War Setup 3

Users Manual

By Jarle (jgaa) Aase

About this manual

Author: Jarle Aase (jgaa@jgaa.com) http://www.jgaa.com/

Product homepage: http://warsetup.jgaa.com

Copyright: Copyright 2007 – 2009 by Jarle Aase. All rights reserved.

This manual is licensed under the terms of the "Common Public License 1.0" or the "GNU Free Documentation License 1.2" after your own choice.

- http://www.opensource.org/licenses/cpl1.0.php
- http://www.fsf.org/licensing/licenses/fdl.html

Please report spelling errors, subjects that are missing, errors in the content or suggestions on how this manual can be improved to jgaa@jgaa.com.

Published:

- Version 1.09 – February 17th 2009 [version 3.13 of War Setup]
- Version 1.08 – February 6th 2009 [version 3.12 of War Setup]
- Version 1.07 – January 30th 2009 [version 3.11 of War Setup]
- Version 1.06 – January 16th 2009 (Upgrade to new Wix 3 Beta Final + bug-fixes and new features) [version 3.10 of War Setup]
- Version 1.05 – October 27th 2007 (new features)
- Version 1.04 – July 26th 2007 (new options)
- Version 1.03 – May 28th 2007 (new features)
- Version 1.02 – May 17th 2007 (updated with more details)
- Version 1.01 – May 15th 2007 (spelling errors, minor changes)
- Version 1.00 – May 14th 2007

Table of contents

1. About War Setup

1. History

I wrote War Setup version 1 in 1996 to make the War Ftp Daemon install package. At that time there was no decent, free install tools for Windows. Microsoft had an Install API and some guidelines, but the library was haunted by bugs, so I ended up writing most of the low-level code myself. War Install was fast, very small, and made packaging manageable. It also supported dll plugins, but I think only the War FTP Daemon ever used it. Version 2 followed version 1, and the installer served all my needs for free and commercial products for over a decade.

Illustration 1: The obsolete War Setup version 2 Project dialog

Today, the old way of installing programs is obsolete. Microsoft have new guidelines, and recommend us to use the Windows Installer to make installations foolproof for all versions of Windows. I tried to find a suitable program to replace War Setup, but the nearest I got was the Open Source Installer toolkit "Wix" from Microsoft. There was however too much details to take care about. I was also looking for a program that could handle my "War Lib" C++ class library, including folders with Doxygen-generated html-files. Those files cannot be added manually. They have to be added automatically, as their names change as the documentation is updated. I ended up writing a new major version of War Install from scratch. The previous versions was written in C and C++. Version 3 is written in C#. It use xml files for it's project definitions, and Wix to build the .msi file.

2. Design goals

The design goals was to make something that is robust, flexible and *very* easy to use and maintain. Some people say that the install-package is just as important as the software you install – and that the developers should maintain and update an install project from the very start of the development. I agree – but I don't have the time to do that. I have to focus on my applications – and then create a setup project with a minimum time and effort when the program is ready. War Setup makes it easy to create and maintain simple as well as advanced setup packages. It handles most of the dirty details, and the user don't need to read books about Windows Installer in order to use it (like some other installers do).

The intended use for WarSetup is to create:

1. **Installers for "Normal" Windows applications** (one or a few executable files, user-documentation, may be some additional resources like dll-files or fonts and probably some merge-modules, like the Microsoft C++ runtime system). These projects shall typically take less than 5 minutes to make.

2. **Installers for Windows services**. Same as above, but some of the "applications" are native Windows services, and installers as such.

3. **Installers for C or C++ libraries**. Anything from a few to tens of thousands of files. In stead of adding files, you add directories, and use regular expressions as include and exclude-patterns to specify what files you want to install. The libraries can integrate automatically with Microsoft Visual Studio 2003 and 2005 (I don't have VS 2008 yet – when I get it, I will add support for this as well. If you can't wait – you may want to sponsor the project by purchasing one for me). These projects shall typically take less than an hour to create. (An alternative installer for a 10.000 files library by scripting can easily take days or even weeks to create – and then turn out to be a nightmare to maintain).

3. Future features

War Setup is written to fit my needs. I will add new features as I see fit. I will add patches from other developers if they provide useful additions and good code quality. If you need some feature for your Open Source project, I might add it if I like the idea. If you need some feature for your commercial program, you can hire me to add it. *That might very well be significantly cheaper than purchasing a commercial install package*.

4. Feedback, bug-reports and support

If you find WarSetup useful and use it, feel free to send some feedback to jgaa@jgaa.com. It is a lot more motivating to work on Open Source projects that are used by others, than projects that are mostly used in-house. Please submit **suggestions** and **bug-reports** to the appropriate **trackers** at http://www.sourceforge.net/projects/WarSetup.

2.Installation and preparation

Just run the install-program to install War Setup. In order to build .msi files or merge-modules. Wix 3.0 must be installed. You can download Wix from wix.sourceforge.net.

The Tools/Options menu allows you to specify where certain directories are on your machine. But normally War Setup will figure this out by itself.

War Setup require the latest .NET framework 2. If you don't already have this, you can download it from Windows Update. (The -NET framework is *not* required on the target machines where the generated installation packages are used.)

3.How it works
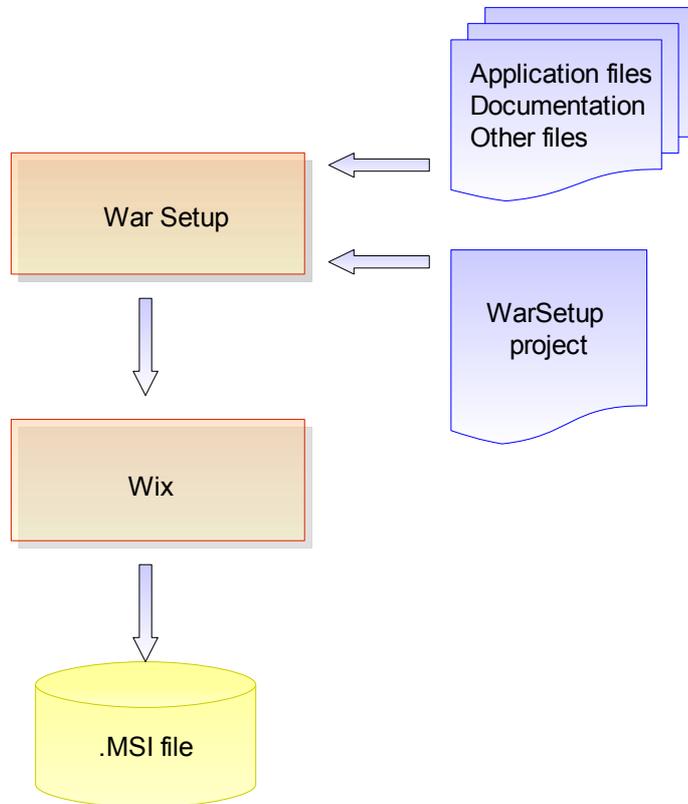
Drawing 1: Flowchart for War Setup

You create a War Setup project and add all the files, directories, shortcuts, merge-modules etc. you need. You can add a hierarchical feature-tree if you wish. You specify the license for the package, and what GUI (Graphical User Interface or "User Experience") you want to use. You can use the Wix stock graphics, or add your own images. Then you press the "Build Target"-button on the tool-bar, War Setup creates a Wix project, compiles it, and if everything is all right, the .msi-file is built.

4. Reference

This section will walk you trough the dialogs in War Setup 3

1. The Project dialog

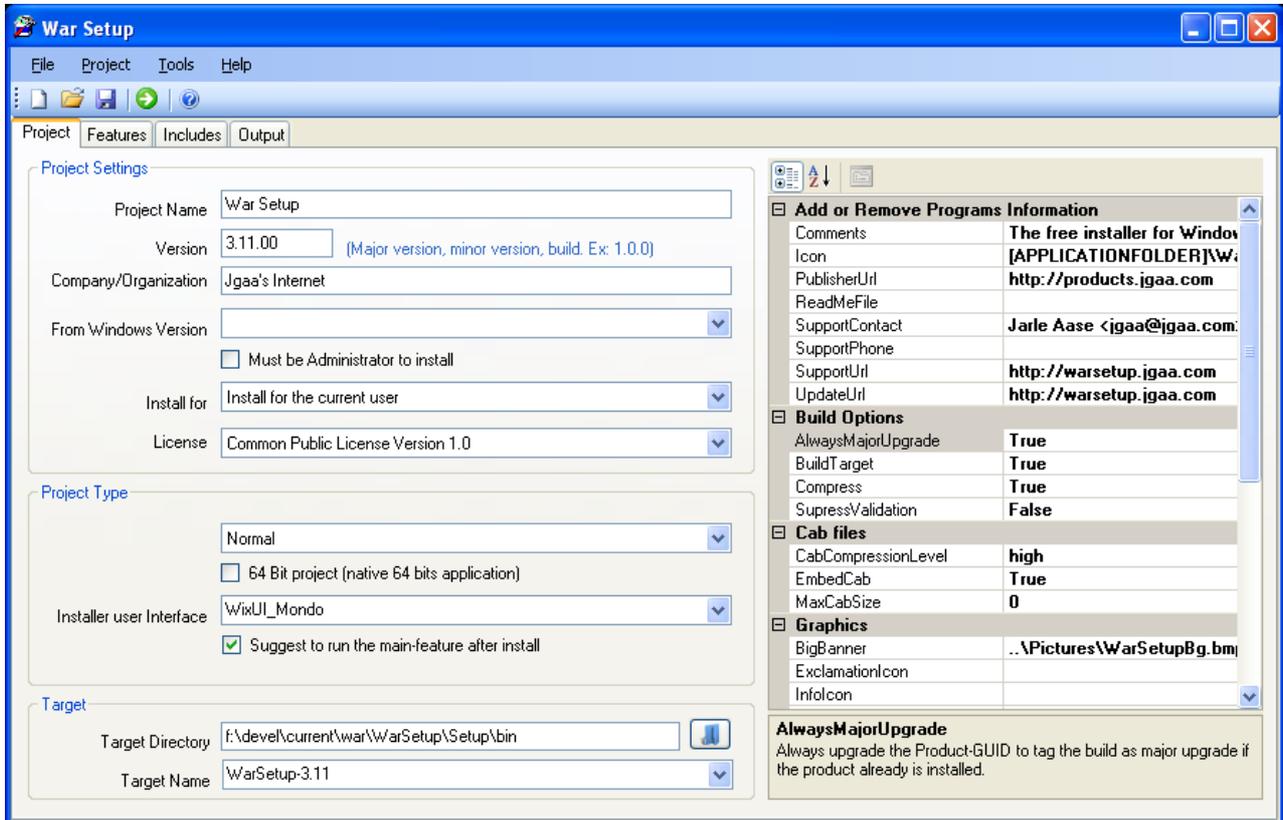This is the main part of war Install.

