TeamSystemPro TEAM

AIT Build Suite 2012

V3.0.1401.1703

Copyright

Version History

Version	Comments	Date	Author
3.0.1209.0400	Inital version	04.09.2012	Michael Ring
3.0.1211.2900	Add support for Team Foundation Service	29.11.2012	Michael Ring
3.0.1212.0500	Add BeforeCompileConfiguration and BeforeCompileSolution MSBuild Extension Points	05.12.2012	Michael Ring
3.0.1301.1400	Fixed bug in build definition parameters export and MSBuild extension point	14.01.2013	Michael Ring & Sven Hubert
3.0.1304.1600	Add support for C++ Versioning and Doxygen	16.04.2013	Michael Ring & Thomas Rümmler
3.0.1304.2600	Add support for Test Result and Code Coverage Summary	26.04.2013	Michael Ring
3.0.1401.1703	Bugfixing	17.01.2014	Nico Orschel & Christian Schlag

Contents

Feature Overview

The AIT Build Suite 2012 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 The update template delivered by Microsoft is obsolete by using the AIT build process template.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 three 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 AIT Build Suite 2012 complex build processes can be configured easily using centralized builds with Microsoft Visual Studio Team Foundation Server 2012.

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).

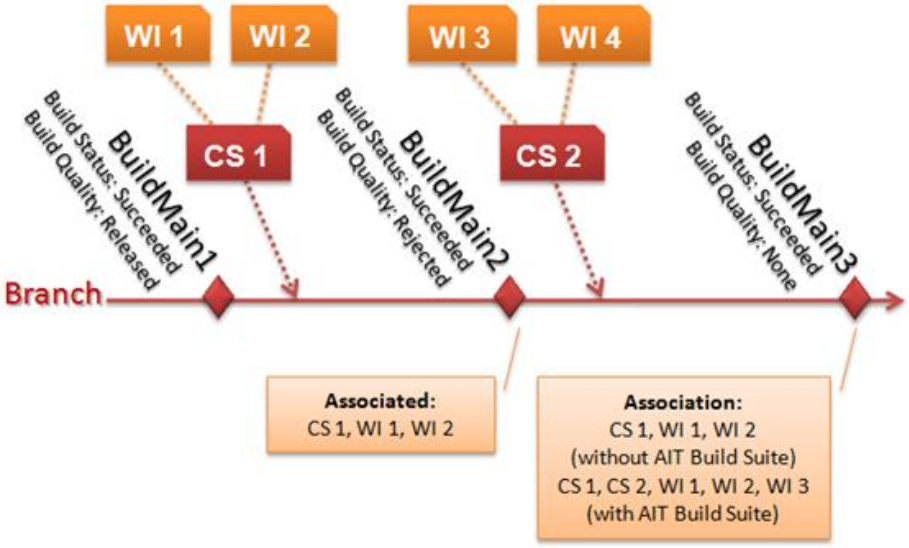

Figure 1 - Build association of Changesets and Work Items

But that's not sufficient. Figure 1 shows the changed behavior using the AIT Build Suite 2012. 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.

