The FieldWorks Installer System

# 0. Introduction

Users install FieldWorks by running a master installer program, setup.exe, which in turn launches the FieldWorks installer, SetupFW.msi. It is also possible for users to upgrade from one release to the next, by means of a patch installer. The advantage of a patch installer is that it contains a binary difference between old and new files, rather than the new files themselves, and is thus much smaller than the full installer.

This document attempts to explain how the installer and its patches are built on a FieldWorks development machine. Most of the processes are automated, and those that are not are fairly straightforward, but it is a significant advantage to understand what you are doing when you carry out these processes.

## 0.1 Important Concepts

In order to understand what is going on in the installer, it is important to understand a few basic concepts and terms as they relate to installers:

* **Resources.** These are the basic elements that can be installed. A file is a resource, and so is a registry setting. There are others too, such as shortcuts, and .ini file settings.
* **Components.** These are the essential building blocks of an installer. Resources that are to be installed are grouped together inside components. A component is an atomic unit in that it is installed in its entirety or else omitted in its entirety. One of the resources in a component is labeled as the Component Key. When the installer attempts to detect if the component is already installed or not, or in need of repair, only the Key resource is tested. If the Key resource is present and intact, then the whole component is regarded as installed, even if some other resource in the component is missing or damaged. For this reason, some installer authors like to put each resource in its own component. This is permitted, but makes installation slower. The FieldWorks installer puts every file in its own component, but collects registry settings into larger groups. Components have two identifiers: an **Id** which is unique within the installer, and a **GUID** which of course is unique throughout all time and space (probably). In this way, a component can be shared between installers, with reference-counting built in, so that uninstalling one application won’t unplug others that are still using the resources in that component.
* **Features.** These group together the components into units that the user can select for installation. For example, the FieldWorks installer has a feature for each localization language, so users can install any combination of languages by making selections in the installer’s feature selection tree dialog. Every component must belong to at least one feature. It is possible for features to share components.

# 1. Building an Installer

To build a FieldWorks installer on a development machine, you should first have a **release** build with **localization** completed. Then you can either run this batch file in the Installer folder:

WIX Installer Full Build.bat

Or you can call the Nant targets **release BuildFwInstaller**.

That’s all there is to it, unless you want to know what’s going on under the hood…

## 1.1. Under the Hood

The building of the FieldWorks installer comprises of various pre-processing scripts/utilities, the compilation of the WIX source files (some of them the results of pre-processing), and finally the linking of the compiled WIX sources.

### 1.1.1. Pre-processing scripts and utilities

First of all, any existing FileLibraryAddenda.xml and RegLibraryAddenda.xml files are deleted. These files are discussed later, but each installer build needs the opportunity to create its own versions of these files. Next, there are four pre-processing steps. One is written in JavaScript, and the rest are compiled C# projects.

#### 1.1.1.1 ProcessMergeModules.js

This script processes MergeModules.wxs, converting relative source paths of consumed merge modules into absolute paths.

#### 1.1.1.2 GenerateFilesSource.exe

This program automatically determines which files go in the installer, and writes their details into the WIX source file AutoFiles.wxs. (Its C# project files are in the GenerateSourceFiles subfolder of the Installer folder). It works by examining various folders (basically Output\Release and DistFiles) for candidate files, using various heuristics to separate needed files from junk (such as unit test results). Much of this behavior is defined in the InstallerConfig.xml file. The program determines which files are needed for FLEx only, which are needed for TE only, and which are needed for both, gathering components into corresponding features. To do this, it calls the Nant targets **remakele** (to build FLEx only) and **remakete** (to build TE only). **Therefore, it is imperative that developers maintain the integrity of these Nant targets, to prevent TE-only files from appearing in a FLEx-only installation.**  (A FLEx-only installer is needed in some sensitive locations.) The Nant targets **release** and **build** are also called with each of the above, to avoid building or running tests, and to generate a release build as opposed to a debug one.

GenerateFilesSource.exe means developers no longer to have to edit installer files when they create new distributable files.

