AIT Applied Information Technologies GmbH & Co. KG
Internet: www.aitgmbh.de   Email: DependencyManager@aitgmbh.de

AIT Dependency Management Build Tasks

Installation and Configuration- *V14.0*

Table of Contents

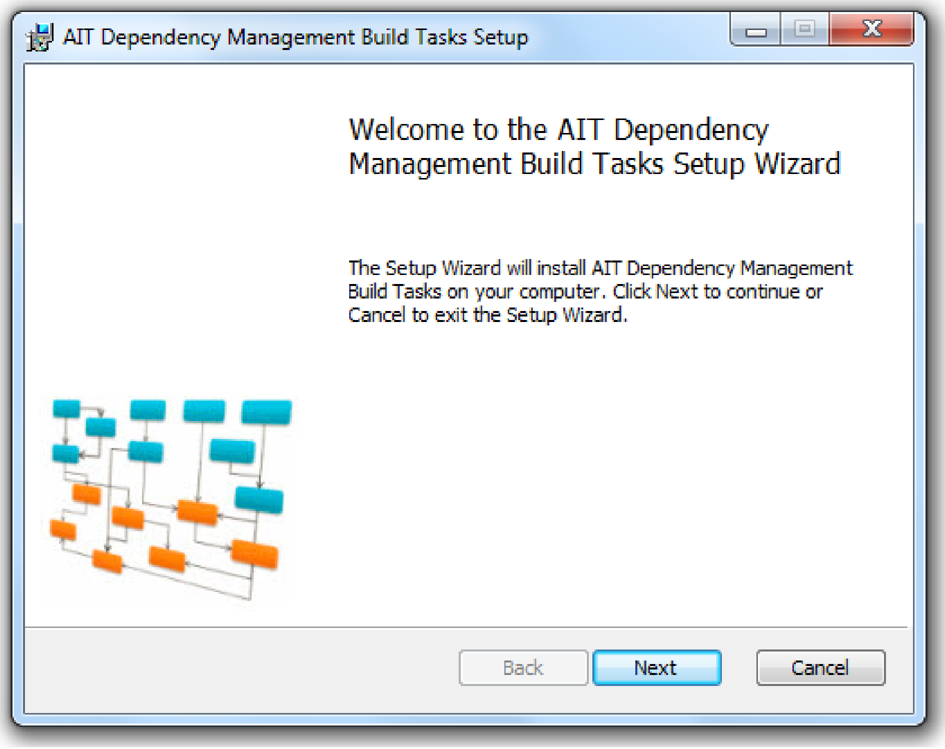
Installation

Preparation steps

1. Close any currently running instances of Visual Studio
2. Execute Setup with administrative rights

Executing the Setup

The installation package contains a MSI Installer (.msi). When the MSI Installer is executed a wizard will start and guide the user through the installation process (Figure 1).


AIT Dependency Management Build Tasks Setup

Welcome to the AIT Dependency Management Build Tasks Setup Wizard

The Setup Wizard will install AIT Dependency Management Build Tasks on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 1 - Setup Wizard

The following components are installed during the Setup:

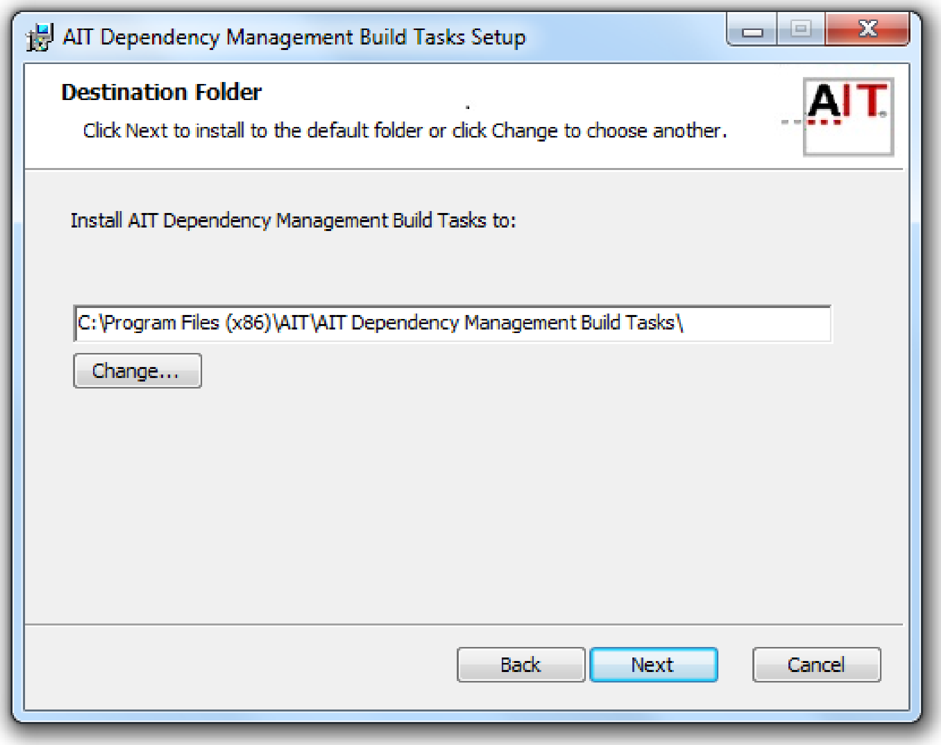
- MSBuild Tasks

During the Setup the following folders are created or modified:

- C:\Program Files (x86)\MSBuild\AIT\DMF (MSBuild Tasks)
- C:\Program Files (x86)\AIT\AIT Dependency Management Build Tasks (Documentation)

Due to the modification of the Program Files directory **administrative rights are needed during the installation**.

The installation directory for MSBuild Tasks is fixed and cannot be modified by the user during the Setup routine. The documentation is copied to the *Installation Folder,* which can be modified by the user (Figure 2).


AIT Dependency Management Build Tasks Setup

Destination Folder

Click Next to install to the default folder or click Change to choose another.

Install AIT Dependency Management Build Tasks to:

C:\Program Files (x86)\AIT\AIT Dependency Management Build Tasks\

Figure 2 – Installer Configuration Dialog

MSBuild Integration

Installation

The MSBuild Task assemblies are installed automatically by the Setup routine. The assemblies are located in *C:\Program Files (x86)\MSBuild\AIT\DMF*.

Prerequisites

For integrating the Dependency Management into the central build it is assumed that the folder structures (especially for the referenced libraries and compile outputs) is the same as on the local developer machine. This might involve additional build customization which is not part of the product.

GetDependencies MSBuild Task

The *GetDependencies* MSBuild Task can be configured via several parameters:

- Required parameters:
    - `DependencyDefinitionPath`
- `Optional parameters:`
    - `RelativeOutputPath`
    - `Force`
    - `Recursive`
    - `BinaryTeamProjectCollectionUrl`
    - `BinaryRepositoryTeamProject`

The parameter *DependencyDefinitionPath* specifies the location of the dependency definition file. This can be a source control path or a relative path and must be defined as a parameter.

Listing 1 shows the minimal configuration for a dependency definition file present in the same folder as the MSBuild script.

```
<Target Name="GetDependencies">
  <GetDependencies DependencyDefinitionPath="component.targets"
</Target>
```

Listing 1 - Minimal configuration for GetDependencies MSBuild task

The default location where downloaded components are placed is the directory of the dependency definition file on the workstation. With the parameter *RelativeOutputPath* a directory relative to the dependency definition file directory can be specified. By default, the *RelativeOutputPath* is configured as *"..\bin"*.

By default, all dependency files are fetched recursively every time a build is triggered. In case of an incremental build the default behavior can be changed to incremental get of dependencies by setting the *Force* parameter to false. By setting the *Recursive* parameter to false the behaviour can be changed to only fetching direct dependencies.