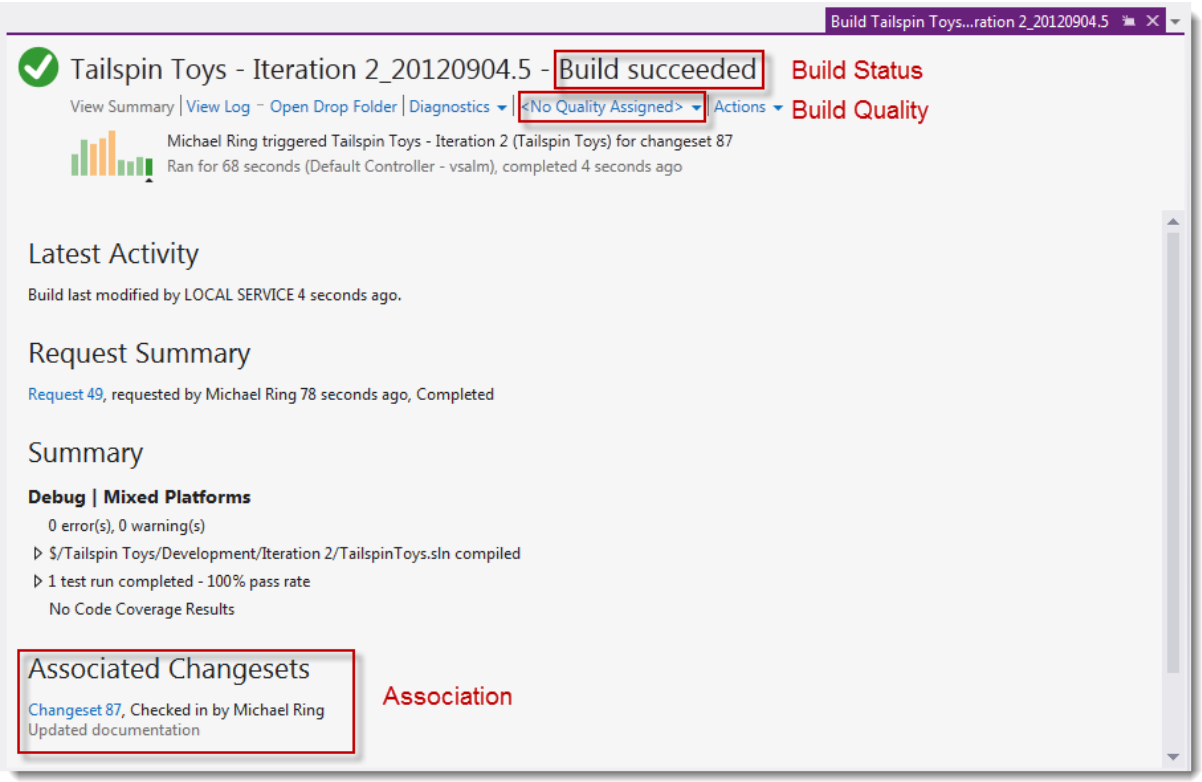

Figure 2 - Build summary

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 Team Foundation Server 2010 Workflow Foundation was used instead of central MSBuild. At the core of compilation, MSBuild was used (new version 4.0). Existing build scripts could not 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 were instrumented. That's why AIT delivered a generic Template with AIT Build Suite 2010 which provides MSBuild extension points. This accelerated the migration of build to TFS 2010 dramatically and can be used with the new version for Team Foundation Server 2012.

Tracing

By default, the AIT Process Template contains 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.

