definition:

Property	Description
<b>Required Build Qualities</b>	The list of qualities used to filter the last good build that will be used for comparison. One of the qualities must apply.
<b>Required Build Status</b>	The status that the filtered build needs to fulfill at least. E.g. Partially Succeeded would also return successful builds.

During the build, the most current previous build is searched that fulfills these criteria. The label of this build – typically the build number – is used to compare the previous with the current build.

We recommend using a specific set of build qualities for your builds. For example the following:

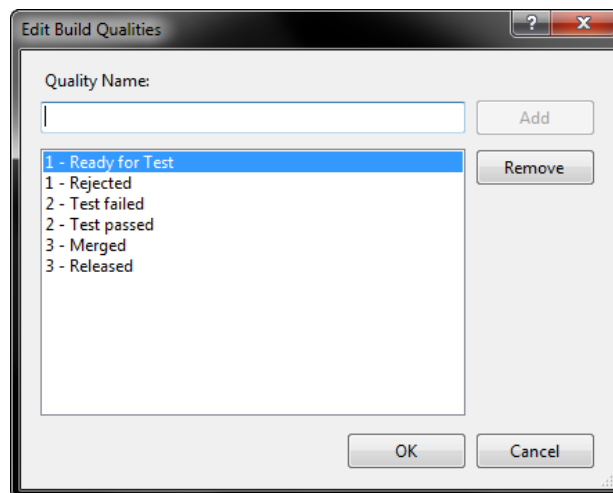


Figure 13 - Custom Build Qualities

## Versioning

During build, all AssemblyInfo files will be updated before compile (not checked in) with version information.

By setting the following properties, you can parameterize the behavior:

Property	Description
<b>Base Year</b>	The base year to calculate the version (method BaseYear) for the third digit (current year – base year is first digit of build version digit) – e.g. for 01.04.2011 this would mean 20401 for base year 2009 (2011-2009)
<b>Explicit Version Info</b>	Set a fixed version string (e.g. “1.10” or “1.10.19.3”) You can set it to a date as YYYY.MM.DD.Rev in order to override the update version methods below (e.g. to reproduce a previous build by using an older version number); you can also set it to major.minor only in order to keep the ReducedDate, BaseYear or ReducedChangeset version number.
<b>Update Assembly Description</b>	Sets the AssemblyDescription attribute in your Assembly to the complete BuildNumber. This way, you can trace an assembly to a distinct build.
<b>Version Update Method</b>	<p><b>None:</b> do not to set a version in AssemblyInfo files</p> <p><b>FullDate:</b> use the full date for all 4 version digits</p> <p><b>ReducedDate:</b> use the date only for the last 2 digits</p> <p><b>BaseYear:</b> use the base year of development for the third digit</p> <p><b>ReducedChangeset:</b> use the <b>SourceGetVersion</b> changeset id and split it at length 4 to fit into build and revision digits.</p>

It is not recommended to change the AssemblyVersion – otherwise you will have to update all .NET assembly references where use distinct version is set to true anytime you built the assembly. You can change this behavior inside the activities only.

In order to use the build number as version info, you have to provide a four digit number as part of it. We recommend the following setting in case you want to use the date for all four version digits (e.g. will result in 2011.03.23.1 or X.Y.1103.231 in case you used build and revision only):

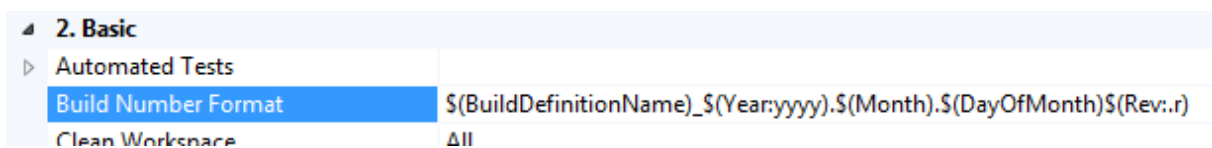


Figure 14 - Custom build number format used to calculate a version number

HINT: You can use a GlobalAssemblyInfo file for your entire solution which is linked to your Visual Studio projects. This way, you can centrally maintain copyright and vendor information.

## Sandcastle

Sandcastle, created by Microsoft, is a tool used for creating MSDN-style documentation from .NET assemblies and their associated XML comments files. The current version is the May 2008 release. It is command line based and has no GUI front-end, project management features, or an automated build process.

