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

AIT Dependency Manager

Extended Dependency Management with TFS and VS 2015 – V14.0

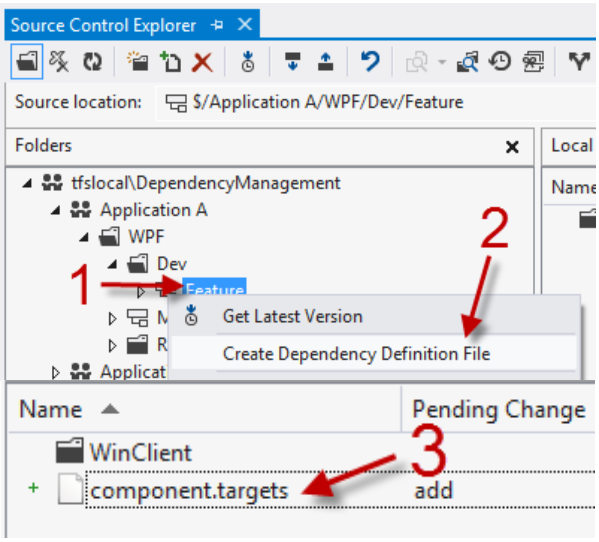
Contents

Quick Start: Installation

A developer can choose to directly install Dependency Manager from the Visual Studio Gallery or from inside Visual Studio by using the Extension Manager.

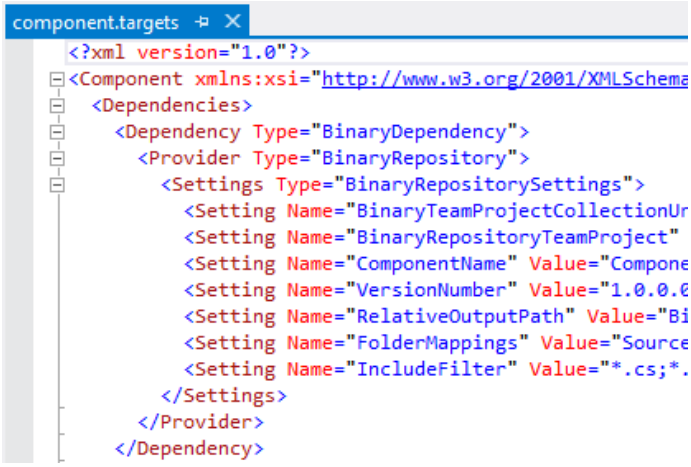
A new version of Dependency Manager is listed in the Updates list of the Visual Studio Extension Manager (assumed that Visual Studio Extension Manager is configured to look periodically for updates).

Quick Start: Add a new dependency definition file



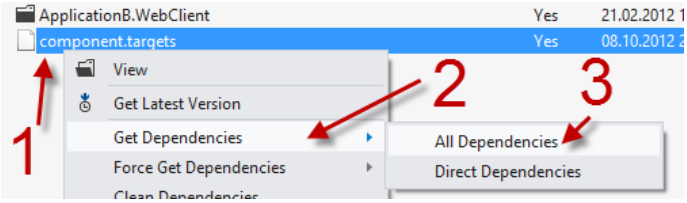
(1) **Right click** on a folder
(2) Click ***Create Dependency Definition File***

(3) A **new dependency definition file** is automatically created and added to the Team Foundation Server (TFS) as a pending change



(4) **Open the dependency definition file** by double clicking on the file
(5) **Add a new dependency** to the dependency definition file:
   a. Create a new dependency (in this example a binary dependency)
   b. Add the build result provider as the dependency provider to use
   c. Specify the dependency component by adding the Team Project name, a build definition and the build number
(6) **Check in** the dependency definition file

Quick Start: Get dependencies



(1) Open the context menu of a dependency definition file
(2) Select **Get Dependencies**
(3) Choose **All Dependencies**

Product Usage

AIT Dependency Manager enables the developer to manage dependencies to external libraries and sources and automatically resolve them inside Visual Studio.

Dependencies to external libraries are defined per software component in a dependency definition file. Dependencies can be defined per branch or per component inside a branch. The default name for this dependency definition file is *component.targets*.

Set up the Visual Studio Environment

In Visual Studio a developer should be connected to the Team Foundation Server and have chosen the corresponding Team Project for the component.

A workspace should be created for the chosen Team Project with active mappings for the directory the developer is working on.

Create a dependency definition file

To create a dependency definition file for a component, in *Source Control Explorer* the developer clicks on a folder and chooses the option „**Create Dependency Definition File“** from the context menu of the target folder. The selected folder has to be a sub folder of a branch folder or the branch folder itself.