Files have to be assigned to components, and components need GUIDs, but we don’t want the same file having a different component GUID every time the installer is released. Therefore GenerateFilesSource.exe maintains a component library so that only new files get put into new components with new GUIDs. The library is in a file called FileLibrary.xml. (There is a separate library for registry data, but we’ll come to that later.) This XML file doesn’t exist, initially, but is created and checked into Perforce *when the first official installer version is released*. There is a special process for this; see section 2. In order to keep a record of new files since the last official release (or a record of all files if there hasn’t yet been a release), GenerateFilesSource.exe maintains a file FileLibraryAddenda.xml. This file is **not** checked into Perforce. The Addenda file should never be deleted or edited, unless you know exactly what you are doing! (Feel free to take a look at it and admire its craftsmanship, though.) Should this file ever be lost, the installer can simply be rebuilt, although the installer associated with the lost xml file should be scrapped.

You will probably see now that it is best to build an installer only on the overnight build machine (ls‑fwbuilder) because the FileLibraryAddenda.xml file is only maintained locally, and we don’t want GUIDs getting out of sync across development machines.

It should be noted that there is a WIX source file Files.wxs which is used to set up internal names for folders that are needed elsewhere in the installer, and to include files which aren’t dealt with automatically by the program.

#### 1.1.1.3 ProcessFiles.exe

This program processes AutoFiles.wxs and Files.wxs which were mentioned in the previous section. It converts the resource file paths from relative to absolute, assigns the Key for each Component, and adds .NET assembly information (where relevant).

[It used to do one more job, which it still could if the line calling AddRegInfo() were uncommented: adding in registry data for both COM and COM-interoperability for every DLL that has it. The main reason for having it was that it is bad practice (and very risky) to have the installer call RegSvr32 or RegAsm on the user’s machine during installation. Instead, the installer is supposed to deliver the registry information ready-prepared. ProcessFiles.exe gathered in all that registry information, and grouped it into components such that it could be updated by a patch installer when new versions of the DLLs were built. (These tended to get new class GUIDs and/or file version numbers on different build dates.) To make sure unchanged registry info was not assigned a new component GUID every time the installer was built, a component library was maintained, in a manner similar to the file library system mentioned in the previous section. The main library file was RegLibrary.xml, and the local changes since the last release were kept in RegLibraryAddenda.xml. This functionality is no longer used because FieldWorks has been changed to no longer need COM registry data, in order to make it so users can install new versions of FieldWorks side by side.]

#### 1.1.1.4 TestInstallerIntegrity.exe

This program tests the WIX sources for situations that make it difficult or impossible to produce a patch installer. For example, Windows Installer patches were not designed to be able to remove obsolete files or registry settings. For a patch to be able to do this, special measures have to be taken. As another example, a component that has moved from one feature to another across release versions makes it impossible to create a patch between those versions. TestInstallerIntegrity.exe detects such situations by analyzing the current installer source files against the component libraries. Developers should be aware that **components must not be removed or relocated to other features when upgrading from one version to the next.** If there are any troublesome situations, a report will be written containing errors, warnings, and details of missing components, along with snippets of WIX source code which *might* correct the problem, where applicable. Such snippets must be carefully analyzed, corrected (where appropriate) and copied into PatchCorrections.wxs. See section 2.1 for details. If this testing is performed on the overnight build machine, the report is emailed to various key developers. On other machines, the report is saved as file TestInstallerIntegrity.log and will be opened immediately. The file InstallerConfig.xml contains settings to determine which machines send out emails and who gets them.

### 1.1.2. Compilation of the WIX source files

This step and beyond can be accomplished by running the batch file WIX Installer Compile and Link.bat. It simply calls the WIX Candle utility to compile each WIX source file. Warnings about short file names are suppressed. (Windows Installer still needs 8.3 format file names!)

### 1.1.3. Linking of the compiled WIX sources

This step can be accomplished by running the batch file WIX Installer Link.bat. It simply calls the WIX Light utility to link all the compiled WIX source files together, along with the WIX custom actions library.