A location for the Binary Repository can be specified by using the two parameters *BinaryTeamProjectCollectionUrl* and *BinaryRepositoryTeamProject*. If none of these is used the Team Project Collection bound to the build agent workspace and the team project *BinaryRepository* is used as configuration for the Binary Repository.

If any of the optional parameters are not configured the default value is used.

CleanDependencies MSBuild Task

The *CleanDependencies* MSBuild Task can be configured via several parameters:

- Required parameters:
  - DependencyDefinitionPath
- Optional parameters:
  - RelativeOutputPath
  - BinaryTeamProjectCollectionUrl
  - BinaryRepositoryTeamProject

Listing 2 shows the minimal configuration for a dependency definition file present in the same folder as the MSBuild script.

```
<Target Name="CleanDependencies">
  <CleanDependencies DependencyDefinitionPath="component.targets"
</Target>
```

Listing 2 - Minimal configuration for CleanDependencies MSBuild task

If any of the optional parameters are not configured the default value is used. The default values are equal to the default values used for the *GetDependencies* MSBuild Task

Configuration

In order to get dependencies and clean already downloaded dependencies during the central build, the MSBuild tasks need to be integrated into the build process. There are two options depending on the build process template which is used.

Microsoft‘s DefaultTemplate.xaml

To integrate the *GetDependencies* and *CleanDependencies* MSBuild tasks using the *DefaultTemplate.xaml* template, the user needs to create an MSBuild script and reference the MSBuild script as an *Items To Build* in the build definition.

The created MSBuild script will get the dependencies, build the solutions and clean the downloaded dependencies after the build. The fetching of the dependencies is triggered by calling the *GetDependencies* MSBuild task. For cleaning dependencies the *CleanDependencies* MSBuild task is triggered. A sample MSBuild script is shown in Listing 3.

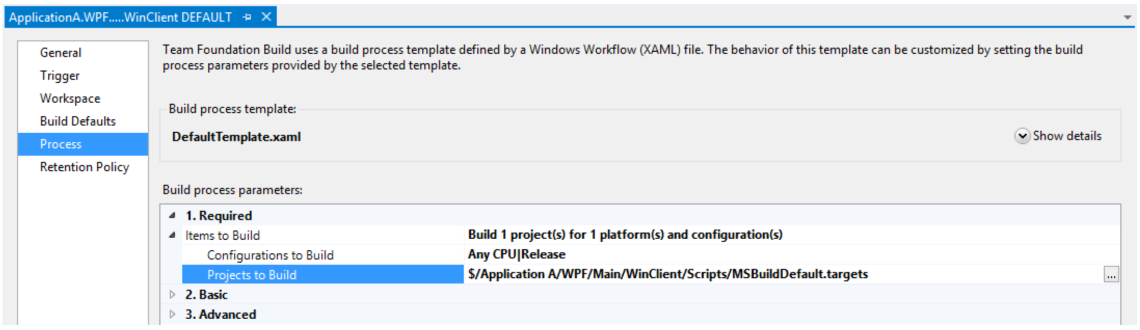
```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="GetDependencies;Build;CleanDependencies" xmlns="
http://schemas.microsoft.com/developer/msbuild/2003" >
  <!-- AIT.DMF.MSBuild -->

  <UsingTask TaskName="AIT.DMF.MSBuild.GetDependencies"
   AssemblyFile="$(MSBuildExtensionsPath32)\AIT\DMF\AIT.DMF.MSBuild.dll" />
  <UsingTask TaskName="AIT.DMF.MSBuild.CleanDependencies"
   AssemblyFile="$(MSBuildExtensionsPath32)\AIT\DMF\AIT.DMF.MSBuild.dll" />

  <ItemGroup>
    <ItemsToBuild Include="ViewModule.sln" />
  </ItemGroup>

  <Target Name="GetDependencies">
    <Message Text="GetDependencies" />
    <GetDependencies DependencyDefinitionPath="component.targets" />
  </Target>

  <Target Name="CleanDependencies">
    <Message Text="CleanDependencies" />
    <GetDependencies DependencyDefinitionPath="component.targets" />
  </Target>

  <Target Name="Build">
    <MSBuild Projects="@(ItemsToBuild)" Condition="'$(OutDir)' != ''" Properties="
OutDir=$(OutDir);VCBuildOverride=$(VCBuildOverride);SkipInvalidConfigurations=$(Sk
ipInvalidConfigurations);TeamFoundationServerUrl=$(TeamFoundationServerUrl);TeamPr
oject=$(TeamProject);SourcesDirectory=$(SourcesDirectory);BinariesDirectory=$(Bina
riesDirectory);DropLocation=$(DropLocation);TestResultsDirectory=$(TestResultsDire
ctory);Reason=$(Reason);SourceGetVersion=$(SourceGetVersion);LabelName=$(LabelName
);RunCodeAnalysis=$(RunCodeAnalysis)"  />
    <MSBuild Projects="@(ItemsToBuild)" Condition="'$(OutDir)' == ''" />
  </Target>
</Project>
```