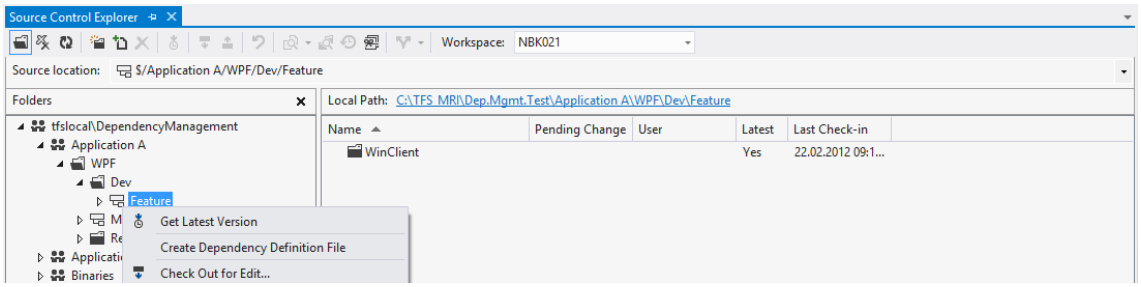


Figure 1 - Adding a new dependency definition file

As a result a new dependency definition file is created in the selected folder and is added to the Team Foundation Server (TFS) Version Control (See Figure 2). The dependency definition file contains a minimal dependency definition markup which must be edited to the developer needs.

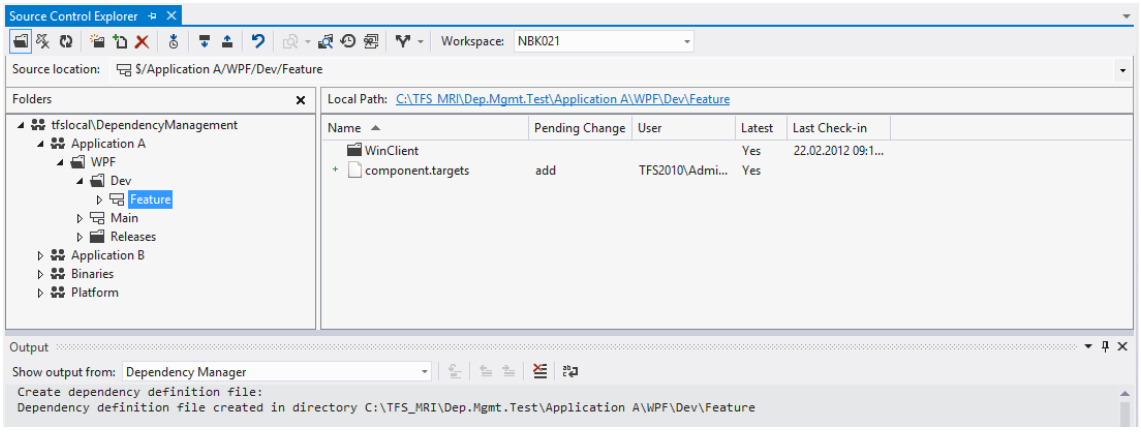


Figure 2 - Automatically created dependency definition file

Edit component dependencies

Visual Editor

The visual editor provides a rich editing experience for users. While editing a dependency definition file, the impact on the dependency graph can be visualized immediately after saving the file.

