Beginner’s Guide to

Windows Installer XML (WiX)

[WiX v3.5]

Alek Davis

http://alekdavis.blogspot.com

March 2011

Disclaimer

- **From *Want to sell books to Americans? Eliminate all traces of self-doubt* by Ezra Klein**
  - Gillian Tett [author of *Fool's Gold*] describes the difference between packaging a book for the American and British markets:
    - *"Initially I planned to start the book by **admitting that I was not a true expert** on high finance: instead I crashed into this world in 2005, after a background spent in journalism-cum-social anthropology – making me a well-intentioned amateur, but without complete knowledge. My friends in the British publishing world loved that honesty; **in the UK, self-deprecation sells**, particularly for "well-meaning amateurs" such as the writer Bill Bryson. But my American friends hated it. **In New York**, I was sternly told, absolutely **nobody wants to listen to self-doubt**. If you are going to write a book – let alone stand on a political platform or run a company – you must act as if you are an expert, filled with complete conviction. For the US version, the preface was removed entirely."*
- **I am not a "true expert" on Windows Installer**

Introduction

- **Experience**
  - Programming
  - Installer-related
- **Tools and articles**
  - *Streamline Your Database Setup Process with a Custom Installer* (MSDN Magazine, September 2004)
    - Follow-up: *Database installer revised* (Blog)
    - Commercial database deployment software
  - *Dude, where is your installer?* (Blog)
  - *Microsoft ate my uninstaller* (Blog)
  - More posts on my blog

What to expect?

- **Agenda**
  - Basic Windows Installer (MSI) concepts
  - Overview of Windows Installer XML (WiX)
  - Using WiX to build simple installers
  - Common advanced WiX techniques
  - Tools and resources

Installer tools and technologies

- **Relevance**
  - Do you build installers?
  - What do you use?
- **Tools and technologies**
  - Windows Installer (MSI)
    - Visual Studio Installer (discontinued)
    - InstallShield
    - Wise Installation Studio
    - InstallAware for Windows Installer
    - Advanced Installer
    - And more
  - Other
    - InstallShield engine
    - NullSoft Scriptable Install System
    - Inno Setup
    - ClickOnce
    - And more