Illustration 2: The Project dialog in War Setup

- **Project Name**: The name of your project. This will appear in the installer as the name of the application that are being installed.

- **Version**: The version of the product. The version is defined by Microsoft as *Major* version number, *Minor* version number, *Build* number. While versions in resource files under Windows have 4 fields, only 3 are allowed by Windows Installer. If you use a major and minor version number, you can use 0 as build-number. You can also increment the build-number each time you build a release-version of your application. Example: 1.0.223.

- **Company/Organization**: The publisher of the software.

- **From Windows Version**: If set, the package can only be installed on the specified or later versions of windows. This is normally used when a package use API functions that are unsupported on previous versions of Windows.

- **Install for**: A windows application can be targeted for the current user or for all users. If you want the user himself to decide, you must use the WixUI_Advanced interface. (That supports unprivileged installs under Windows Vista).

- **License**: The license for your application. War Setup supplies some of the most common licenses for Open Source projects. You can add your own license in the War Setup\License directory. The License must be a text-document in rtf format, with a ".rtf" extension. Wordpad.exe is probably the best suited application to create the license-file. (Microsoft Word may create rtf-code that Windows Installer cannot properly format). You can also add your own license to an optional directory and use the *LicensePath* option (Project properties) to enable licenses from this directory.

- **Project Type**:

  o *Normal*: A normal install project (.MSI file).

  o *Merge Module*: A install package without a user-interface, to be included in a Normal package. Merge Modules are used to provide a uniform installation of shared components. Visual Studio ships with a number of Merge Modules, and most serious software component suppliers also provide Merge Modules. This mode let you create your own merge-modules.

- **64 bit project**: Make a native 64-bit project. This feature is currently highly experimental.

- **Installer user Interface**: The appearance of the installer. Currently War Install only support the user interfaces from Wix.

  o **WixUI_Mondo** includes the full set of dialogs (hence "Mondo"): welcome, license agreement, setup type (typical, custom, and complete), feature customization, directory browse, and disk cost. Maintenance-mode dialogs are also included. Use WixUI_Mondo when you have some of your product's features aren't installed by default and there's a meaningful difference between typical and complete installs.

  o **WixUI_FeatureTree** is a simpler version of WixUI_Mondo that omits the setup type dialog. Instead, the user goes directly from the license agreement dialog to the feature customization dialog. WixUI_FeatureTree is more appropriate than WixUI_Mondo when your product installs all features by default.

  o **WixUI_InstallDir** doesn't allow the user to choose features but adds a dialog to let the user choose a directory where the product will be installed.

  o **WixUI_Minimal** is the most spartan of the WixUI stock dialog sets. Its sole dialog combines the welcome and license-agreement dialogs and omits the feature customization dialog. WixUI_Minimal is appropriate when your product has no optional features.

  o **WixUI_Advanced** provides the option of a one-click install like

WixUI_Minimal but allows directory and feature selection like other sets. **Note that WixUI_Advanced is still in active development. When complete, it will support per-user/per-machine choice and allow for compile-time selection of custom dialogs.**

- **Target Directory**: The directory where the target-file will be created.

- **Target Name**: The name of the target file (without the file extension name). The Extension is added by War Install when the target is built, depending on the *Project Type*.

- **Build Options:**

  o *AlwaysMajorUpgrade*: Each build is tagged as a "major upgrade" to Windows Installer. See the chapter How to upgrade your product.

  o *BuildTarget*: If False, only the ,wix file is created. This is useful if you are deploying a large project, and want to verify the .wix file manually before time-consuming merge-modules are included (The Crystal Reports merge-module for .NET 2005 alone takes four minutes to process on my laptop).

  o *Compress*: Option in WiX / Microsoft Installer. I ldon't think it's a good idea to use any other value than True. If you want to use an external program to pack the .msi, or if you want to test the .msi before wasting time to compress it, use the CabFiles / CabCompressionLevel attribute.

  o *SupressValidiation*: Some third-party merge-modules will not pass validation (Business Objects are notorious!). Use this attribute to go around the problem. (The build will normally work, even if it don't validate).

- **Graphics**: The graphics properties allows you to use your own images and icons in the user- interface. Just add the paths to the image-files you want to use. The files must be at the size specified in the property-explanation displayed when you select a property.

- **Interface**:

  o *LaunchAppText:* Use this property to define your own text on the "Launch application" checkbox on the final UI-page (if the user gets that choice). If this field is blank, a default text is used.

  o *UseWixUI_ErrorProgressText*: Include a reference to WixUI_ErrorProgressText. This is normally required if you use the wix UI module.

  o *UseWixUI*: Include the WixUI module. This includes the WixUI dll when the project is "compiled" with candle.exe. If you use any of the Wix user interfaces, this option *must* be enabled. If you don't know what all this talk about Wix is, just keep it enabled and everything will be fine.

- **Project**:

  o *LicensePath:*An optional path to a directory containing *.rtf file(s) with the Software License Agreement for the package. The Open Source

licenses will still be available, but you can use this option to add your own commercial (or other) license to the list of available licenses.

- o *UseRelativePaths*: If true, paths to files and directories that are dragged into the project are made relative to the project-file (.warsetup file). This is convenient if you change the disk-location of your files. The algorithm that calculate the relative path allows one level of redirection. If project.warsetup is in C:\projects\my_projects\my_project\Setup\, all files dragged from C:\projects\my_projects\my_project will be relative to C:\projects\my_projects\my_project\Setup\.

- **Target Options**:

  - o *CreateUninstallShortcut*: Creates a menu-item "Uninstall [Program Name]" in the Program Files menu. This is accomplished by adding a feature for this in the .msi. The end-user can therefore choose to include or exclude this option.

- **Visual Studio Integration**: See also the chapter about Visual Studio Integration.

  - o *VS2003Integration*: ID of a feature that will enable integration with Visual Studio 2003. Add a feature in the feature-tree with this ID in order to integrate your C++ class library or C library with Visual Studio.

  - o *VS2005Integration*: ID of a feature that will enable integration with Visual Studio 2005. Add a feature in the feature-tree with this ID in order to integrate your C++ class library or C library with Visual Studio.

- **Windows Installer**

  - o *InstallerVersion*: The minimum version of Microsoft Windows Installer required (on the target-machine) to install this package. Visual Studio 2003-projects typically require version 2 (=200). Visual Studio 2005-projects typically require version 3 (=300).

2. The Features dialog

An install package is grouped into one or more "features". The features decide what gets installed based on the "Normal" or "Full" installation options with the WixUI_Mondo interface, and give the user a tree-view of the features to choose from in all but the simplest interfaces.
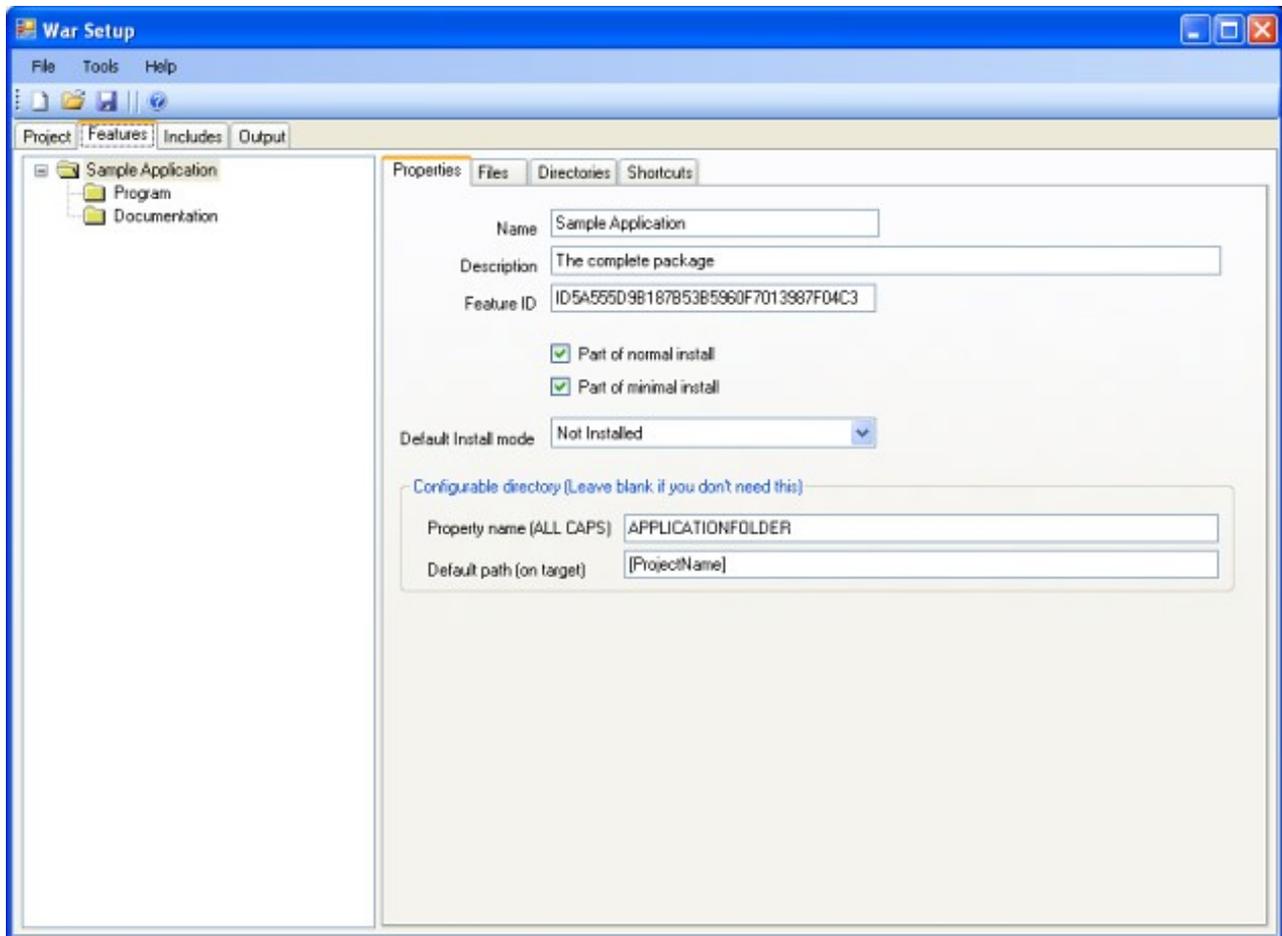