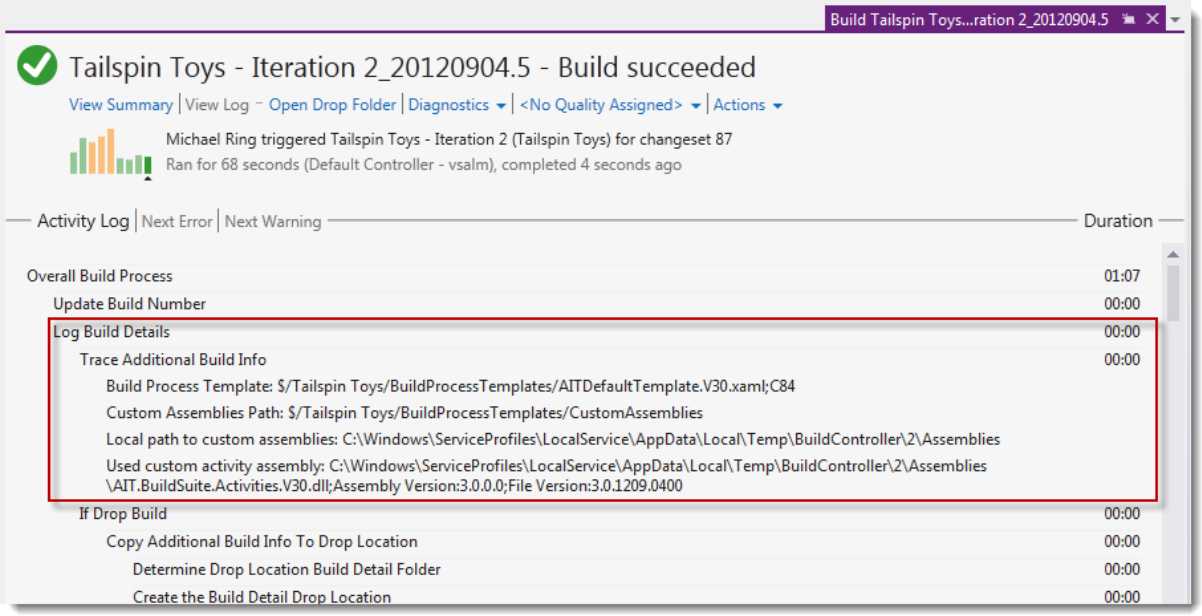

Figure 4 - Build trace in summary log

Boot Strapping

In order to deploy binaries which are necessary for the build (e.g. custom MSBuild task libraries) to the Build Agent, the **Deploy Custom Assemblies** switch can be used (see Figure 5).

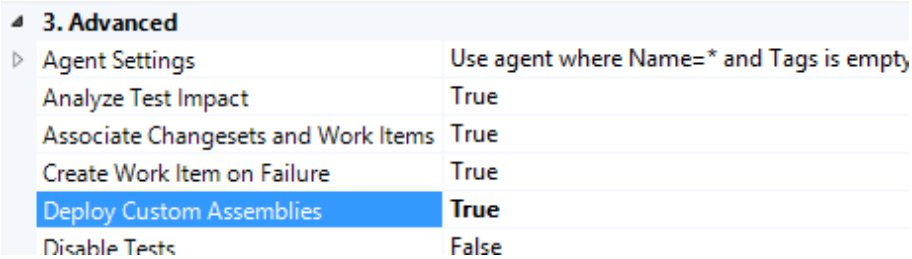

Figure 5 - Deploy Custom Assemblies

When set to True, the contents of the build controller's custom assemblies path is copied to a subfolder within the build directory on the agent (see Figure 6).