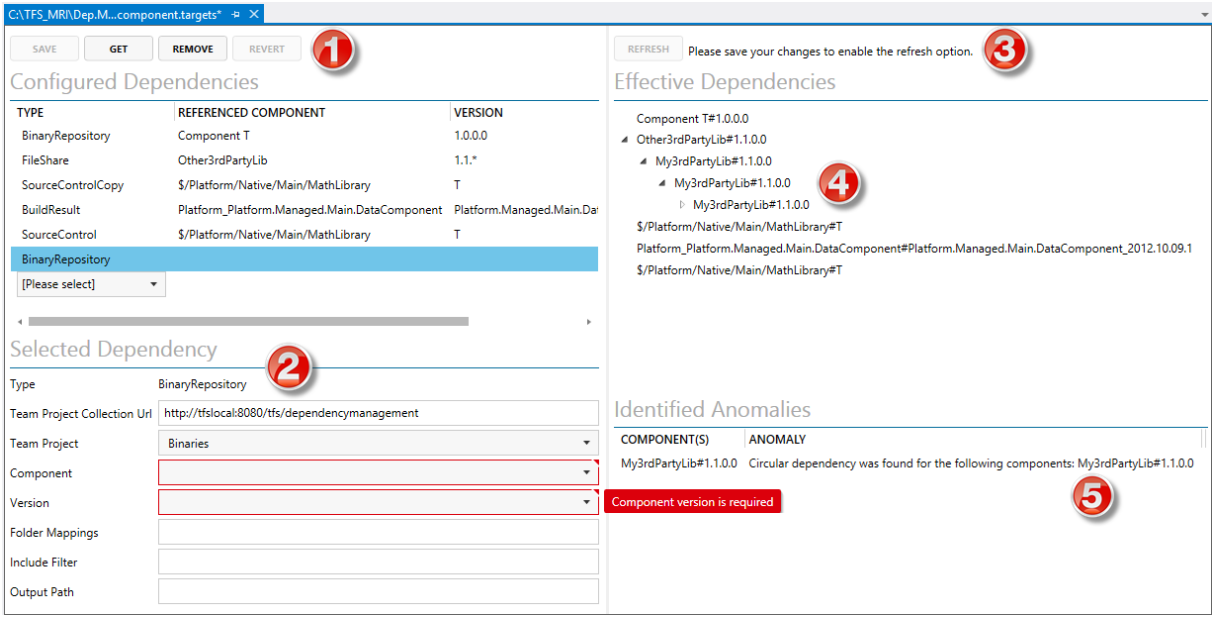


Figure 3 - Editing a dependency definition file with the Visual Editor

The Visual Editor is divided into two sides. The left-hand side can be used to add, change or remove dependencies. The right-hand side shows the resulting dependency tree and detected anomalies.

On the left, the following functions are available (1):

Save … saves the file.

Get … gets the dependencies incrementally as defined.

Remove … removes the selected dependency from the list.

Revert … sets back the selected dependency to the state last saved.

A selected dependency can be configured in the bottom area on the left (2). Red boxes show validation errors for each invalid setting with a validation message indicating the necessary change.

On the right, the ***Refresh*** (3) button should be used to trigger the generation of the dependency tree (4) and anomalies list (5) after loading or trigger a refresh after changes to the file have been saved.

The file can also be saved using the save functions of Visual Studio e.g. via the File menu.

XML Editor

To add or edit dependencies the dependency definition file can also be edited in Visual Studio by opening the dependency definition file (See Figure 4). IntelliSense Support is provided in Visual Studio when editing a dependency definition file.

Figure 4 - Editing a dependency definition file with the Xml editor

After the creation of the dependency definition file, it must be checked in. After the first check in the developer can make changes locally but must check in if he wants to distribute the changes.

Get dependencies

The download of the dependencies can be triggered via several ways in *Source Code* Explorer and *Solution Explorer*: The component folder (with the dependency definition file inside) or the dependency definition file itself can be selected.

In Solution Explorer the developer must add the dependency definition file to the solution before the context menu will list the entries "**All Dependencies**" and "**Direct Dependencies**" for the „**Get Dependencies**" submenu (See Figure 5).

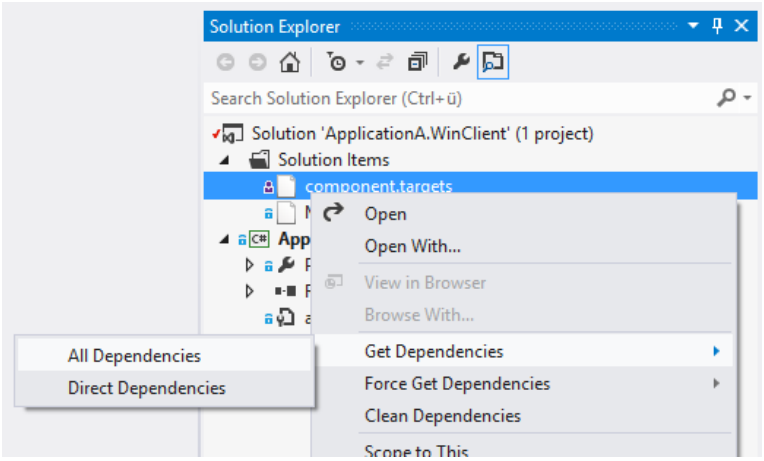


Figure 5 - Get dependencies in Solution Explorer

In the *Source Control Explorer* the developer clicks on the dependency definition file. The context menu for the selected item will contain the entries „**All Dependencies**“ and “**Direct Dependencies”** in the “**Get Dependencies**” submenu (Figure 6).

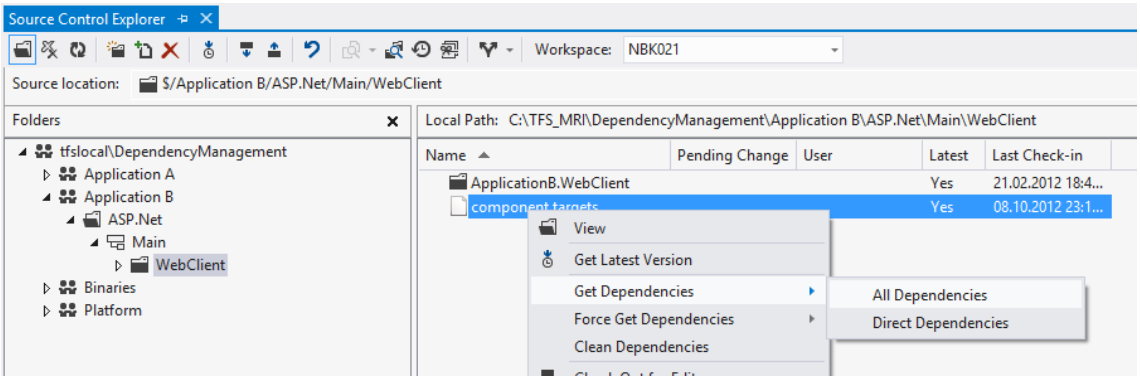


Figure 6 - Getting dependencies for a specific component

After clicking one of the entries of the **„Get Dependencies“** submenu, the dependency definition file is analyzed and the dependencies are resolved. Resolving the dependencies is done by downloading the referenced libraries or sources to a local directory and (non)recursively resolving the dependencies of the referenced libraries/sources itself. Libraries/sources can be downloaded from a network share, can be a TFS Build result or can be fetched from the TFS Source Control. The referenced libraries which are downloaded are logged in the **Dependency Manager Output Window**.