Illustration 3: The Features dialog in War Setup

The features in Windows Installer work like this: If the user select a sub-feature, all parent features will also be installed. So if you need to add some special, optional features, you must place them as a sub-feature of a required feature, or as a root-feature.

War Setup allow you to right-click in the features-tree to add or delete features.
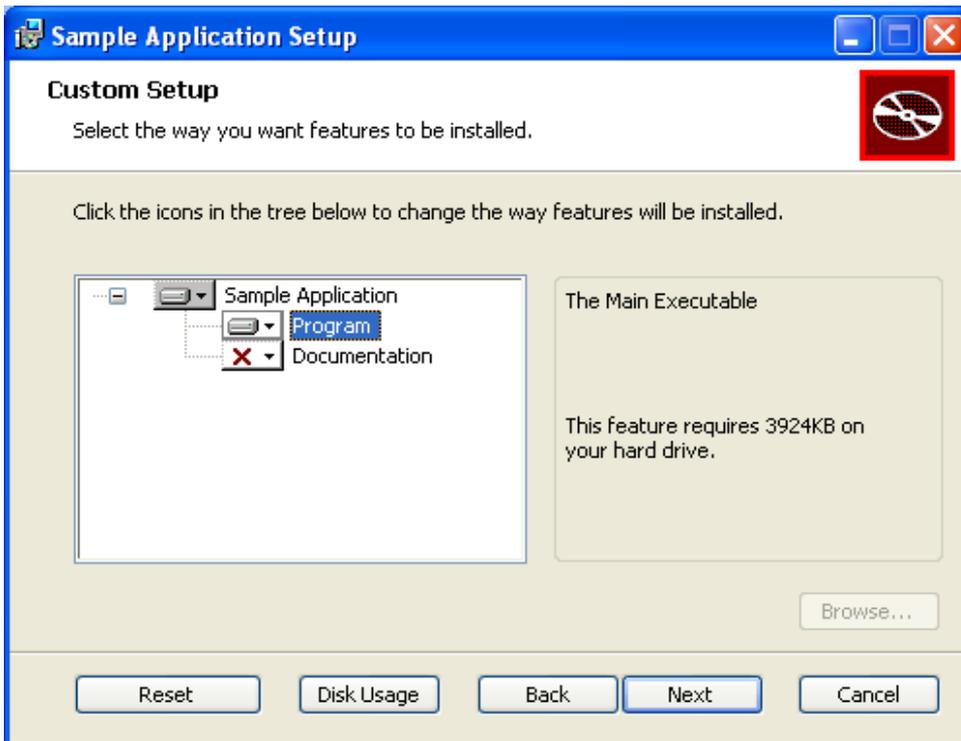
Illustration 4: The features-tree when the package is installed
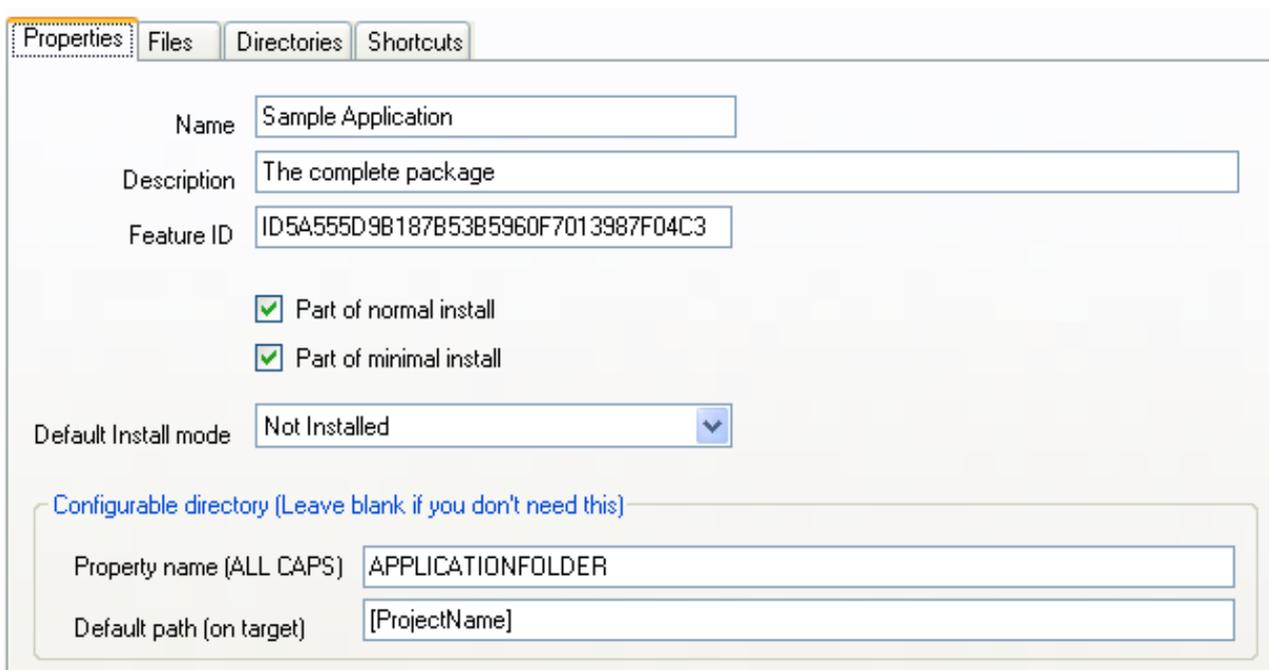
1. Properties

Illustration 5: The Properties dialog under Features

- **Name**: Name of the feature as it will appear in the feature-tree when the user install the package.

- **Description**: A description offered to the user when the feature is selected (see the right pane in Illustration 4: The features-tree when the package is installed).

- **Feature ID**: A unique ID for the feature. Windows installer have some odd constraints for ID names. In short; use the default ID suggested by War Setup, or a name consisting of UPPER letters and digits, beginning with a letter. Unattended, scripted installations can use these names to select what features to install. See the documentation for Windows Installer for more information.

- Part of normal install: If true, this feature will always be installed, unless the user unselect it in the features-tree. (For Installer experts: This feature is Level 3 by default).

- **Part of minimal install**: If true, this feature will always be installed, unless the user deselect it in the features-tree. (For Installer experts: This feature is Level 1 by default).

- *Neither of the above*: (For Installer experts: This feature is Level 1000 by default).

- **Default install mode**: If/How the feature is installed.

  o *Not Installed*: The feature is not installed

  o *Same as parent feature*:

  o *Run from source*: An odd option in Windows Installer to enable the feature, without copying the files. This may be useful for large files where the source is a network share.

  o *Install locally*: Install the feature locally on the target machine. (The normal way of doing it).

- **Configurable directory**: This is an option to allow the user to select the target directory. If this is possible when the package is installed depends on the User Interface that is used. WixUI_Modo allows each feature to have a configurable directory, while the others just allow one, which must have the property name "APPLICATIONFOLDER". War Setup will configure the default root-feature to be the APPLICATIONFOLDER when you start a new project.

  o *Property name*: Must consist of UPPER letters, digits and underscore. The name must start with a letter. The name used can be addressed by any target-file or directory if it is enclosed in square brackets.