Listing 3 - Example Script for use with DefaultTemplate.xml

The sample script needs to be adapted to reference the solution which should be build (as a relative path to the location of the script), contain the relative or source control path to the dependency definition file and optional parameters for the Dependency Management MSBuild Tasks should be added as needed. After the adaption it should be checked into Source Control underneath the component path (within the branch folder).

In some cases the MSBuild script is not working correctly when **$(MSBuildExtensionsPath32)** is used. If this is the case, please use **$(MSBuildExtensionsPath)**.

As a second step the existing build definition must be changed to reference the MSBuild script as an Item To Build (Figure 3).



Team Foundation Build uses a build process template defined by a Windows Workflow (XAML) file. The behavior of this template can be customized by setting the build process parameters provided by the selected template.

Build process template:
DefaultTemplate.xaml

Build process parameters:

1. Required	
Items to Build	Build 1 project(s) for 1 platform(s) and configuration(s)
Configurations to Build	Any CPU|Release
Projects to Build	$/Application A/WPF/Main/WinClient/Scripts/MSBuildDefault.targets
2. Basic	
3. Advanced	

Figure 3 - Items To Build customization

AIT Build Suite

In order to separate getting dependencies from building the solutions the AIT BuildSuite can be used, which includes hooks to call MSBuild scripts from other so called extension points during the build.