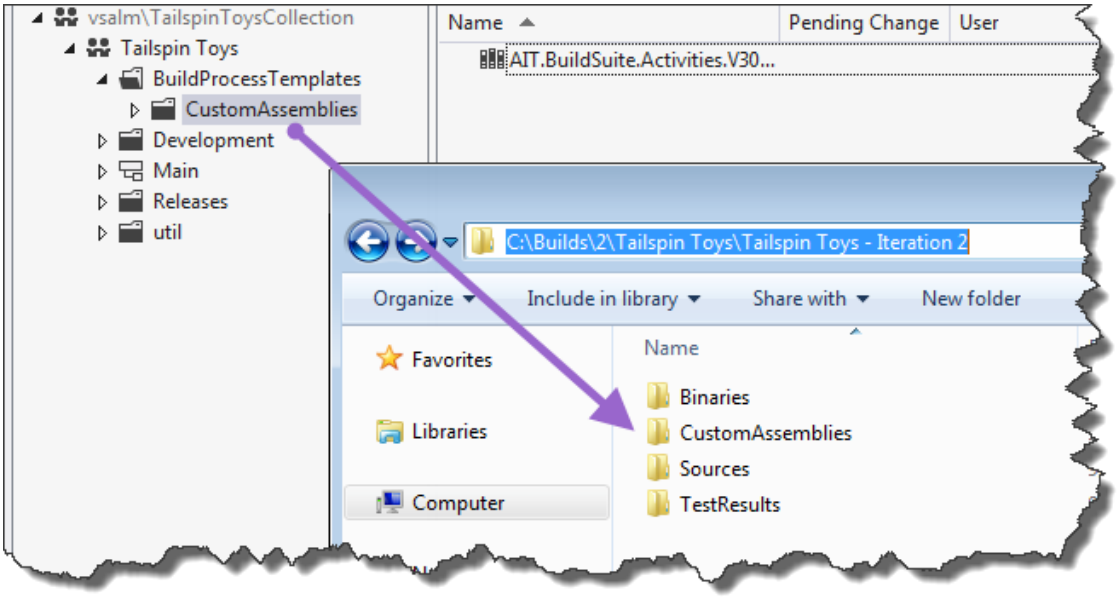

Figure 6 - Custom Assemblies folder in Drop Location

You can refer to this path as **$(CustomAssembliesPath)** within your MSBuild scripts e.g. to use in imports or UsingTask statements.

Deployment

The AIT Build Suite 2012 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.V30.xaml**. The additional activity library has to be deployed using the TF Build deployment features. Just put **AIT.BuildSuite.Activities.V30.dll** under source control and specify the folder in the build controller properties:

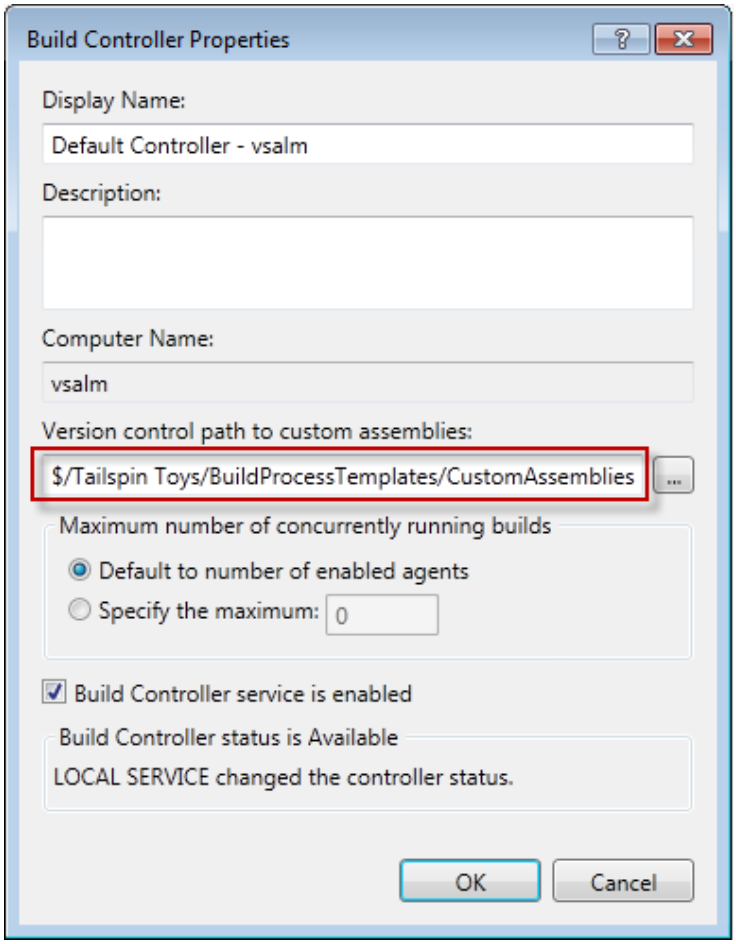

Figure 7 - Build controller properties with path to custom assemblies

In order to edit the process **AITDefaultTemplate.V30.xaml** file, you have to copy the **AIT.BuildSuite.Activities.V30.dll** to your local *PrivateAssemblies* folder of Visual Studio under:

`C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\PrivateAssemblies` (64 bit) or

`C:\Program Files\Microsoft Visual Studio 11.0\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.