  o *Default path*: The default path on the target machine. You can use the special name "[ProjectName]" to use the Project Name as the path-name. The target path is "\Program Files\ Your Organization Name\Default Path". (The actual name of the Program Files folder is decided by the localization and version of Windows).
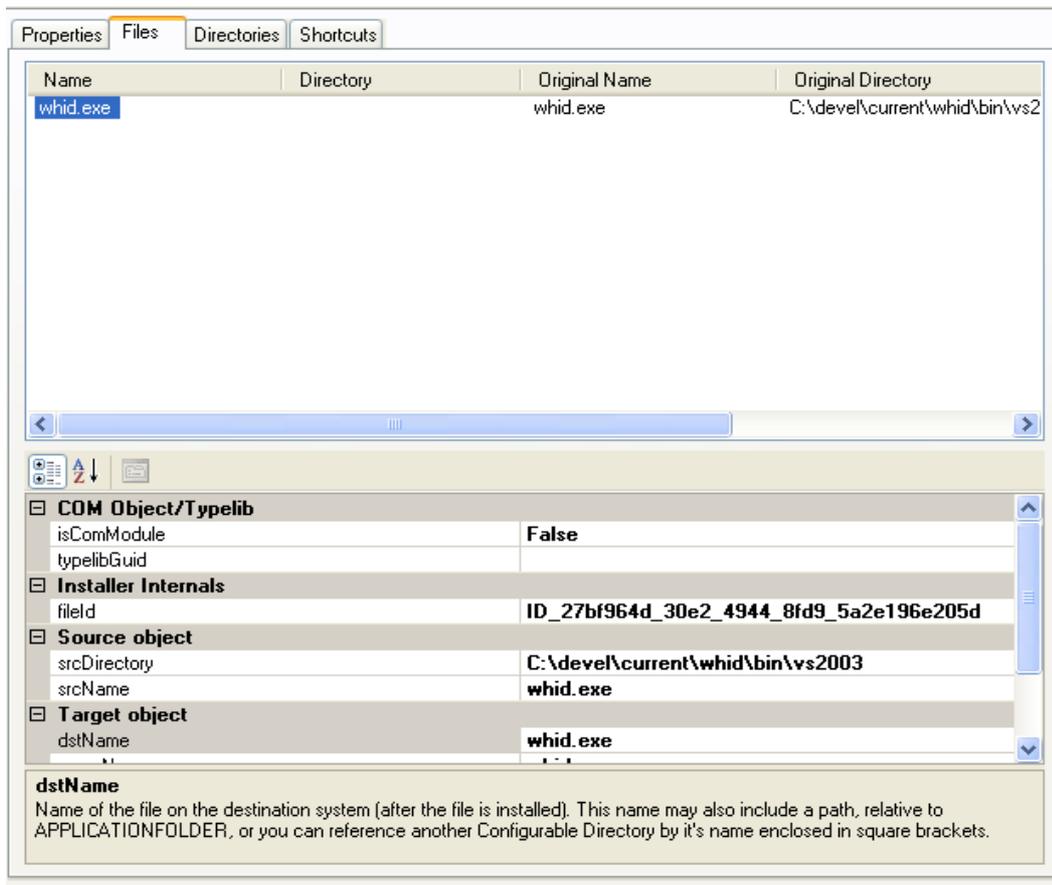
2. Files

Illustration 6: The Files dialog under features

To add files, Right-click and select "Add files" from the pop-up menu, or drag them from Windows Explorer and drop them in the file-list. To delete a file, right click on the file and execute the Delete menu that pops up.

- **COM Object/Typelib**:
    - *isComModule*: True if this file must be registered as a COM/Typelib object.
    - *COM Object/Typelib*: GUID for the COM/Typelib object. I've not figured out how to extract this information automatically yet, so you must find this in the objects source-code or in the registry.
- **Installer Internals**
    - *fileId*: Unique id for the object.
- **Service**: This section allows you to install a program as a Windows

Service. A service is a special kind of a Windows application, that run in the background, totally independent of any logged-on users. If you don't know if your application is a Windows Service, you can safely assume that it is not, and skip these settings. War Setup will install the service as a "OwnProcess" service. It can not install system or kernel drivers (this is a limit in Windows Installer).

- o *cmdLineAruments*: Contains any command line arguments or properties required to run the service.

- o *description*: Sets the description of the service.

- o *errorControl*: Determines what action should be taken on an error. I must admit that I don't know if this applies to setup-errors, or runtime-errors when the service is in normal operation, and that I have no idea what the different options mean. I have not seen these options in any documentation before, when I have developed and deployed Windows services.

- o *interactive*: Whether or not the service interacts with the desktop. Normally it does *not*. It's *very* rare for a properly written service to interact with the desktop. If you have to interact with the desktop from a service, it usually mean that you have no clue about Windows Services.

- o *IsService*: If true, the program is installed as a service. Else, the entire Service section is ignored.

- o *loadOrderGroup*: The load ordering group that this service should be a part of.

- o *Remove*: Specifies whether the service should be removed on install, uninstall or both. This attributes value should be one of the following:
  - ▪ *install*
  - ▪ *Uninstall*
  - ▪ *Both*

- o *serviceName*: The name of the service. This is normally a single-letter word in the English alphabet. See the Services control panel applet for some examples.

- o *startMode*: Determines when the service should be started:
  - ▪ *auto*: The service is started when the machine is booted.
  - ▪ *manual*: the service is started by the operator on demand.
  - ▪ *disabled*: The service is started.

- o *StartWhenInstalled*: If true, the service is started by the installer.

- o *userAccount*: The account under which to start the service. Most services runs as the local "System" account. In that case, leave the userAccount empty.

- o *userPassword*: The password for the account. (The local System account does not have a password).

- o *vital*: The overall install should fail if this service fails to install.
- **Source object**: the file on the source machine. This is the file that is copied into the .msi file.
- **Target Object**: This is the file when it is installed on the target machine.
  - o *dstName*: Name of the file on the destination system (after the file is installed).
  - o *dstPath*: Folder on the destination system. Use square brackets to reference any declared directory. If no path is given, the configurable path to the parent feature is assumed. If no such path exists on any feature up to the root, the [APPLICATIONFOLDER] is used.
  - o *menuName*: Name of the Menu/Desktop shortcut referencing this object. If empty, the file-name will be used.
  - o *Shortcuts*; If true, the file will be referenced by the shortcuts. This means that the file can appear on the Desktop or in the Program Files menu.
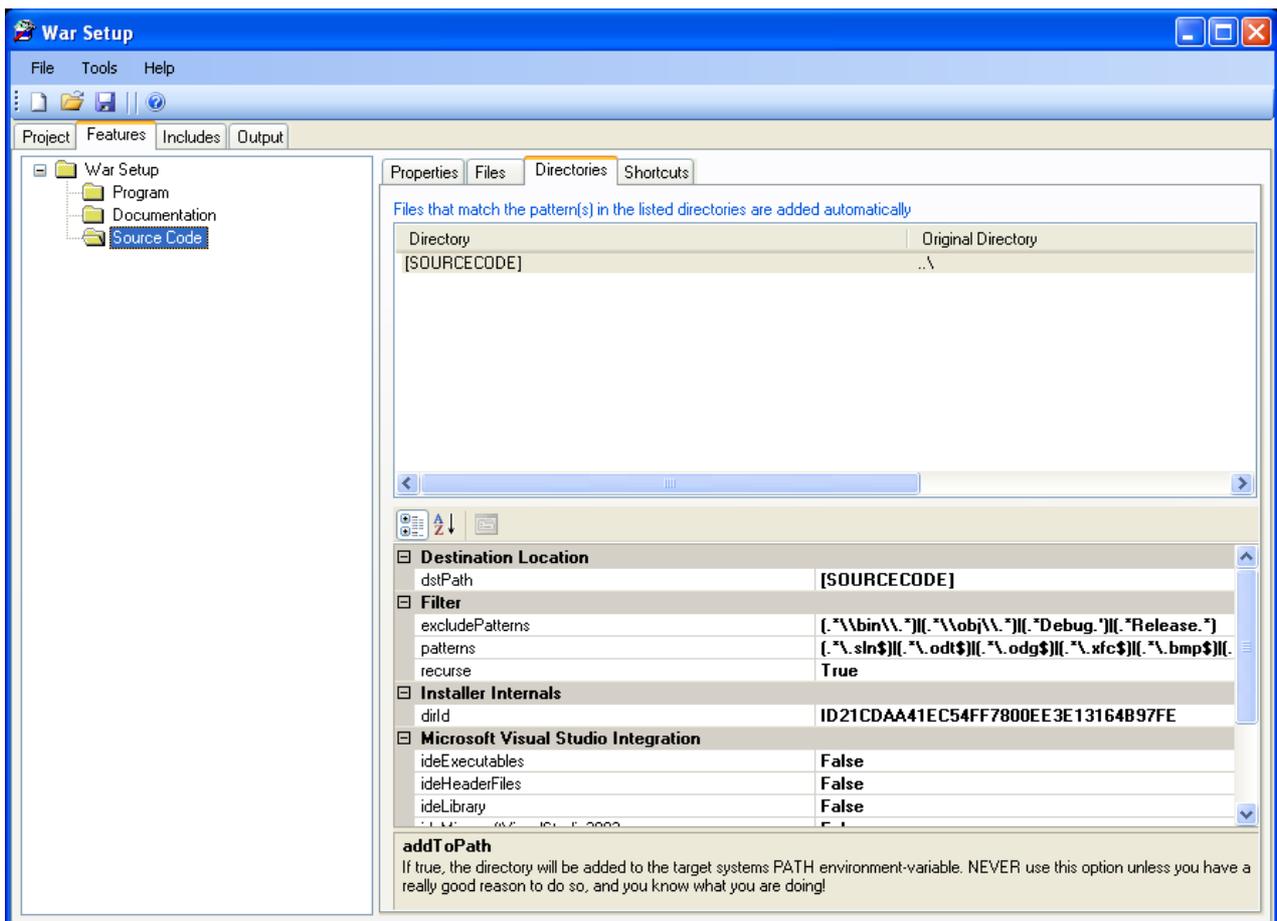
3. Directories

Illustration 7: The Directories dialog under features

Normally when you make an installation package, you will define each single file to install in the Files dialog. This is how most installers work. With the drag

and drop interface, it's also fast and easy to accomplish in War Setup. But there are situations where you have *lots* of files to add, or even lots of files in nested directories. That means lots of work, and that's contradicts with the design goals of War Setup! I also had a special problem with some of my projects, where the documentation was generated with Doxygen (a free C++ class documentation tool), and I had a combination of many files, nested directories, and unpredictable file-names. The answer to these challenges is the Directories feature in War Setup.

Using this feature, you can add whatever files you want from nested directories, based on regular expressions.

To add a directory, right-click in the directory list and select "Add ..." from the pop-up menu, or drag and drop it from Windows Explorer to the directory list. To delete a directory, right click on it and use the pop-up menu.

- **Destination Location:**
  - o *dstPath*: You can specify the target directory for the dir(tree), just as you can for files. Unless you specify a root-target with square-brackets, the APPLICATIONFOLDER is used as root-dir.