## Installation

Certain prerequisites have to be installed on the local system in order to generate the Sandcastle documentation during the build process. Therefore the following steps have to be executed before the actual Sandcastle Activity can be used in the TFS Build Process:

- 1) Install Sandcastle on all Build Servers. (<http://www.codeplex.com/Sandcastle>)
- 2) Install the Sandcastle Help File Builder on all Build Servers (<http://www.codeplex.com/SHFB>)
- 3) Recommended:
  - a. Create a Sandcastle Build Tag and tag the corresponding servers
  - b. Create a SHFB Build Tag and tag the corresponding servers
- 4) Reboot the Build Server

## Create a SHFB Project

Sandcastle Help File Builder was created in order to fill the gaps of Sandcastle as it has been released by Microsoft. The Sandcastle Help File Builder addresses two major gaps:

- 1) Sandcastle Help File Builder offers a rich graphical User interface. This UI can be used to configure all properties that control the features and appearance of the compiled help file. It also offers the opportunity to manage the files that have to be included into the help file itself
- 2) Sandcastle Help File Builder offers an MSBuild integration that can be used to generate the documentation during a build process.

Setting up a SHFB project is very easy and straight forward. It is recommended to create the SHFB project files on the build server in order to avoid path issues. The following guidelines can be used to setup the SHFB project file

- 1) Start a normal build process without sandcastle integration and wait for it to finish
- 2) Navigate into the Source Build Working Directory. The default directory is `$(SystemDrive)\Builds\$(BuildAgentId)\$(BuildDefinitionPath)\Sources`
- 3) Create a new SHFB project that is stored in you source control structure. It's recommended to store the file next to the main solution file. This gives you the opportunity to branch the SHFB project together with the main project.
- 4) Add references to the DLL's that have to be included to the Help File. You have to reference the local output directory for this purpose. The default output directory is

\$(SystemDrive)\Builds\\$(BuildAgentId)\\$(BuildDefinitionPath)\Binaries. Also ensure that your project file contains relative paths. See figure below

- 5) Start the helpfile generation process in the SHFB manually.
- 6) Save the created project file and check it in into the TFS.