In case of “**Direct Dependencies**” only the dependencies specified in the selected dependency definition file are fetched. If dependencies should be recursively fetched “**All Dependencies**” must be selected.

By default the referenced libraries/sources are downloaded to a **Bin** directory in the root folder of the dependency definition file directory. The download directory can be specified for every dependent component inside the dependency definition file.

The two options in the **„Get Dependencies“** submenu fetch only files of referenced libraries/sources that have changed since the last time dependencies were fetched. If the developer wants to fetch all files of referenced libraries/sources again he has to use options from the **„Force Get Dependencies“** submenu in *Source Control Explorer* or *Solution Explorer*.

Clean dependencies

The cleanup of downloaded libraries can be triggered via the same way in *Source Code* Explorer and *Solution Explorer*: The dependency definition file should be selected.

In the Source Code Explorer the developer clicks on the dependency definition file. The context menu for the selected item will contain an entry „**Clean Dependencies**“ (Figure 7).

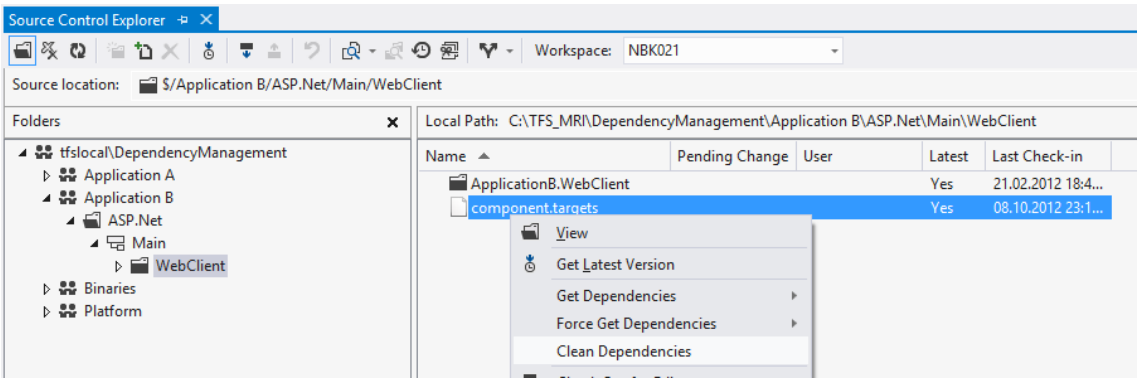


Figure 7 – Cleaning up all downloaded files for a specific component

After clicking the context menu entry **„Clean Dependencies“**, the dependency definition file is analyzed and the local directory of every referenced library is deleted. The libraries which are cleaned up are logged in the **Dependency Manager Output Window**.

If a developer selects a dependency definition file in *Source Control Explorer* or *Solution Explorer* the context menu will also show the entry **„Clean Dependencies“** with the same functionality (Figure 8).

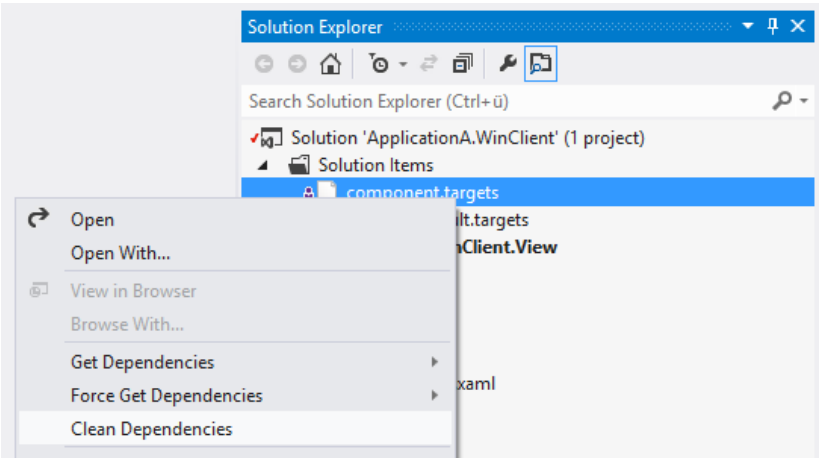


Figure 8 - Clean dependencies in Solution Explorer

Additional information

Selecting files and folders

For dependencies other than SourceControlMapping you can provide additional information to further control what files and folders are fetched, Figure 9 shows the user interface. If you set an include filter, only items that match this filter are fetched. If you set an exclude filter, only items that don't match that filter are fetched. If a file or folder is encountered that matches both include and exclude criteria, excluding takes precedence.