- **Filter**:
  - o *excludePatterns:* Patterns to exclude from inclusion, even if the patterns-property below match. This exclusion is meant to keep certain files and directories out of the package; like temporary files and directories, source control files and directories, backup-files etc. The regular expression is matched with all included directories, and the full path of each file in the scanned directories.
  - o *patterns*:A regular expression specifying the files to add. If no patterns is given, no files will be added. (This can be used to offer sub-features for things like Visual Studio integration).. Examples:
    - ■ .* : Add all files
    - ■ .*\.html : Add all "*.html" files.
    - ■ (.*\.h)|(.*\.cpp) : Add all files ending at ".h" or ".cpp".
  - o *Recurse to subdirs Add the target directory and all directories it contains recursively.* If false, subdirectories are ignored.

- **Options**:
  - o *Add to Path*: If true, the directory is added to the PATH environment-variable. *Be careful with this option, as it's easy to create problems on the target-machine if you distribute executables or dlls with the same name as existing files in the target-machines PATH! To many installers add "standard" dll's like zlib.dll, and add it to the machines PATH, in stead of placing it in the target-applications directory. Ideally files like this ought to be installed in the system repository and referenced by the target-application(s) by manifest-file(s).*

- **Microsoft Visual Studio Integration**: These goodies are designed for C++ developers who want to add libraries for use with Visual Studio. See the chapter about Visual Studio Integration.

- **Source Object**:
  - *srcPath*: Path to the directory on the source system.
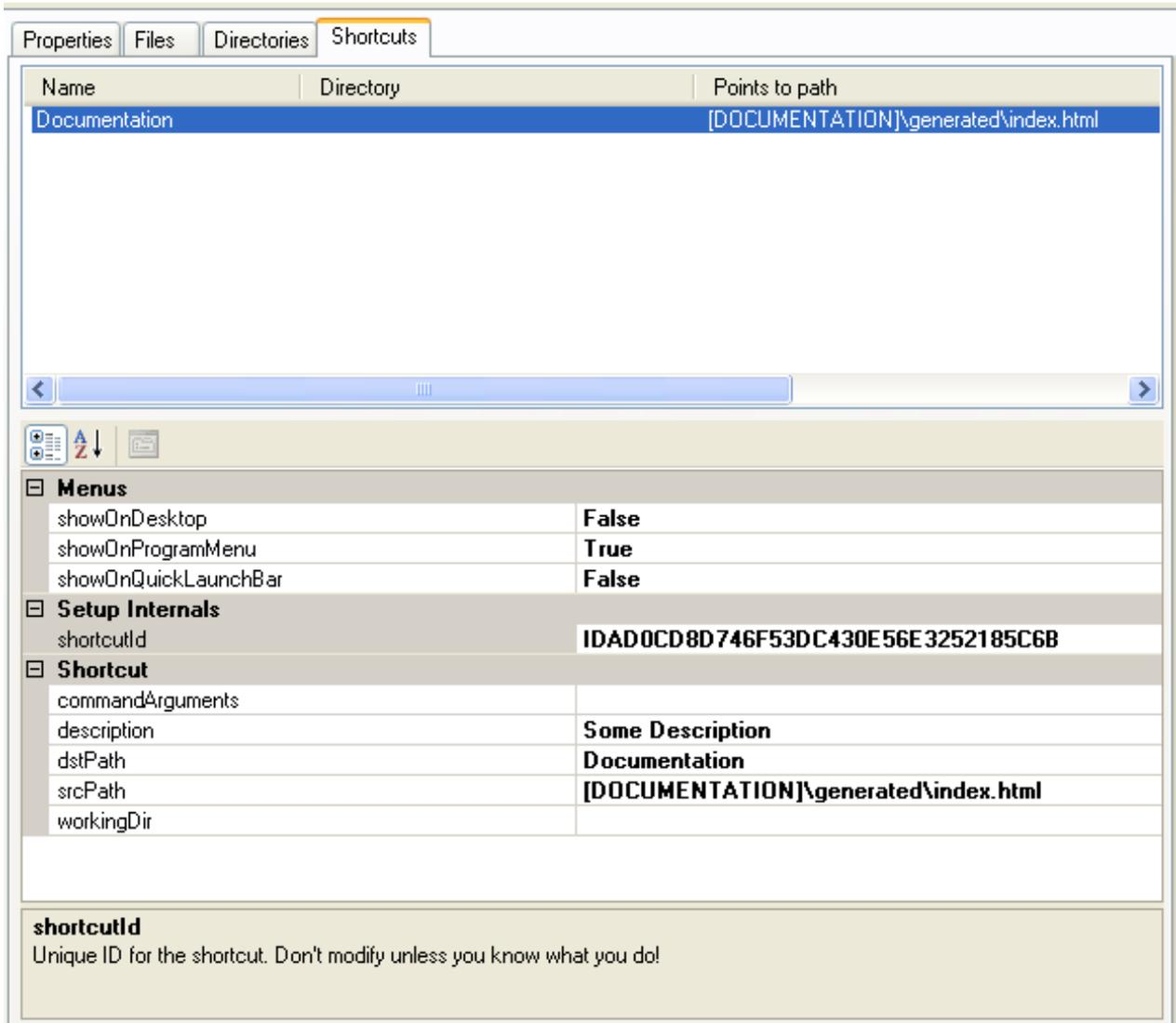
4. Shortcuts

Illustration 8: The Shortcuts dialog under features

Sometimes you need a shortcut to your application to start it with certain arguments, or in a certain directory. You may also want a shortcut to a file that is imported at build-time with the Directory feature in War Setup. The Shortcuts menu allows for both needs. Since these shortcuts often are immediate objects that will be used in the Program menu or on the Desktop, these needs are handled by War Setup.

- Menus:
  - showOnDesktop: Adds a copy of the shortcut to the users Desktop.
  - showOnProgramMenu: Adds a copy of the shortcut to the users

Program menu.

- o showOnQuickLaunchBar: Adds a copy of the shortcut to the users Quick Launch bar.

- Setup Internals:

  - o shortcutId: Unique id for the shortcut. This only applies for the shortcut itself, and not for the copies specified above.

- Shortcut:

  - o commandArguments: Optional command-line arguments for the shortcut.

  - o description: Required description for the shortcut. This is displayed when the mouse hovers above the shortcut on the target machine.

  - o dstPath: Where the shortcut will be installed.

  - o srcPath: Path to the shortcut on the *destination* machine. War Setup will resolve this path when it builds the Wix project file used to compile the target .msi file. I created this feature with Doxygen-generated files in mind (there is always an index.html file). As long as you know the destination path and filename, you can address a shortcut to the file - even if it's not declared until a Directory is scanned during the War Setup build process.

  - o workingDir: An optional directory on the target machine where the application will run.
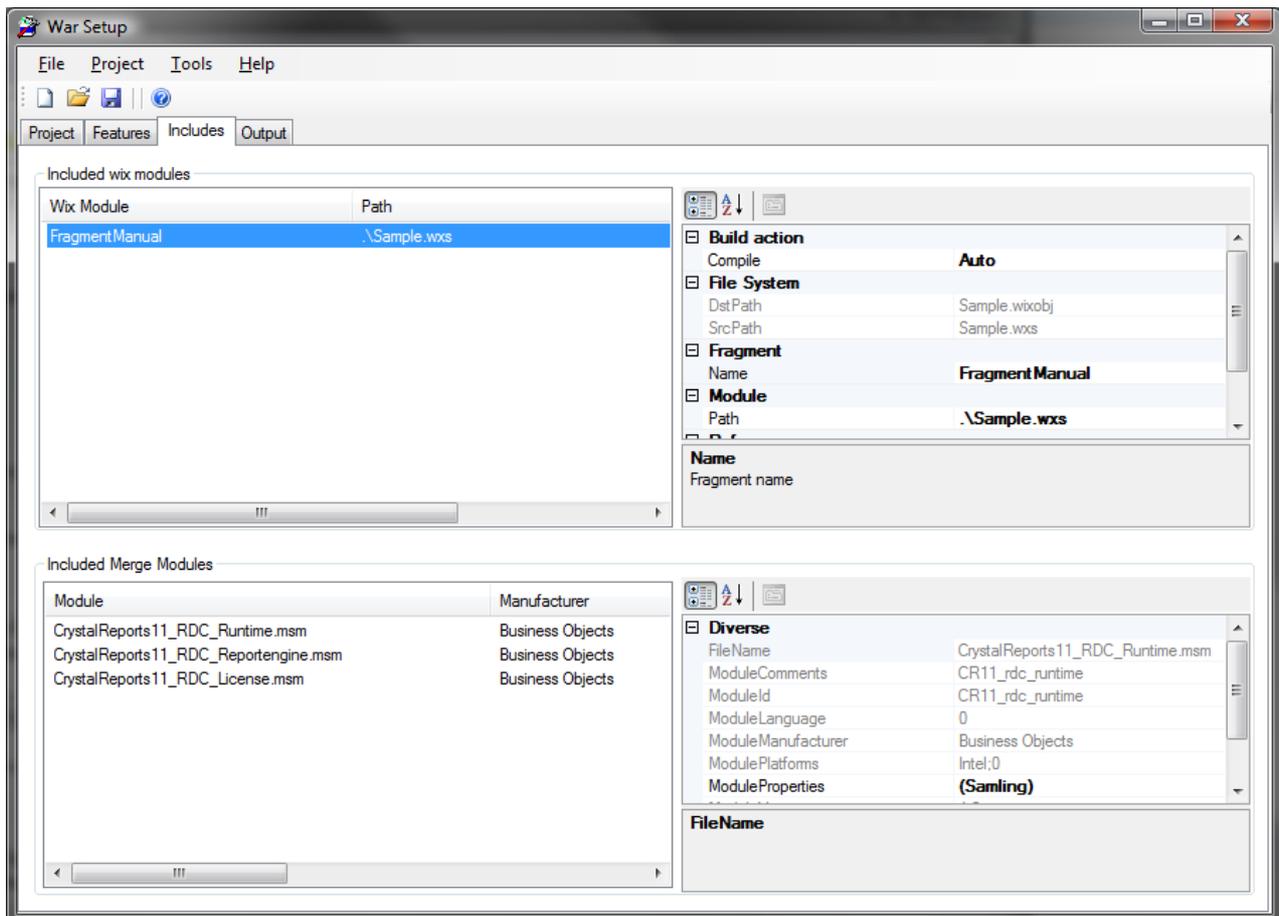
3. The Includes dialog


Illustration 9: The Includes dialog under features

- **Included Wix modules:** Here you can include Wix "fragment"-files to your project. Use the right-click pop-up menu and select "Add ...", or drag & drop the files onto the Wix-module window.
  - o **Build Action**
    - ■ *Compile*: If "yes", WarSetup will attempt to "compile" the module (.wxs-file) each time the setup-project is built. If "No", WarSetup will never attempt to compile the module. If "Auto", WarSetup will attempt to "compile" the module if the source-file (.wxs) is newer than the "object-file" (.wixobj).
  - o **File System**: Here you can see the full name of the source and object-files, as WarSetup sees them. You can use one as the Path, but WarSetup will still consider a source and object file. At least if you tell it to compile the module. Else, only the DstPath is used (included when the project is built).
  - o **Fragment:**
    - ■ *Name***:** The name of the fragment (or one of the fragments) in the file.
  - o **Reference**: Unless your fragment reference an object in your project

(like a feature-id), WarSetup must add a reference in the project to your fragment. This must be to an existing ID in *your* fragment file. You must specify the ID of the object, and it's type. Don't ask me why – that's just how Wix wants it.