Installation challenges

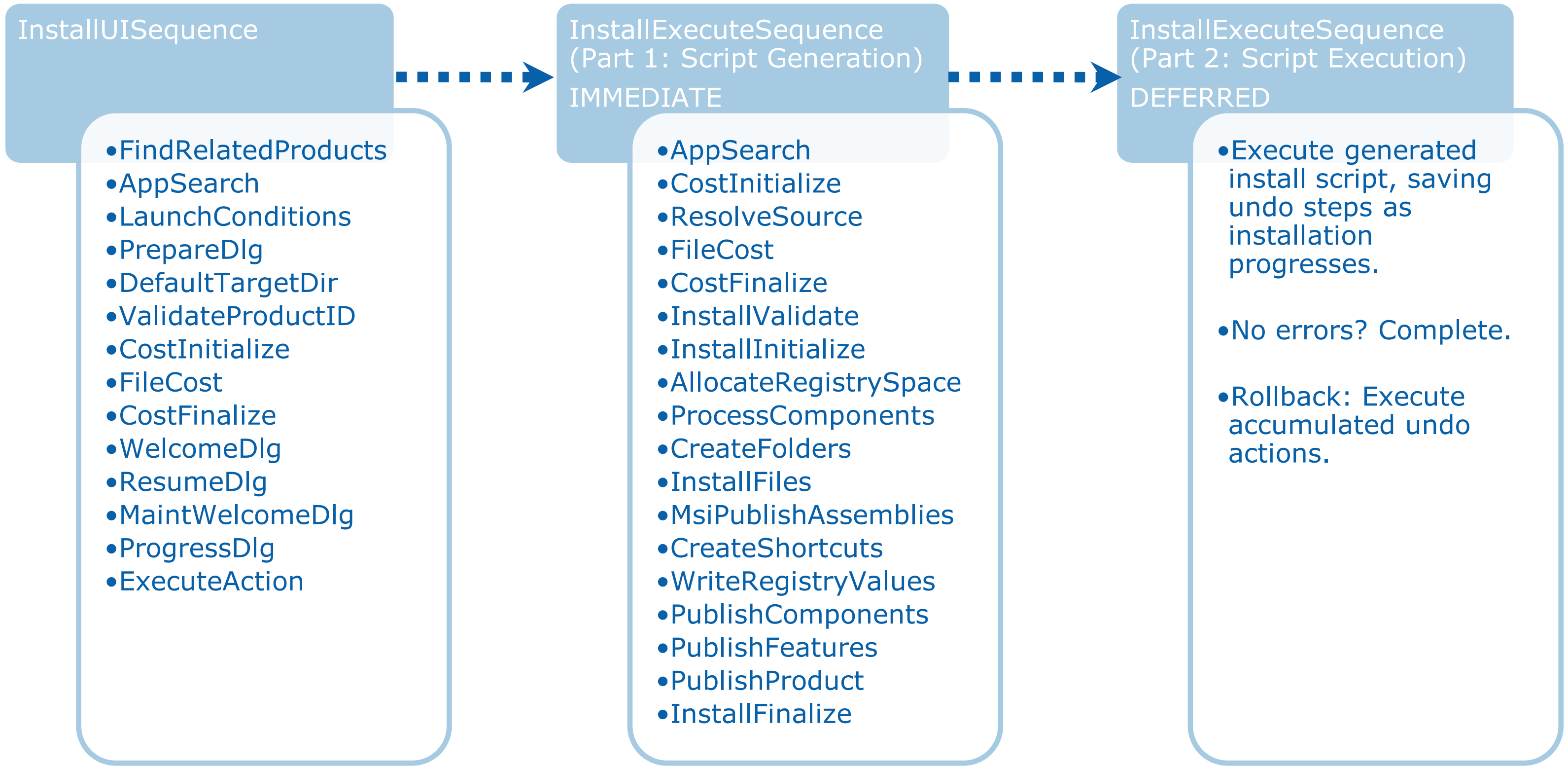
- **Consider**
  - Error in the middle of setup
  - Per-user or per-machine installation
  - Roaming profiles and folder redirection
  - x86, x64, any CPU
  - 32-bit, 64-bit OS
  - Patches and upgrades
  - Conditional installations
  - …

Windows Installer (MSI)

- **What is Windows Installer?**
  - Msiexec.exe (see command-line options)
  - Windows Installer API
  - Windows Installer SDK
- **History**
  - Started around 1999 (Office 2000)
  - Current version: 5 (Windows 7, Server 2008 R2)
- **Pros**
  - Transactional, merge modules, discovery, automation, APIs, localization, restart management, package validation, declarative, …
- **Cons**
  - Complexity, upgrades, architecture (one key [file] per component, …), shortcuts, command line, touches registry and file system, tools, …
- **FAQs**
  - Windows Installer FAQ – Part 1 (see parts 2, 3, 4, etc)

Windows Installer setup process

- **Installation sequence** *(courtesy to Jeffrey Sharp)*
  - Installation (uninstallation/upgrade/etc) events occur in sequence



InstallUISequence

- FindRelatedProducts
- AppSearch
- LaunchConditions
- PrepareDlg
- DefaultTargetDir
- ValidateProductID
- CostInitialize
- FileCost
- CostFinalize
- WelcomeDlg
- ResumeDlg
- MaintWelcomeDlg
- ProgressDlg
- ExecuteAction