Figure 9 - Fields to control file and folder fetching

By default, if no filter are set all files are included which is equivalent to setting the include filter to "*" and the exclude filter to "". While fetching a dependency, the filters are applied to all items found at the dependency location. Folder filters must end with a backslash ('\') and are treated as paths relative to the dependency location, so excluding "Log\" will only exclude this folder if it is a direct subfolder of the dependency location. Other subfolders with the Name "Log" are not affected. You can use wildcards ("?","*") in the filters and filters are not case sensitive. Multiple filters must be separated by a semicolon.

Additionally, you can use folder mappings to rename folders while fetching a dependency. The syntax is "SourceOffset=A,LocalOffset=B" where A is the name of the folder in the dependency location and B the name you want for this folder locally. Multiple mappings must be separated by a semicolon.

Example: Given the drop location Figure 10, you want the build result dependency to only copy the "FormsApplication.exe" and the log folder with its contents except the ActivityLog.xsl. One solution is the configuration shown in Figure 11. If you then decide that you need the log folder but none of its subfolders you can add "Logs*\" to the exclude filter. This will exclude all subfolders but still include all contained files.

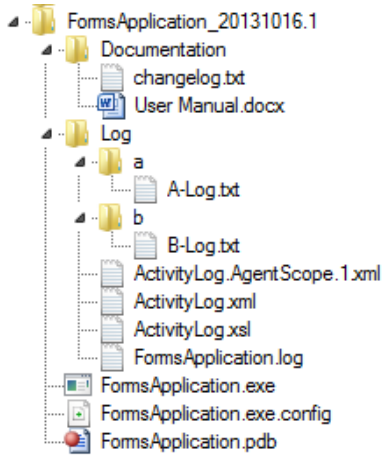


Figure 10 - Drop location

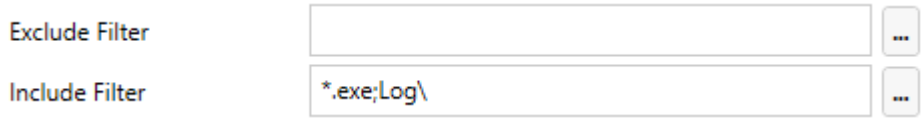


Figure 11 - Solution

The xml configuration for this example

```
<Dependency Type="BinaryDependency">
  <Provider Type="BuildResult">
    <Settings Type="BuildResultSettings">
      <Setting Name="TeamProjectName" Value="Project B" />
      <Setting Name="BuildDefinition" Value="FormsApplication" />
      <Setting Name="BuildStatus" Value=" Succeeded" />
      <Setting Name="BuildQuality" Value="Ready for Deployment" />
      <Setting Name="IncludeFilter" Value="*.exe;Logs\" />
      <Setting Name="ExcludeFilter" Value="" />
    </Settings>
  </Provider>
</Dependency>
```

IncludeFilter	Semicolon separated list of filters that define what files and folders to include.
ExcludeFilter	Semicolon separated list of filters that define what files and folders to exclude

Dependency definition file syntax

The dependency definition file contains XML which describes the dependencies a component has. The component is represented by the component node (*<Component>*) with one required and two optional attributes. The *xmlns* attribute defines the namespace for the component (Default value: http://schemas.aitgmbh.de/DependencyManager/2011/11). The name and the version of the component can be specified by adding the *Name* and *Version* attributes.

Each component node must have a child node <Dependencies>, which defines the dependencies of a component. When a component has no dependencies then the dependencies node has no child nodes. Every component must define these elements as a minimal dependency definition file (See Listing 1).

```
<Component xmlns="http://schemas.aitgmbh.de/DependencyManager/2011/11">
  <Dependencies />
</Component>
```

Listing 1 - Minimal dependency definition file

When the developer specifies dependencies each dependency is represented as a dependency node (*<Dependency>*). Each dependency node must define the type of the dependency via the *Type* attribute. Valid dependency types are *BinaryDependency* and *SourceDependency*.

Every dependency is resolved via a dependency provider and settings which are used to specify the dependency in detail. Therefore every dependency node must have a provider node (*<Provider>*) which itself has a Type and a provider settings child node (*<Settings>*).

For a source dependency the only valid providers are the Source Control Provider (Type *SourceControl*) and the Source Control Copy Provider (Type *SourceControlCopy*). Components checked in into the Team Foundation Server can be fetched via these providers. A component is specified by the source control path and the version to fetch and must contain a dependency definition file itself in the root folder of the component. The source control path and the version must be provided via a child node in the settings node (Type *SourceControlSettings or SourceControlCopySettings*) (See Listing 2).