   ▪ *ReferenceId*: The ID of the object to reference.

   ▪ *ReferenceType*: Type of the object to add the reference from. You can choose between all valid object-types that can be referenced. If your object is of another type – just add a new object of one of these types. And add a reference. Or add a reference from your fragment to any object in the WarSetup project. If there exist a single cross-reference between your fragment and the project, *all* the objects in your fragment-file will be added.

● **Included Merge Modules**: This dialog allows you to include merge-modules globally into the project. By *globally*, I mean that the modules are independent of what features the user install. You can use the right-click pop-up menu or drag and drop merge modules to the list from Windows Explorer. If you right-click in the modules-list, you get the option to open the merge-modules folder with Windows Explorer.
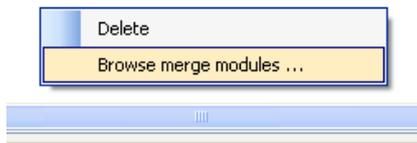
Some Merge-modules have options that can be set by the installer. War Setup will automatically detect these options, and allow you to set the values. That means that ie. the Crystal Reports runtime can be added as a Merge-module, and the required drivers and the serial-number can be set.
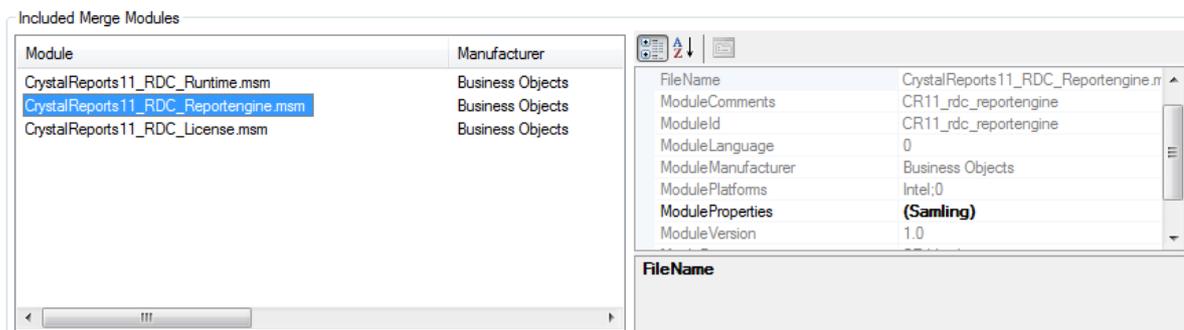

Just click on the "collection" property ("Samling" in the image above, as I use a Norwegian version of Windows) to get the full list of properties.
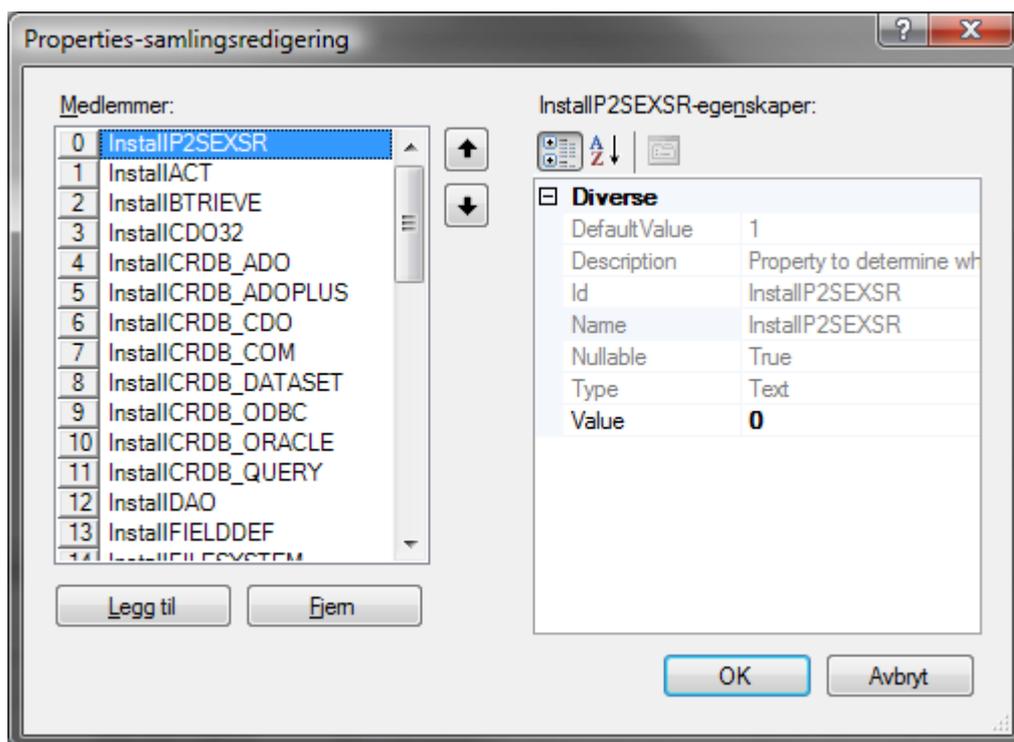
Now you can browse trough the options and set the values.

The Merge-modules are scanned by War Setup when they are added to the project. If you get a newer version of a Merge-module, and the name is the same, you should refresh the information in the War Setup project by right-clicking on it and choosing "Reload properties" on the Pop-up menu.
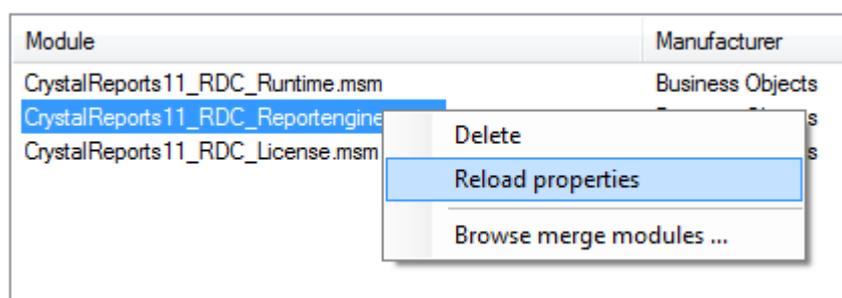
4. The Output dialog

The Output dialog simply dumps the output from the Wix programs. That way you can look at the warnings and error-messages that occur. The commands that are executed are also listed with the complete set of command-line options. That's useful if you are merely using War Setup to create a skeleton Wix-project.

5. Building an install project

1. A typical application project:

1. Start War Setup

2. Fill in the project name, version, Organization and other fields in the Project dialog.

3. Move to the Features dialog. Select the "Program" feature and the "Files" dialog. Open Windows explorer, find the application file(s) and drag them into the file-list. Select the target program in the list and check the shortcutInProgramFilesMenu property. (Most users do not appreciate that the programs they install appear on the desktop or in the Quick Launch bar).

4. Select the Documentation feature and drag over the documentation.

5. Select Includes and add (drag and drop) the merge-modules required by your application. (You can right-click on the merge-modules list and open Windows Explorer in the merge-modules directory from the pop-up menu).

6. Select the Project dialog once again and press the "Build Target" button on the tool bar. If the build fails with some odd verification-error, you can check the "Suppress validation of target" box and try again.

2. A typical C library

1. Start War Setup

2. Fill in the project name, version, Organization and other fields in the Project dialog.

3. Move to the Features dialog. Select the "Program" feature. Change the name to "Library".

4. Drop the library and header files to the Files list, or add the directories for the library and header files to the Directory-list with appropriate regex patterns. Mark the directories for what version of Visual Studio you want to integrate with (if you want to integrate with Visual Studio).

5. Add documentation

6. Follow the guidelines above to build the .msi file.

6. Visual Studio Integration

Microsoft Visual Studio is defacto standard IDE for professional C++ windows development. Making an install-package for a C/C++ library has until now been non-trivial. The reason is that the user-settings for header and library paths in Visual Studio is kept in a non-standard format that Windows Installer cannot handle. War Setup contains a Wix/Windows Installer plug-in for this purpose. It is enabled automatically if you check one of the versions of Visual Studio under the Directories dialog. The paths will be added when the feature is installed, and removed when it is uninstalled.

You can specify if the directory contains header-files, library-files and Executables (like code-generators). These selections does *not* affect the files that are installed (that's filtered by the regex pattern). It simply tells War Setup if the the path to the directory on the target machine should be added to the appropriate Directory setting for the version(s) of Visual Studio that is targeted.