InstallExecuteSequence (Part 1: Script Generation)
IMMEDIATE

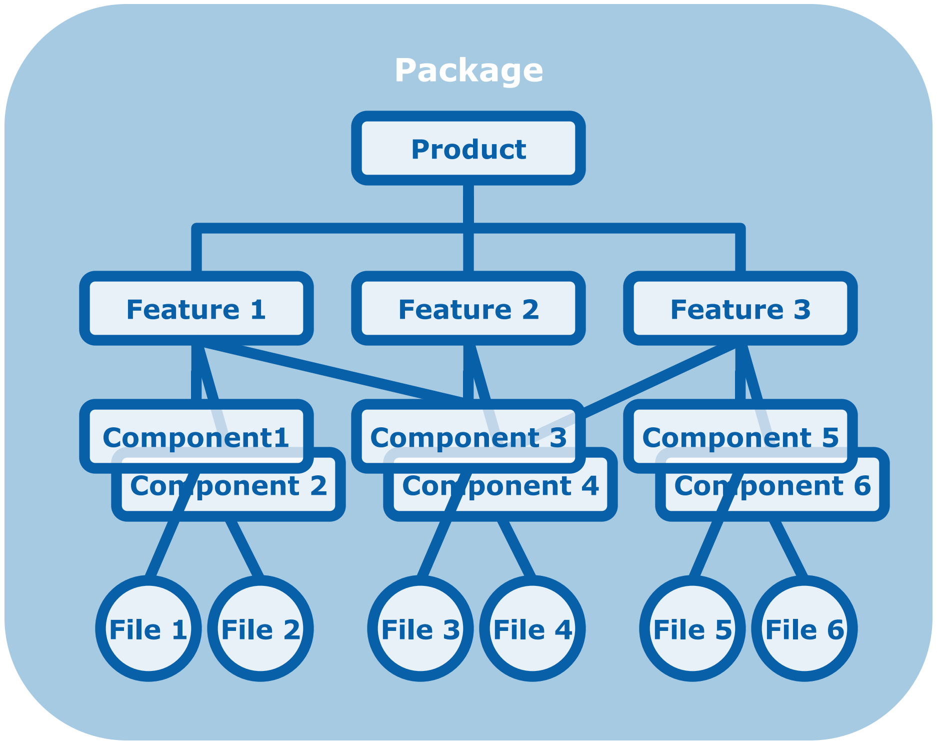
- AppSearch
- CostInitialize
- ResolveSource
- FileCost
- CostFinalize
- InstallValidate
- InstallInitialize
- AllocateRegistrySpace
- ProcessComponents
- CreateFolders
- InstallFiles
- MsiPublishAssemblies
- CreateShortcuts
- WriteRegistryValues
- PublishComponents
- PublishFeatures
- PublishProduct
- InstallFinalize

InstallExecuteSequence (Part 2: Script Execution)
DEFERRED

- Execute generated install script, saving undo steps as installation progresses.
- No errors? Complete.
- Rollback: Execute accumulated undo actions.

Windows Installer basics

- **Major constructs**
  - **Product**
    - Deployed as a **package**
    - Made of feature(s)
  - **Feature**
    - Made of component(s)
  - **Component**
    - Made of installable(s)
  - Installables
    - Files, directories, shortcuts
    - Registry keys, registry values
    - ODBC data sources
    - …