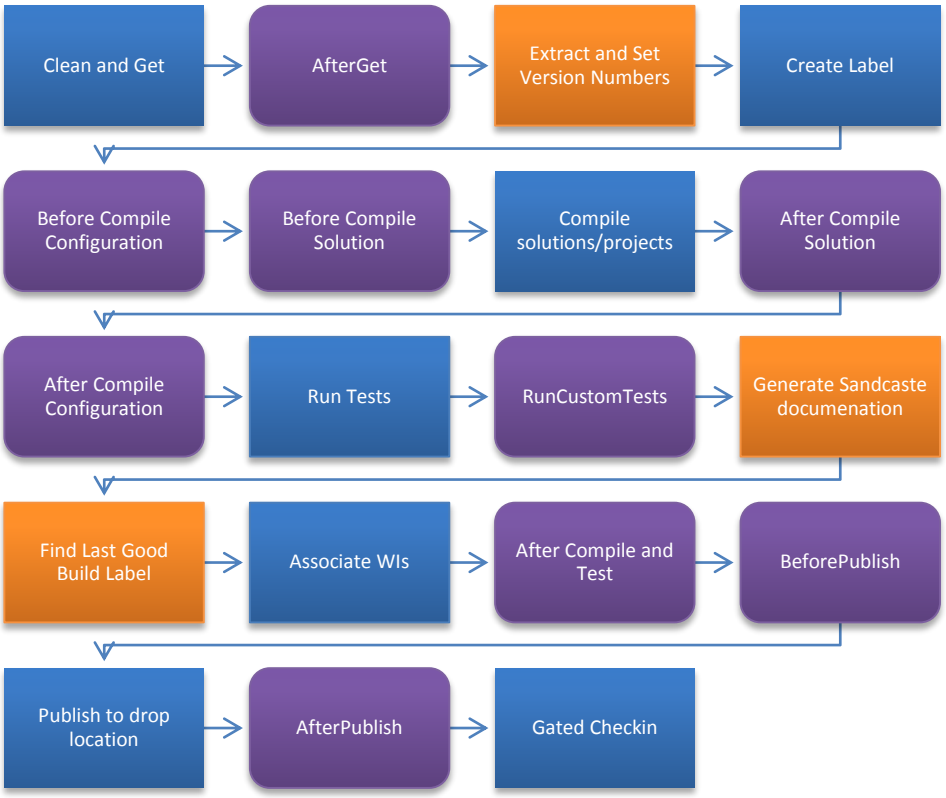

Figure 8 – Basic Build Process

Figure 9 – 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 as Figure 10 shows (Process tab).