All Components can be excluded from the Side-By-Side Analysis Checks by setting the *IgnoreInSideBySideAnomalyChecks* (See Listing 2).

```
<Settings Type="SourceControlSettings">
  <Setting Name="ServerRootPath" Value="$/TestFramework/Main/TestFramework" />
  <Setting Name="VersionSpec" Value="D2011-12-01T15:40" />
  <Setting Name="IgnoreInSideBySideAnomalyChecks" Value="True" />
</Settings>
```

Listing 2 – Valid source control settings (*SourceControl* provider)

Additional settings for the *SourceControlCopy* Provider are the *IncludeFilter* and *FolderMappings*: Via the *IncludeFilter* setting only a subset of files will be fetched from the source control folder and the setting *FolderMappings* can be used to fetch a subset of folders (See Listing 3).

```
<Settings Type="SourceControlCopySettings">
  <Setting Name="ServerRootPath" Value="$/TestFramework/Main/TestFramework" />
  <Setting Name="VersionSpec" Value="D2011-12-01T15:40" />
  <Setting Name="IncludeFilter" Value="*.dll" />
  <Setting Name="FolderMappings"
           Value="SourceOffset=SourceSubfolder,LocalOffset=LocalSubfolder;..."/>
</Settings>
```

Listing 3 – Valid source control settings (*SourceControlCopy* provider)

Binaries can be also specified as a dependency. Common binary locations are a folder or file share, are part of a build result or binaries can be checked into the source control. For this use cases the Dependency Manager provides the *FileShare*, the *BuildResult* and the *BinaryRepository* provider (Type *FileShare*, *BuildResult* and *BinaryRepository*).

A component on a file share can be specified via the settings *FileShareRootPath*, *ComponentName* and *VersionNumber*. Via the *IncludeFilter* setting only a subset of files will be fetched from the component folder on the file share and the setting *FolderMappings* can be used to fetch a subset of folders from the file share (See Listing 4).

```
<Settings Type="FileShareSettings">
  <Setting Name="FileShareRootPath" Value="\\nbk021\Fileshare" />
  <Setting Name="ComponentName" Value="quickgraph" />
  <Setting Name="VersionNumber" Value="3.*" />
  <Setting Name="IncludeFilter" Value="*.dll" />
  <Setting Name="FolderMappings"
           Value="SourceOffset=SourceSubfolder,LocalOffset=LocalSubfolder;..."/>
</Settings>
```

Listing 4 - Valid file share settings

When a recent build result should be used as a dependency it can be specified with the use of the following settings: *TeamProjectName*, *BuildDefinition*, *BuildNumber*, *BuildStatus*, *BuildQuality*, *IncludeFilter*, *FolderMappings* and *RelativeOutputPath*

A build result is specified by either the *BuildNumber* or a combination of *BuildStatus* and/or *BuildQuality*. For the build status and the build quality several values can be specified via comma separated values (See Listing 5).

```
<Settings Type="BuildResultSettings">
  <Setting Name="TeamProjectName" Value="TestFramework" />
  <Setting Name="BuildDefinition" Value="TestFramework.Main" />
  <Setting Name="BuildStatus" Value="Partially succeded,Succeeded" />
  <Setting Name="BuildQuality" Value="Initial Test Passed,Lab Test Passed"/>
  <Setting Name="IncludeFilter" Value="Test*.log;*.dll"/>
  <Setting Name="FolderMappings"
           Value="SourceOffset=BuildSubfolder,LocalOffset=LocalSubfolder;..."/>
</Settings>
```

Listing 5 - Valid build result settings

In case a Team Project in the Team Foundation Server should be used as a repository, a dependency can be specified by using the following settings: *BinaryTeamProjectCollectionUrl*, *BinaryRepositoryTeamProject*, ComponentName, VersionNumber and RelativeOutputPath

The repository can be specified via the *BinaryTeamProjectCollectionUrl* and the *BinaryRepositoryTeamProject* setting. In case of the repository the output path must be specified via the *RelativeOutputPath* setting (See Listing 6).

```
<Settings Type="BinaryRepositorySettings">
  <Setting Name="ComponentName" Value="PaketXY" />
  <Setting Name="VersionNumber" Value="1.1.0.16"/>
  <Setting Name="RelativeOutputPath" Value="Bin\Package\PaketXY"/>
  <Setting Name="BinaryTeamProjectCollectionUrl"
           Value="http://nbk021:8080/tfs/defaultcollection" />
  <Setting Name="BinaryRepositoryTeamProject" Value="000_test_repository"/>
</Settings>
```

Listing 6 - Valid binary repository settings

Additionally, the default output path can be overwritten for all providers by specifying the local root path relative to the dependency definition file via the *RelativeOutputPath* setting (See Listing 7).

```
<Setting Name="RelativeOutputPath" Value="Lib\quickgraph" />
```

Listing 7 - Specify a output folder

When using the *RelativeOutputPath* setting the path is interpreted as a path relative to the local dependency definition file location. In case folder mappings are defined by the *FolderMappings* setting, the local offset paths are interpreted relative to the relative output path.