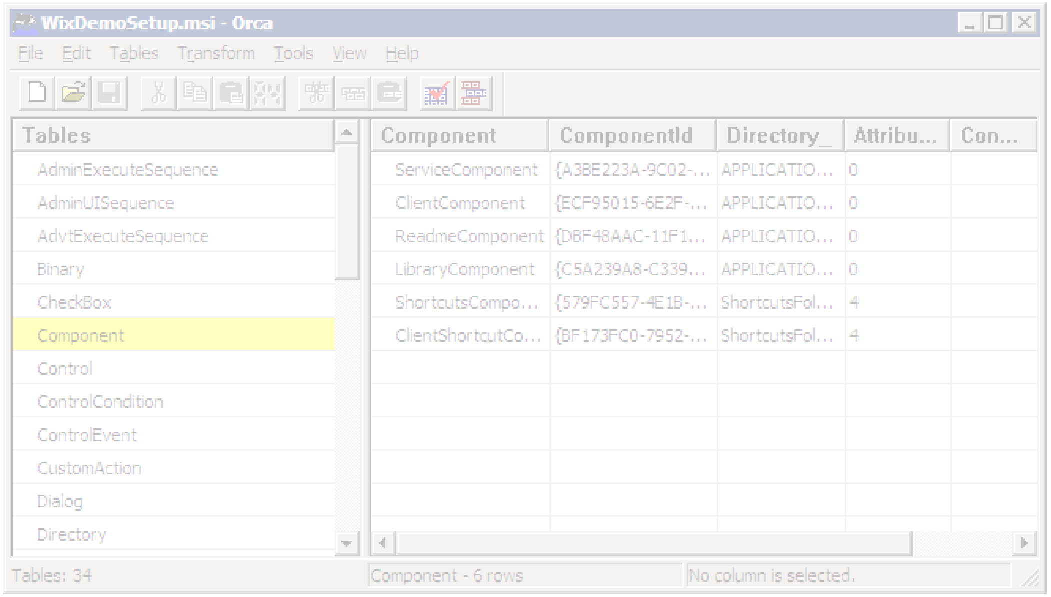


Windows Installer package (MSI file)

- **What is it?**
  - COM structured storage file
    - Summary info stream (product name, package GUID, MSI version, …)
    - Database (tables: product description, install sequence, dialogs, …)
    - Data streams (files: product files, support files, icons, …)
    - Declarative
- **Types**
  - MSI
    - Standard setup package
  - MSM
    - Merge module
  - MSP
    - Update patch
  - MST
    - Transform package

MSI tables

- **Required**
  - Component
  - Condition
  - Control
  - CustomAction
  - Dialog
  - Directory
  - Feature
  - File
  - …
- **Optional**
  - Not defined in the specification
  - Apps can use to store app-specific info



Tables
AdminExecuteSequence
AdminUISequence
AdvtExecuteSequence
Binary
CheckBox
Component
Control
ControlCondition
ControlEvent
CustomAction
Dialog
Directory

Component	ComponentId	Directory_	Attribu…	Con…
ServiceComponent	{A3BE223A-9C02-…	APPLICATIO…	0	
ClientComponent	{ECF95015-6E2F-…	APPLICATIO…	0	
ReadmeComponent	{DBF48AAC-11F1…	APPLICATIO…	0	
LibraryComponent	{C5A239A8-C339…	APPLICATIO…	0	
ShortcutsCompo…	{579FC557-4E1B-…	ShortcutsFol…	4	
ClientShortcutCo…	{BF173FC0-7952-…	ShortcutsFol…	4	

Tables: 34 | Component - 6 rows | No column is selected.

Windows Installer SDK tools

- **Orca.exe**
  - MSI database viewer and editor
- **Scripts**
  - WiRunSQL.vbs
    - Updates tables in MSI database (can be used in a post-build step)
  - More
    - See C:\Program Files\Microsoft SDKs\Windows\v7.0\Samples\sysmgmt\msi\scripts

WiX project

- **Open-source**
  - Started in 1999
  - "Sponsored" by Microsoft
  - Lead by Rob Mensching
  - Current version: 3.5
- **Home pages**
  - Codeplex (distributions)
  - SourceForge (documentation, bug tracking)

WiX defined

- **Windows Installer XML (WiX)**
  - XML syntax
    - Defines Windows Installer (MSI) package
    - Declarative
  - Tools

WiX tools

- **WiX Toolset**
  - Command-line tools that convert WiX source files to MSI packages
    - Candle.exe
    - Light.exe
    - …
  - Utilities and documentation
- **Votive**
  - Integrates WiX Toolset with Visual Studio
  - WiX project templates
  - IntelliSense

WiX/Votive vs. Visual Studio Installer

- **Cons** **(WiX/Votive)**
  - Steeper learning curve
  - Requires understanding of Windows Installer
  - No drag-and-drop
  - No automatic dependency inclusion
  - Does not build bootstrappers (setup.exe)
  - Web site
- **Pros** **(WiX/Votive)**
  - More flexible
  - More powerful
  - Produces cleaner MSI package
  - XML-based
  - Clean syntax