If you check a Visual Studio version, this version of Microsoft Visual Studio *must* be installed on the target machine if the user selects the feature containing this Directory. If you check both IDE's, and the user only have Visual Studio 2003, the installation will fail. You can solve this by *not* using the developer goodies for the feature containing the directories for the library, and then add sub-features for each IDE you support with the same directories, but with empty patterns. That way the directories and files will be installed, but the user can choose if he want to install the IDE integration. (That also mean that users of other developer tools can install your libraries). If you do it this way, you must clear the Visual Studio Integration VS* properties in the Project properties. The advantage of using sub-features like this, is that you can fine-grain the integration, and if you provide several libraries, the user can choose which ones he wants to integrate. The disadvantage is that these features are "hidden" deep down in the feature-tree.

7. Using features

You can also use dedicated features to enable or disable the integration globally.
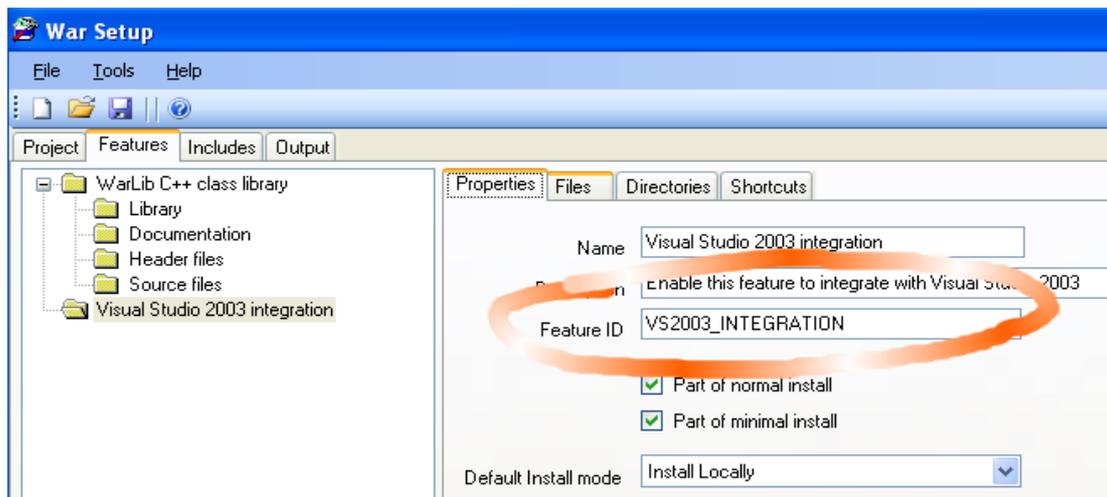
Illustration 10: Example on a Feature used for Visual Studio integration
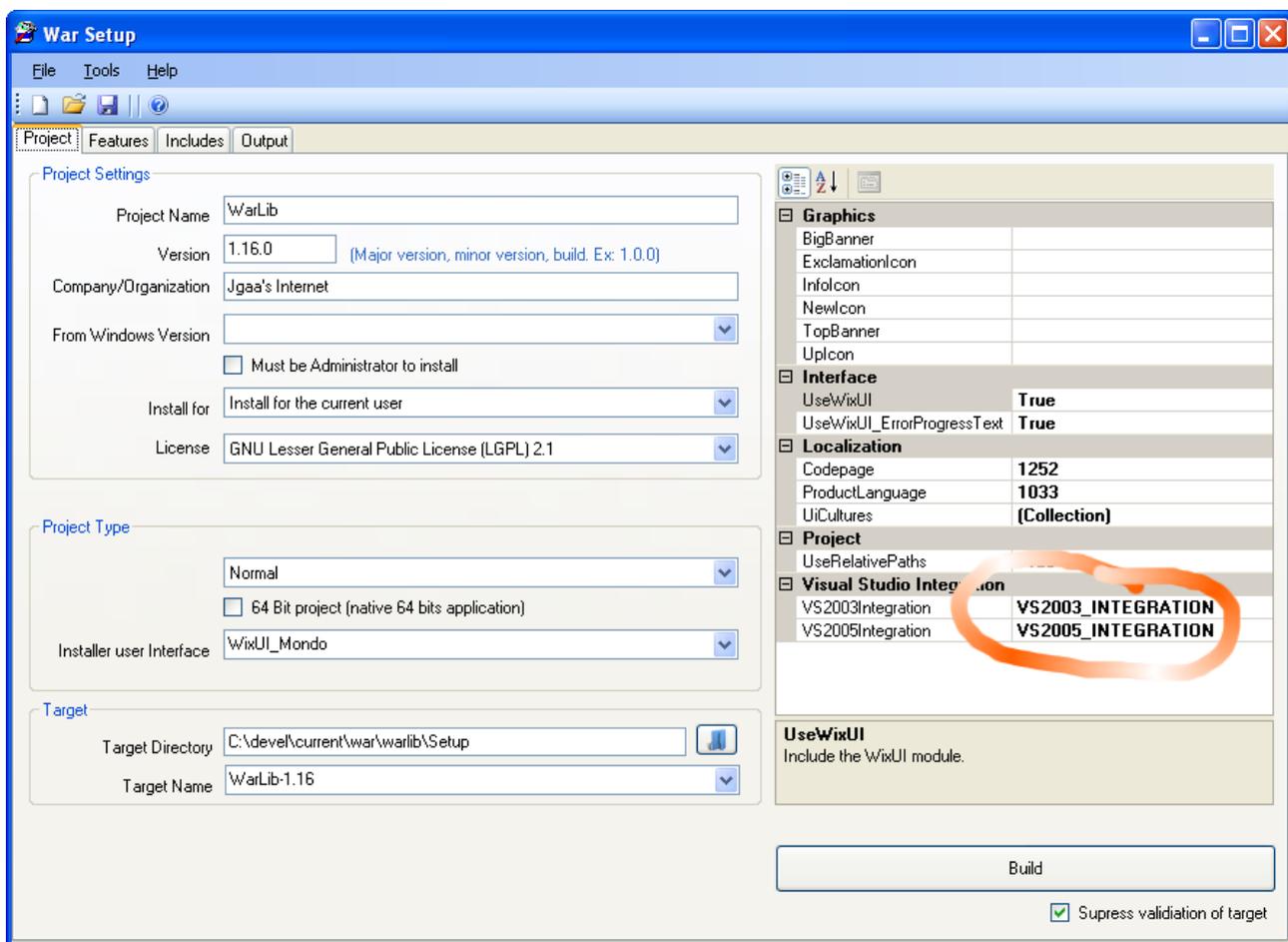
By adding a feature with the same ID as the corresponding VS*Integration property for the project, you can make an installer that will integrate *all* the directories that are tagged for that version of Microsoft Visual Studio if that feature is enabled. The feature can be empty, except for it's properties.

The picture below shows how the user can choose if he wants to have the library integrated with Visual Studio.

Illustration 11: How the Feature-tree appears when the package is installed

8. How to upgrade your product

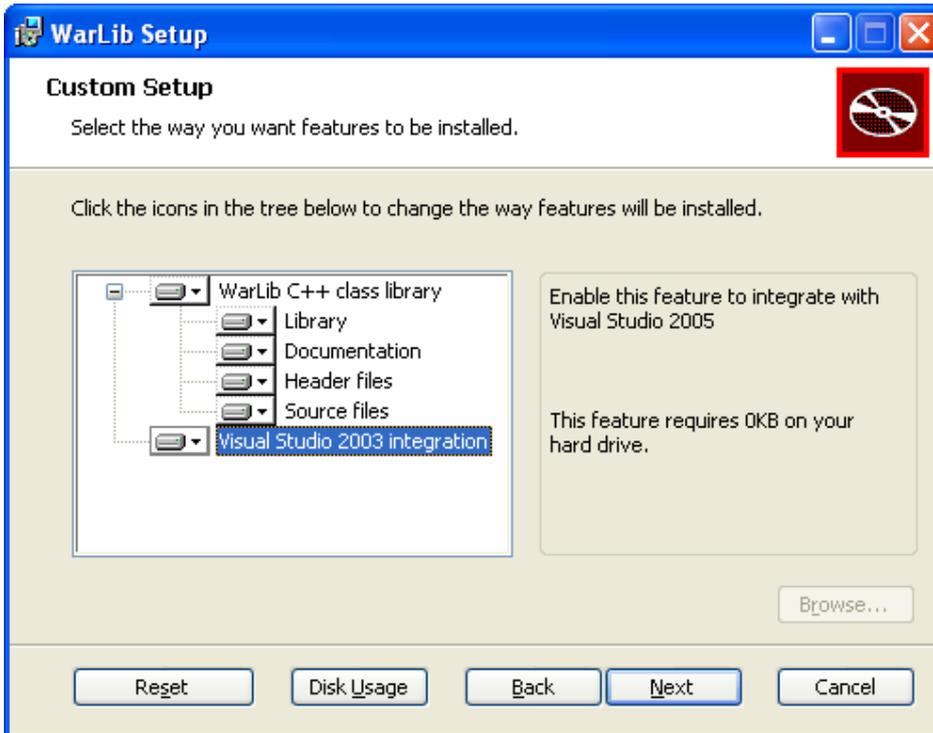Windows installer support several advanced options for upgrades. But they can't be used "out of the box". War Setup will therefore by default tag each build as a "major upgrade", and tell Windows Installer to do a full remove of the existing package if it is already installed. This means that your end-users can use the latest and greatest .msi package to upgrade from any previous version of your product – as well as installing it for the first time! The downside is that the whole package must be downloaded/distributed each time, and that the installation will take a little longer to complete.

If you use fragments to clean up configuration-files or data-files during uninstall, you must make sure that the product is actually being removed and not just removed prior to a new install ("upgrade"). I have not diged deep enough into Windows Installer to determine how to do this. But you will probably get help in the Wix mailing-list if you run into this situation.

9. Target Directories

When you install a simple application, you will normally be installing to one directory, like "C:\Program Files\Your Company\App Name\". War Setup shields you from all the dirty tricks to accomplish this.

In some cases however, you might need to copy files to some other locations. Windows Installer supports a number of system locations, and you can use most of them from War Setup. To install a file to the desktop, simply use [DesktopFolder] as dstPath. The paths can be combined, like [DesktopFolder]\test\testing, to install to a subdirectory two levels under the Desktop. If you do this, remember that most of the directories used by Windows have different names in different localizations of Windows. So if you install something to [ProgramFilesFolder]\Common Files\SpeechEngines, you might get unexpected results on a non-English version of Windows. However, [CommonFilesFolder]\your-app will work on all versions of Windows (at least if your-app is a token not used by anybody but you).