For example if a component **TestFramework** is defined in a dependency definition file *C:\Source\component.targets* and has set a relative output path `"..\Bin\Package\TestFramework"` and folder mappings `"SourceOffset=Include,LocalOffset=..\..\..\Source\Include"`, the *Include* folder of the component will be placed in *C:\Source\Include*.

Repository structure requirements

For components to be detectable by the providers the structure of a repository has to follow pre-defined conditions. The following paragraphs explain the provider specific requirements.

FileShare Provider

Components inside a file share have to be placed in a pre-defined way to be detectable by the FileShare provider.

1. A directory with representing the component name has to be created for every component in the root directory of the file share
2. Each version of a component is placed inside the component directory (The folder name must equal the component version)
3. Component files must to be placed in the version folder together with a dependency definition file

Figure 12 shows the folder structure for a component **Other3rdPartyLib** in version **1.1.0.0** on a network file share.

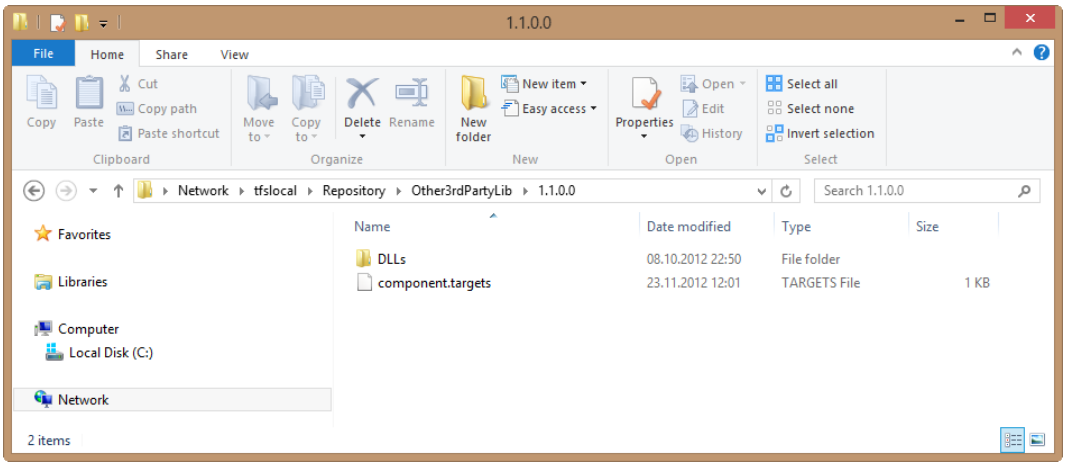


Figure 12 - File share component folder structure

BinaryRepository Provider

Components inside a binary repository have to be placed in a pre-defined way similar to the *FileShare* provider. The same folder structure and the presence of a dependency definition file apply for this provider.

Figure 13 shows the source control folder structure for a component **Component B** in version **1.1.0.0**.

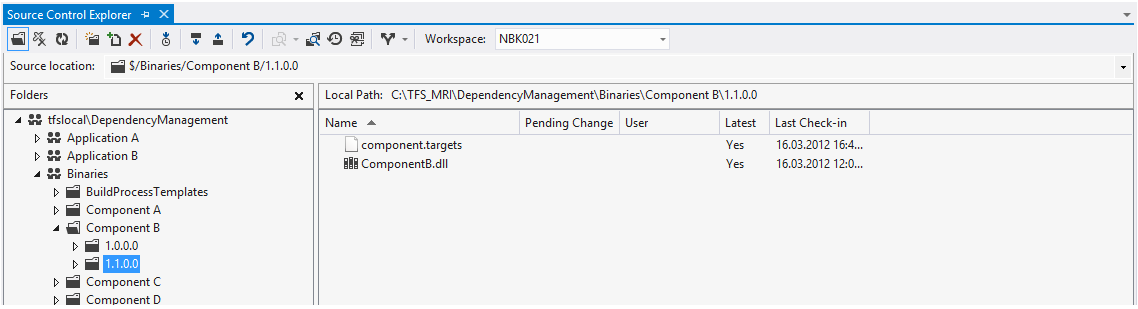


Figure 13 - BinaryRepository folder structure

SourceControl Provider

Components inside source control have to contain a dependency definition file in the root folder of the component.

BuildResult Provider

The BuildResult provider does not define any requirements. Every folder representing a build result component in the Drop Location represents a valid component. The creation of a dependency definition file is optional.

Enabling Visual Editor

Visual Editor should be the default editor for editing dependency definition files. In case the default editor is changed Visual Editor can be reset to be the default editor by clicking the **View With …** context menu on a dependency definition file and click on Set as Default when *AIT Dependency Editor* is selected (See Figure 14).