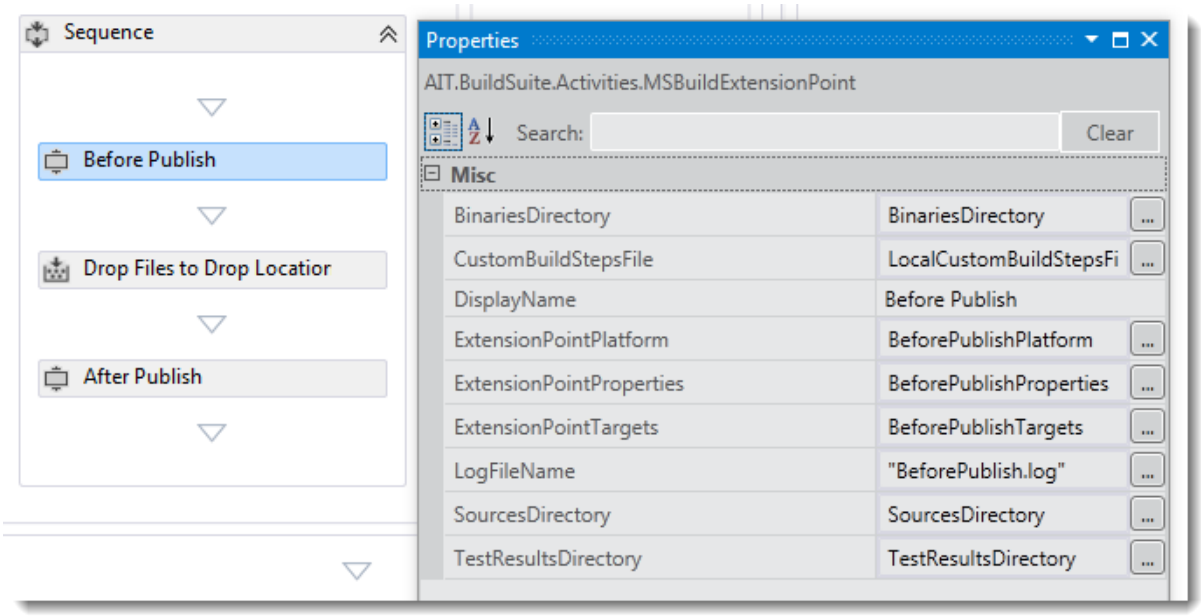

Figure 10 - Sample Extension Point with default initialization using global build arguments

Each extension point defines the following properties:

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* (see Figure 11) 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="'$(TeamFoundationServerUrl)' == ''">$(TFSUrl)</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 11 - Custom script sample

Extending an Extension Point by additional properties

The extension point activity is a sequence (see Figure 12). At first it will initialize some variables used to configure the call to MSBuild such as Script name to call, Targets to execute, Properties to transfer.