# 2. Releasing an Installer and Building a Patch Installer

Whereas the initial release of a major version of FieldWorks has a full installer, subsequent bug-fix and upgrade releases will be available both as full installers and patch installers. The production of patch installers has largely been automated, but some manual steps are still required, and in some cases, some expertise as well. Once it has been decided that the current FieldWorks build is to be released, the principle steps are:

1. Check out from Perforce the files FW.wxs, FileLibrary.xml, RegLibrary.xml and PatchCorrections.wxs. (Some of these may not exist; if so, they can be ignored.)
2. Examine the TestInstallerIntegrity.log file and incorporate any necessary changes into the PatchCorrections.wxs file. If there are changes, rebuild the installer.
3. Run the ArchiveAndBuildPatch.exe utility.
4. Post the built installer and patch package(s) on the web server.
5. Increment the installer version number in the WIX source file FW.wxs ready for the next release.
6. Submit to Perforce the files FW.wxs, FileLibrary.xml, RegLibrary.xml and PatchCorrections.wxs.

The above list can be used as a checklist of items to carry out. More details on each step are given below.

**Important note:** If step 3 is not carried out before a new installer is built, important data needed for patching will be overwritten in the Installer folder. The data can be put back by collecting it from one of two places:

1. on the build machine: D:\ FWBuilds\FW\_*<branch-and-date-details>*\Installer\\*.\*
2. on a server: \\BEREA2\FWBuilds\FW\_*<branch- and-date-details>*\FieldWorks\Release\\*.\*

The essential files are ProcessedAutoFiles.wxs, FileLibraryAddenda.xml, and RegLibraryaddenda.xml. (Some of these may not exist.)

## 2.1 Check Out Perforce Files

The files FW.wxs, FileLibrary.xml, RegLibrary.xml and PatchCorrections.wxs are likely to get edited in the steps below, so make sure they are opened for edit in Perforce, if they exist.

## 2.2 Adding Patch Corrections

After TestInstallerIntegrity.exe is run as part of an installer build (see section 1.1.1.4) there may be some snippets of WIX source in file TestInstallerIntegrity.log which need to be added into the installer. These WIX snippets generally reinstate missing/obsolete components with new resources that actively remove the resources that were in those components. (If missing components are not reinstated, you will probably create a patch which fails spectacularly while giving the impression it is succeeding.) Although the log file attempts to give you the necessary WIX code for each missing component, it is not guaranteed error-free. An installer expert should examine the log file, and decide what to add to the WIX source PatchCorrections.wxs. One problem may be duplication of internal Id’s. (Of course, the GUIDs are duplicates from the previous release—we’re trying to maintain GUIDs in the new version.) Another problem may be where an entire folder of files has become obsolete. The WIX snippet created by TestInstallerIntegrity.exe will likely refer to the obsolete folder, and so the folder will have to be reinstated in PatchCorrections.wxs in order for the snippet to work.

If you edit the PatchCorrections.wxs file, you will have to scrap the existing installer and rebuild it. You can do this with WIX Installer Compile and Link.bat, because none of the pre-processing needs to be repeated.

## 2.3 Run the ArchiveAndBuildPatch.exe utility

This tool automates the bulk of the patch production process, but it does other important tasks too:

1. Consolidation of component libraries
2. Archival of current installer
3. Administrative install of current installer
4. Building of patches
5. Wrapping of patches into self-extracting archives

Details are given below. The output for steps 2, 3, 4 and 5 can be found under a subfolder of Installer\Archives and Patches. The actual subfolder name is the version number of the current installer.

The C# source and project files for this tool are in Perforce under Installer\ArchiveAndBuildPatch.