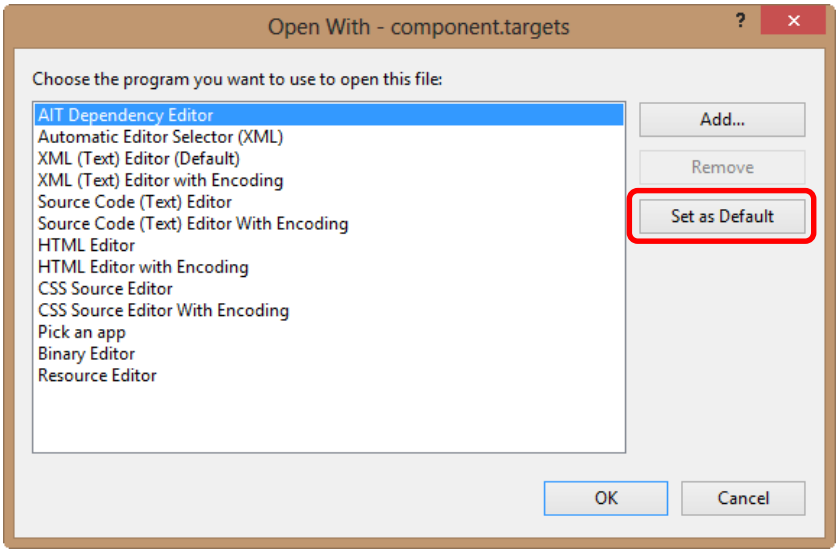


Figure 14 - Set Visual Editor as Default Editor

Enabling IntelliSense Support

To enable IntelliSense support the developer must copy the **AITDependency.xsd** and **AITDependencySchemaCatalog.xml** files from the Dependency Manager Extension installation directory to the Visual Studio Installation directory.

The Dependency Manager Extension installation directory is based on the developer's application data directory. To open the installation directory in Windows Explorer go to the directory *%LOCALAPPDATA%\Microsoft\VisualStudio\11.0\Extensions*. The *Extensions* folder contains all installed extensions which are randomly named. The **AIT Dependency Manager** extension folder contains the two files to copy into the Visual Studio installation directory (See Figure 15).

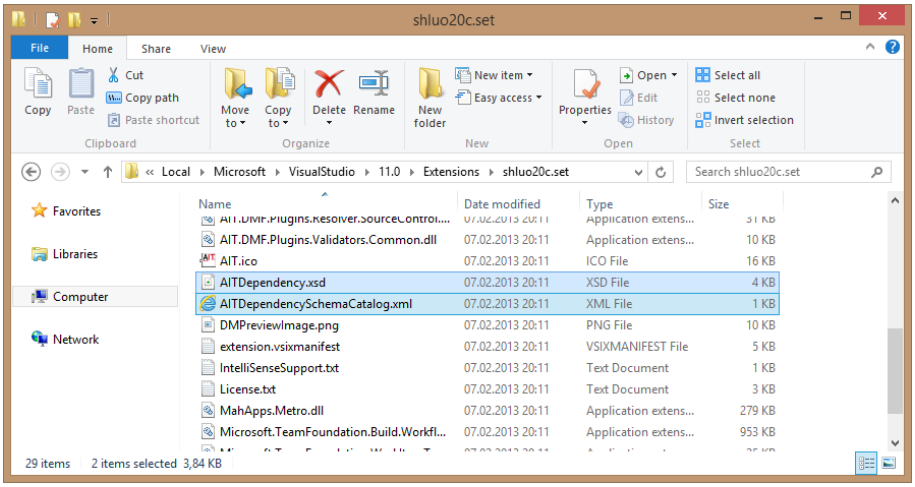


Figure 15 - Dependency Manager VSIX installation directory

The Visual Studio Installation directory depends on the version of Windows is used:

- *C:\Program Files (x86)\Microsoft Visual Studio 11.0\Xml\Schemas* (For Windows 64-bit (x64) installations)
- *C:\Program Files\Microsoft Visual Studio 11.0\Xml\Schemas* (For Windows 32-bit (x86) installations)

After Visual Studio is restarted IntelliSense Support is enabled when editing the dependency definition file.

Known Issues

Visual Editor is Default Editor for MSBuild Targets

Visual Editor opens any file with an .targets extension. Visual Studio Extensions only supports to register an editor based on an extension and not based on a filename pattern.

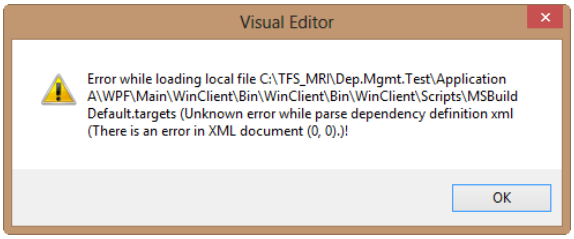


Figure 16 - Visual Editor opens non dependency definition files

Simultaneous calls on a single dependency definition file

Multiple calls of *Get Dependencies* and/or *Clean Dependencies* on the same dependency definition file can cause non-deterministic behavior.

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