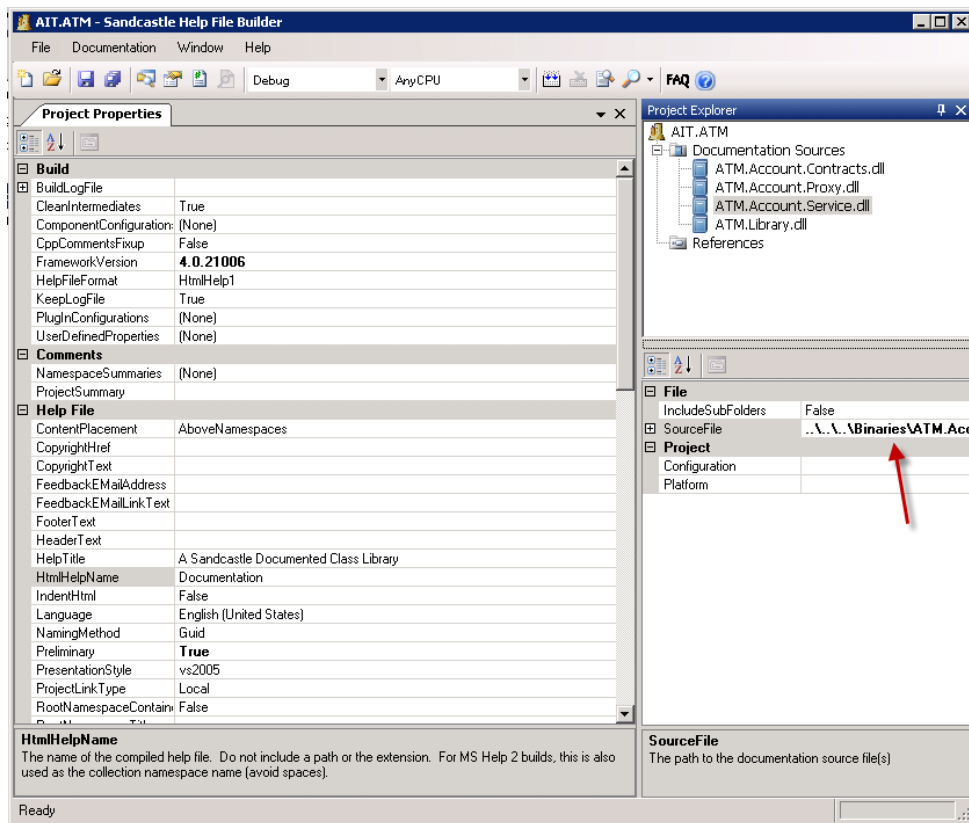


Figure 15 - A sample Sandcastle Help File Build project

## Configuration of the Build Process

The configuration of the Build process is very easy. All required properties are exported into a section that can easily be edited in the Build Definition Editor of Visual Studio. There are two properties that can be changed in order to configure the Sandcastle generation process

- 1) Enable: This property decided whether the Sandcastle Documentation will be generated
- 2) Project Files: This is a collection that contains all Sandcastle Helpfile Builder Project Files. These files will be compiled to proper documentation during the Build Process.

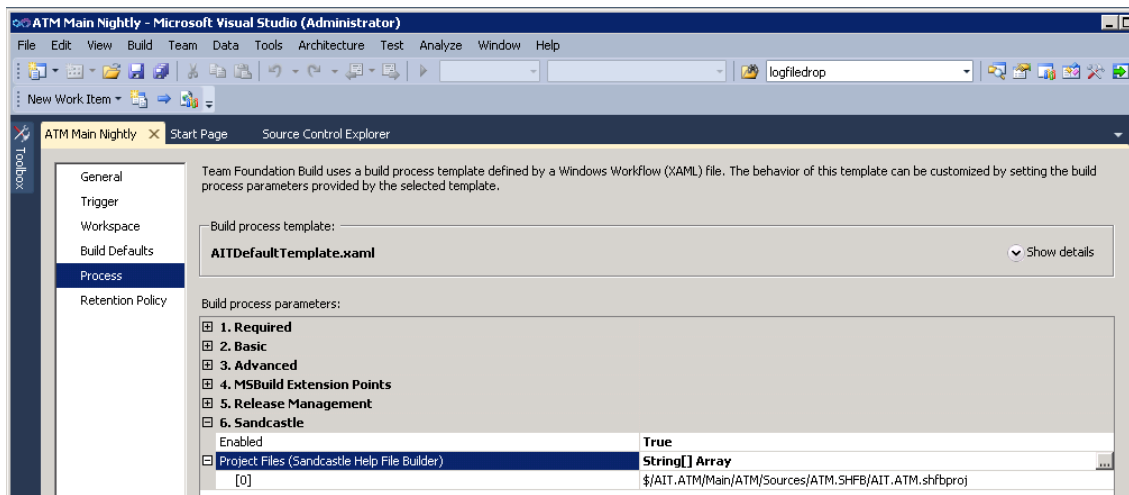


Figure 16 - Sample settings inside the build definition

## Appendix - Workflow details



Figure 17 – Basic Build Process

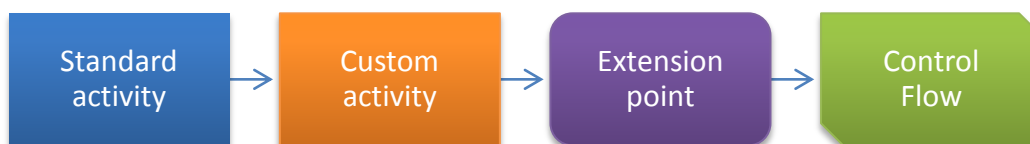


Figure 18 – Legend

## This document was created by

**AIT Applied Information Technologies GmbH & Co. KG**  
AIT TeamSystemPro Team

Postal address:  
Leitzstr. 45  
70469 Stuttgart

Amtsgericht Stuttgart  
HRA 725452

General Partner:  
AIT Verwaltungs GmbH  
Amtsgericht Stuttgart  
HRB 734136  
CEO: Lars Roith

IBAN: DE80 61191310 0664310001  
SWIFT: GENODES1VBP

Phone           +49 711 49066 430  
Fax              +49 711 49066 440  
Email           [info@aitgmbh.de](mailto:info@aitgmbh.de)  
Internet        [www.aitgmbh.de/teamsystempro](http://www.aitgmbh.de/teamsystempro)

This document is protected by German copyright laws and may only be reproduced, modified or extended with the written consent of the authors. All Rights reserved.