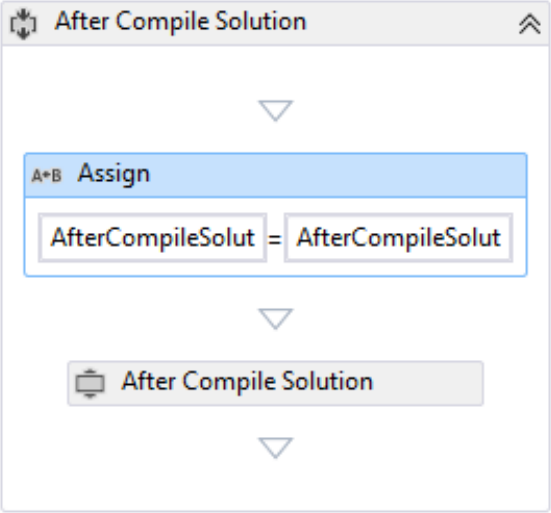

Figure 12 - Sample Extension Point with additional properties assigned

The Properties variable can be set inside the build definition and extended inside the process template like this:

```
AfterCompileSolutionProperties + ";Project=" + localProject + ";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.

The activity is encapsulated in the MSBuildExtensionPoint activitiy inside the AIT Build Suite 2012 activities DLL, so you don't have to create it yourself. Figure 13 shows the default properties for an extension point.

Figure 13 - Default Extension Point Activity Properties

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

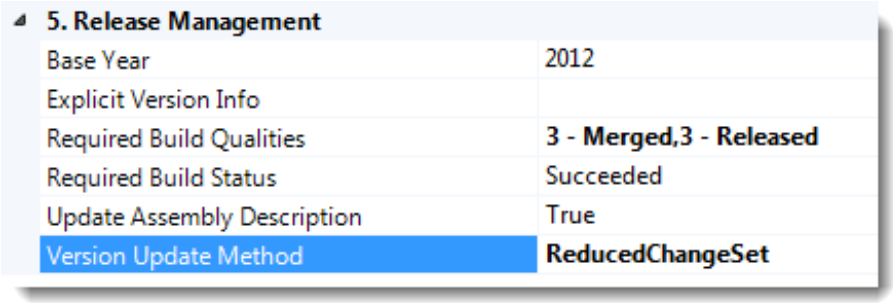

Figure 14 - Release Management Properties

Work Item and Changeset Association

The AIT Build Suite 2012 lets you change the default association behavior. The following properties can be set inside your build definition:

Property	Description
Required Build Qualities	The list of qualities used to filter the last good build that will be used for comparison. One of the qualities must apply.
Required Build Status	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 (see Figure 15).

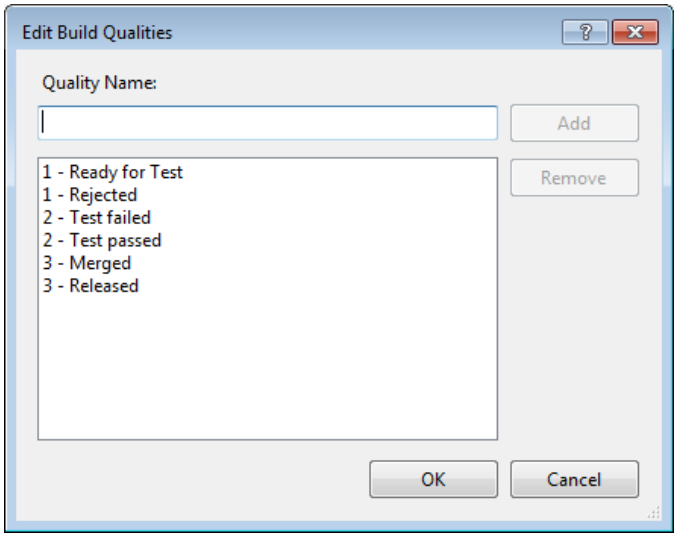

Figure 15 - Custom Build Qualities

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
Base Year	The base year is used 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)
Explicit Version Info	Sets 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.
Update Assembly Description	Sets the AssemblyDescription attribute in your Assembly to the complete BuildNumber. This way, you can trace an assembly to a distinct build.
Version Update Method	**None:** do not to set a version in AssemblyInfo files **FullDate:** use the full date for all 4 version digits **ReducedDate:** use the date only for the last 2 digits **BaseYear**: use the base year of development for the third digit **ReducedChangeset:** use the **SourceGetVersion** changeset id and split it at length 4 to fit into build and revision digits.

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 16 - 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.

Documentation

During build, documentation can be generated after compilation based on the annotations found in the source code.

Doxygen

Doxygen is an open-source tool used for creating documentation from annotated source code for C#, C++, and other languages.

Installation

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

1) Install Doxygen on all build servers (http://doxygen.org)
2) Reboot the build server

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 three properties that can be changed in order to configure the Doxygen generation process:

Property	Description
Enable Doxygen	Determines whether Doxygen documentation will be generated or execution will be skipped.
Doxygen Configuration Files	The list of Doxygen configuration files that will be used as instructions when generating the documentation.
Doxygen Additional Arguments	Additional arguments for configuration can be specified here to alter the default Doxygen configuration.

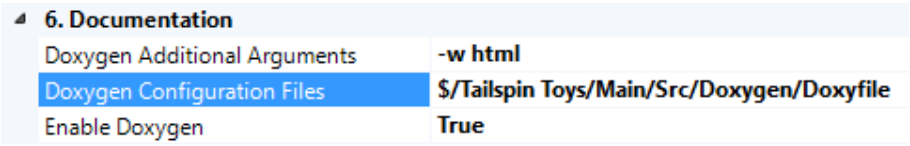

Figure 17 - Doxygen configuration parameters

Appendix – Workflow details

Figure 18 – Basic Build Process

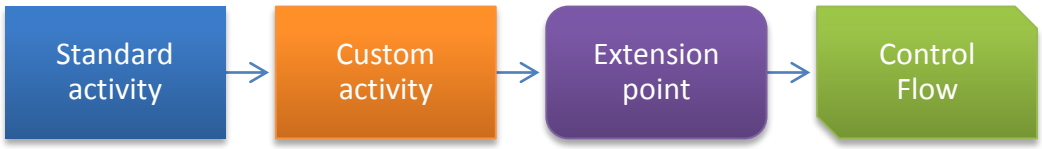

Figure 19 – Legend

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
Internet	www.aitgmbh.de/teamsystempro