Note: It is generally not recommended to copy files to Windows system folders. If you install your own version of some common .dll in C:\Windows\System32, you can easily end up breaking other installed applications. If possible, you should copy any such dll's to the same directory as your application. If you install True Type fonts, all you have to do is to mark the file as a True Type Font, and War Setup will make sure the file gets to the correct location.

The following tokens are recognized:

Name	Description
[AdminToolsFolder]	The AdminToolsFolder contains the full path to the file system directory that stores administrative tools for a user. Microsoft Management Console (MMC) saves customized consoles to this directory and roams with the user. If the ALLUSERS Property is set, this property points to the file system directory that contains the administrative tools for all users of the computer.
[AppDataFolder]	The installer sets the value of the AppDataFolder property to the full path to the Application Data folder for the current user.
[CommonAppDataFolder]	The CommonAppDataFolder property is the full path to the file directory containing application data for all users. A common path is "C:\WINNT\Profiles\All Users\Application Data".
[CommonFilesFolder]	The installer sets the CommonFilesFolder property to the full path to the Common Files folder for the current user. The installer sets this property. For example, on 32-bit Windows, the value may be "C:\Program Files\Common Files". On 64-bit Windows, the value may be "C:\Program Files (x86)\Common Files".
[CommonFiles64Folder]	The installer sets the CommonFiles64Folder property to the full path of the predefined 64-bit Common Files folder. The existing CommonFilesFolder property is set to the corresponding 32-bit folder. The installer sets this property on 64-bit Windows. This property is not used on 32-bit Windows. When using 64-bit

	Windows, the value may be "C:\Program Files\Common Files".
[CommonFilesFolder]	The installer sets the CommonFilesFolder property to the full path to the Common Files folder for the current user. For example, on 32-bit Windows, the value may be "C:\Program Files\Common Files". On 64-bit Windows, the value may be "C:\Program Files (x86)\Common Files".
[DesktopFolder]	The installer sets the DesktopFolder property to the full path to the current user's Desktop folder. If an "All Users" profile exists and the ALLUSERS property is set, then this property is set to the folder in the "All Users" profile. Common values for this property are C:\Winnt\Profiles\[LogonUser]\Desktop\ (Windows NT/Windows 2000) and C:\Windows\Desktop\ (Windows 95 and Windows 98).
[FavoritesFolder]	The installer sets the FavoritesFolder property to the full path of the Favorites folder for the current user.
[FontsFolder]	The installer sets the FontsFolder property to the full path of the system Fonts folder. **Note: If you install true-type fonts, you should set the isTrueTypeFont property for the file to true, and leave the magic to War Setup.**
[LocalAppDataFolder]	The LocalAppDataFolder property is the full path to the file system directory that serves as the data repository for local (non-roaming) applications. A common value for this property is "C:\WINNT\Profiles\username\Local Settings\Application Data".
[MyPicturesFolder]	The MyPicturesFolder property is the full path to the MyPictures folder. A common path is "C:\WINNT\Profiles\username\My Documents\My Pictures".
[PersonalFolder]	The installer sets the PersonalFolder property to the full path to the Personal folder for the current user. Common values for this property are C:\Winnt\Profiles\[LogonUser]\Personal\ (Windows NT/Windows 2000) and C:\My Documents\ (Windows 95 and Windows 98).
[ProgramFiles64Folder]	The installer sets the ProgramFiles64Folder property to the full path of the predefined 64-bit Program Files folder. The existing ProgramFilesFolder property is set to the corresponding 32-bit folder. For example, on 64-bit Windows, the value may be C:\Program Files. This property is not used on 32-bit Windows.
[ProgramFilesFolder]	The installer sets ProgramFilesFolder property to the full path to the Program Files folder. For example, on 32-bit Windows the value may be C:\Program Files. When using 64-bit Windows, the value may be C:\Program Files (x86).
[ROOTDRIVE]	The ROOTDRIVE property specifies the default drive for the destination directory of the installation. If the Directory column of the Directory table indicates the root destination directory by a property name that is undefined, the installer uses the value of the ROOTDRIVE property to resolve the path to the destination directory. If ROOTDRIVE is not set at a command line or authored into the Property table, the installer sets this property. During an administrative installation the installer sets ROOTDRIVE to the first connected network drive it finds that can be written to. If it is not an administrative installation, or if the installer can find no network drives, the installer sets ROOTDRIVE to the local drive that can be written to having the most free space.
[ProgramMenuFolder]	The installer sets the ProgramMenuFolder property to the full

	path to the Program Menu folder for the current user. If an "All Users" profile exists and the ALLUSERS property is set, then this property is set to the folder in the "All Users" profile. Common values for this property are C:\Winnt\Profiles\[LogonUser]\Start Menu\Programs\ (Windows NT/Windows 2000) and C:\Windows\Start Menu\Programs\ (Windows 95 and Windows 98).
[SendToFolder]	The installer sets the SendToFolder property to the full path to the SendTo folder for the current user. Common values for this property are C:\Winnt\Profiles\[LogonUser]\SendTo\ (Windows NT/Windows 2000) and C:\Windows\SendTo\ (Windows 95 and Windows 98).
[StartMenuFolder]	The installer sets the StartMenuFolder property to the full path to the Start Menu folder. By default, this property is set to the folder for the current user. If an "All Users" profile exists and the ALLUSERS property is set, then this property is set to the folder in the "All Users" profile.
[StartupFolder]	The installer sets the StartupFolder property to the full path to the Startup folder. By default, this property is set to the folder for the current user. If an "All Users" profile exists and the ALLUSERS property is set, than this property is set to the folder in the "All Users" profile.
[System64Folder]	The installer sets the System64Folder property to the full path to the predefined System64 folder. The existing System64Folder property is set to the corresponding 32-bit folder.
[SystemFolder]	The installer sets the SystemFolder property to the full path of the System folder. For example, on 32-bit Windows the value may be C:\Windows\System32. On 64-bit Windows, the value may be C:\Windows\SysWow64.

This folder is normally a subdirectory of the Windows folder. However, it resides on a server when configured for Shared Windows. |
| [TempFolder] | The installer sets the TempFolder property to the full path to the Temp folder. A common value for this property is C:\Temp. |
| [TemplateFolder] | The installer sets the TemplateFolder property to the full path to the Template folder for the current user. Common values for this property are C:\Winnt\Profiles\[LogonUser]\ShellNew\ (Windows NT/Windows 2000) and C:\Windows\ShellNew\ (Windows 95 and Windows 98).
On Windows 2000 only, if the ALLUSERS property is set, this property points to a file system directory that contains the templates that are available to all users. A common path is C:\WINNT\Profiles\All Users\Templates. |
| [WindowsFolder] | The installer sets the WindowsFolder property to the full path to the Windows folder. Common values for this property are C:\Winnt (Windows NT/Windows 2000) and C:\Windows (Windows 95 and Windows 98). |
| [WindowsVolume] | The installer sets the WindowsVolume property to the volume of the windows folder. The property always ends with a backslash. This can be used to set the default location to the volume the Windows folder is on because the ROOTDRIVE property does not necessarily equal this drive. |

Please refer to the Windows Installer documentation for further information.

10.Troubleshooting

I always test my install programs on a "virgin" Windows installation using the free Microsoft Virtual PC. That's fast, and if the installation fails, I can always roll back (stop the virtual PC without saving any changes) and try again. All without messing up my normal Windows installation. I have a number of Windows installations ready for this; totally clean ones, with .NET, with Visual Studio etc. So when I need to test something, I use a virtual PC similar to the expected target system.

1. Windows Services

The Windows Installer is *not* very helpful if you experience problems with Windows Service installation. Basically, it will complain about a failure with the installation, and suggest that you might not have permissions to install services. This applies even if the problem is missing dlls that prevents the service from starting. I will suggest that you try to install the service as a normal file (non-service) in a virtual PC, and try to start it from the command-line. If that works, try to install it as a service with manual startup, and with the *StartWhenInstalled* flag set to false. Then you can troubleshoot the problem and hopefully find the solution. When the service starts and behaves OK, you can change the options and verify that the installation works.

11. Change log

Version 3.13 – Tuesday February 17th 2009

- Fixed bug in System Folders (Target Directories). They should now work all right.

- Added *LicencePath* to the Project options.

- Added *LaunchAppText* to the Interface options*.*

Version 3.12 – Friday February 6th 2009

- Support for font-installation from WarSetup was broken with the December 2008 release of WiX 3 Beta. Fixed.

- Added support for most of the built-in folders supported by Windows Installer, including the infamous system32 ([SystemFolder]).

- Added drop-down list in properties for target-directories. This simplifies the use of these properties a lot.

- Added shortcutInStartupFolder property on files, to support auto-launch when the user signs in to Windows.

Version 3.11 – Friday January 30th 2009

- Added "Open Recent Projects" menu. (This is not supported directly by VS 2005 – believe it or not!).

- Code Cleanup [internally in War Setup]: Moved the code-generation and compiler-stuff out of MainFrame and into the project class where it belong.

- Moved the Build-command from a button on the first tab to the tool bar.

- Added multi treading. Compilation is now performed in a separate thread. That makes the program look nicer during compilation. This feature can be disabled from the Tools / Options menu.

- Compilation is now performed at a lower thread-priority. Windows have a problem when several programs compete for CPU-time at the same priority-level. As a result, most of the CPU is wasted switching between the competitors. By lowering the priority, both the compilation and other tasks should as a result run faster. And the machine will be more responsive for your other work. This feature can be disabled from the Tools / Options menu.

- Added support for the AdvertiseExecuteSequence table.

- Added feature for "Uninstall" from Program Files (patch from user).

- Added menu: Tools/Run Target.

- Added menu Tools/Open Target Directory.