### 2.3.1 Consolidation of Component Libraries

After an installer is released, and before any new development work is carried out, it is important to consolidate the libraries of component GUIDs. This makes sure that any new files added to the installer for the next release are properly sequenced in the installer (sequenced at the end, to avoid re-sequencing existing files which would ruin a patch). The libraries are stored in the files FileLibrary.xml and RegLibrary.xml. Once these files exist they are stored in Perforce, so it is not appropriate for them to be edited by the overnight build whenever a new file or registry setting is added to FieldWorks. Instead, temporary files are created to hold just the updates. These files are FileLibraryAddenda.xml and RegLibraryAddenda.xml. These addenda files are not in Perforce, so once an installer gets to be released, we need to add the addenda data to the main library files. The ArchiveAndBuildPatch.exe utility processes the File and Registry Library Addenda files, merging their data into the main library files, and organizing them so the installer will know which data are new since the last release.

### 2.3.2 Archival of Current Installer

As a precaution, the current installer is copied to a subfolder of Installer\Archives and Patches. The actual subfolder name is the version number of the installer. This enables patches to be rebuilt without having to rebuild FieldWorks, should the need arise. Also archived there is the ProcessedAutoFiles.wxs file, which can assist in tracing obsolete folders should they need to be reinstated in PatchCorrections.wxs. (See section 2.1)

### 2.3.3 Administrative Install of Current Installer

An “administrative installation” simply extracts all the files embedded in an installer .msi and .cab files so that they lie in a folder structure. This is important because the patch creation tool (see section 2.3.4.4) cannot access files embedded in .msi and .cab files. The administrative install also creates a new copy of the original .msi file but without the embedded files.

An administrative install can be performed via a DOS command:

Msiexec /a *<installer.msi>* TARGETDIR=”*<target-folder>*”

### 2.3.4 Building of Patches

Currently, a maximum of 2 patches is built each time ArchiveAndBuildPatch.exe is run: one to patch from the earliest archived installer up to the current one, and one to patch from the last archived installer to the current one (unless that is the same thing as the first patch). If the current installer is the first one to be archived, then no patch can be built, obviously.

There are several stages to creating a patch:

1. Create WIX source for the patch
2. Compile WIX source
3. Link WIX source
4. Run the Microsoft patch builder tool.

#### 2.3.4.1 Create WIX source for the Patch

The WIX source file for the patch is created on the fly. Its file name is of the form patch???to???.wxs, where the first ??? represents the initial installation version number, and the second ??? represents the final installation version number. Each patch source file is unique, to reflect the old and new versions of the installer that the patch has to upgrade from and to, respectively. In particular, the following WIX node attributes are usually unique (but don’t treat this list as comprehensive):

* PatchCreation[Id] – this needs a new GUID for each patch.
* PatchInformation[Description], PatchMetadata[Description] , PatchMetadata[DisplayName] , UpgradeImage[Id], TargetImage[Id] – these all need to reflect the previous and current versions of the installer, as appropriate.
* Family [DiskId] – this needs to be one higher than the number of .cab files produced by the main installer build.
* Family [MediaSrcProp] – this needs to be different for each patch released.
* Family [SequenceStart] – this needs to be higher than the highest file Sequence number in the main .msi installer, which can be inspected by loading the .msi into Microsoft’s Orca tool and examining the File table, sorted by sequence.
* UpgradeImage[src], TargetImage[src] – these need to reflect the locations of the old and new administrative installations performed in the previous section.
* TargetImage[Order] – probably needs to be one higher than in previous patch.

#### 2.3.4.2 Compile WIX Source

The WIX source for the patch is compiled with the WIX Candle utility.

#### 2.3.4.3 Link WIX Source

The WIX Light utility is used to produce a .pcp file.

#### 2.3.4.4 Run the Microsoft Patch Builder Tool

The .pcp file and administrative installs are combined to make a patch installer using the msimsp.exe tool.

### 2.3.5 Wrapping of Patches

In order to launch a patch installer correctly, the command line must contain several parameters, as in this example:

msiexec /update Patch700to701.msp REINSTALL=ALL REINSTALLMODE=omus

This is generally too complicated for an end user to be expected to carry out, especially as merely double-clicking the patch file appears to work (although it doesn’t, and the differences are subtle). Consequently, the patch file is wrapped in a self-extracting .exe file which is configured to launch the patch with the correct parameters, and delete the patch after running it.