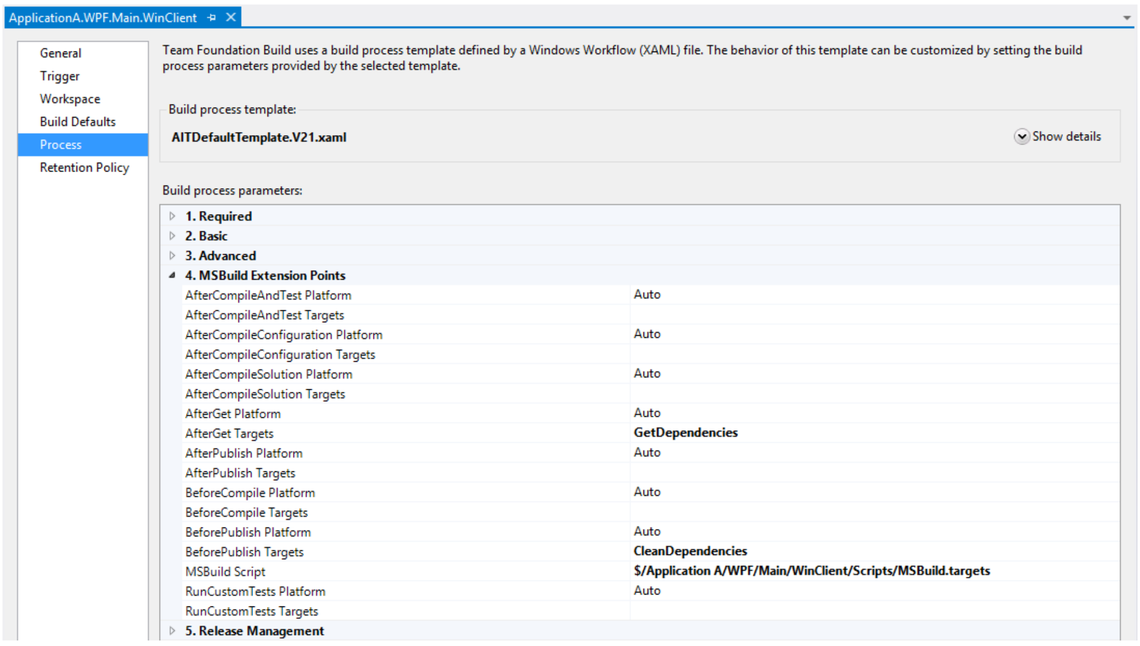
The MSBuild script only calls the *GetDependencies* and *CleanDependencies* MSBuild Tasks and therefore is much shorter as when using the *DefaultTemplate.xaml* Build Template. Listing 4 provides a sample script.

```
<?xml version="1.0" encoding="utf-8"?>
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <!-- AIT.DMF.MSBuild -->

  <UsingTask TaskName="AIT.DMF.MSBuild.GetDependencies"
   AssemblyFile="$(MSBuildExtensionsPath32)\AIT\DMF\AIT.DMF.MSBuild.dll" />
  <UsingTask TaskName="AIT.DMF.MSBuild.CleanDependencies"
   AssemblyFile="$(MSBuildExtensionsPath32)\AIT\DMF\AIT.DMF.MSBuild.dll" />

  <Target Name="GetDependencies">
    <GetDependencies DependencyDefinitionPath="component.targets" />
  </Target>

  <Target Name="CleanDependencies">
    <CleanDependencies DependencyDefinitionPath="component.targets" />
  </Target>
</Project>
```

Listing 4 - Example script for use with AIT Build Suite

The sample script needs to be adapted to contain the path the component.targets in the parameter *DependencyDefinitionPath* and optional parameters must be added as needed.

To call the MSBuild script during the build the After Get extension point, Before Publish extension point and MSBuild Script parameter must be modified in the Build Definition. These parameters can be configured in the *Process Tab* when editing the Build Definition (Figure 4).



Team Foundation Build uses a build process template defined by a Windows Workflow (XAML) file. The behavior of this template can be customized by setting the build process parameters provided by the selected template.

Build process template:
AITDefaultTemplate.V21.xaml

Build process parameters:

1. Required	
2. Basic	
3. Advanced	
4. MSBuild Extension Points	
AfterCompileAndTest Platform	Auto
AfterCompileAndTest Targets	
AfterCompileConfiguration Platform	Auto
AfterCompileConfiguration Targets	
AfterCompileSolution Platform	Auto
AfterCompileSolution Targets	
AfterGet Platform	Auto
AfterGet Targets	**GetDependencies**
AfterPublish Platform	Auto
AfterPublish Targets	
BeforeCompile Platform	Auto
BeforeCompile Targets	
BeforePublish Platform	Auto
BeforePublish Targets	**CleanDependencies**
MSBuild Script	**$/Application A/WPF/Main/WinClient/Scripts/MSBuild.targets**
RunCustomTests Platform	Auto
RunCustomTests Targets	
5. Release Management	

Figure 4 - Process parameter customization

Feedback

You need further information, or help? Or you want to use *Dependency Manager* at your project but you need some special functionality at the product? You need support for your dependency repository or use custom download methods? Or just want to leave us your ***Feedback***?

Contact us at DependencyManager@aitgmbh.de!

Copyright

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

Email	info@aitgmbh.de
Internet	www.aitgmbh.de
Phone	+49 711 49066 430
Fax	+49 711 49066 440

Postal address:	General Partner:	CEO: Lars Roith
Leitzstr. 45	AIT Verwaltungs GmbH	
70469 Stuttgart	Amtsgericht Stuttgart	IBAN: DE80 61191310 0664310001
Deutschland	HRB 734136	SWIFT: GENODES1VBP