WiX project templates

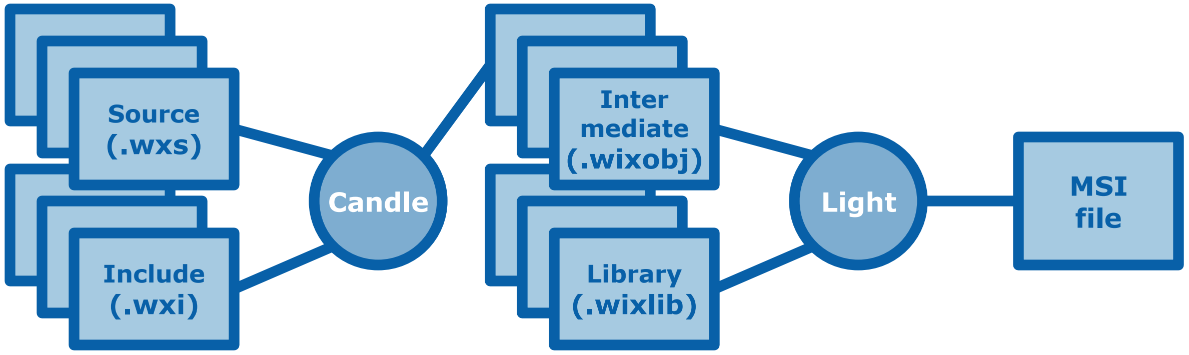
- **Visual Studio projects**
  - Windows Installer XML
    - **Setup** Project
      - Creates MSI package
    - **Merge Module** Project
      - Creates MSM package
    - **Library** Project
      - Creates WIXLIB package for use by other WiX projects
    - (C# | VB | C++) **Custom Action** Project
      - Creates a module implementing a custom action

WiX project

- **Files**

– Solution file:	.sln
– Project file:	.wixproj
– Source file(s):	.wxs
– Include file(s):	.wxi (optional)
– Localization file(s):	.wxl (optional)

- **Build process** (simplified)



WiX source file

- **Basic structure**

```
<?xml version="1.0" encoding="UTF-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
<Product …>
  <Package … />
  <Media … />

  <Directory Id="TARGETDIR" Name="SourceDir">
    <Directory Id="ProgramFilesFolder">
      <Directory Id="APPLICATIONFOLDER" Name="WiX Demo"/>
    </Directory>
  </Directory>

  <DirectoryRef Id="APPLICATIONFOLDER">
    <Component Id="C_ID" Guid="ABCDEF01-2345-6789-ABCDEF01234576" …>
      <File …/>
    </Component>
 </DirectoryRef>

 <Feature Id="F_ID" …>
   <ComponentRef Id="C_ID"/>
 </Feature>
</Product>
```

Product.wxs

Product, package, media

- **Product element**
  - *Id* (GUID, identifies product/version/edition/etc)
  - *UpgradeCode* (GUID, identifies product upgrade path)
  - *Name*
  - *Manufacturer*
  - *Version*
  - *Language* (LCID, e.g. "1033")
- **Package element**
  - *Platform* (x86, x64, ia64)
  - *InstallScope* (perUser, perMachine)
  - *InstallerVersion* (100, 200, 300, 301, 405, 500)
  - *Description*
- **Media element**
  - Important only when CAB file is not embedded in MSI (`EmbedCab="no"`)
  - Use default values

Package types

- **Platform**
  - x86, x64, ia64
    - See *Advanced Installer Package Types*
    - See *Define platform variables for x86 and x64 builds*
- **Scope**
  - User
    - AVOID IF POSSIBLE!
  - Machine

Directories

- **Directory element**
  - Nested directory structure
    - Under `<Directory Id="TARGETDIR" Name="SourceDir">`
  - Properties
    - *Id* (primary key)
      - *System*: ProgramFilesFolder, ProgramFiles64Folder, SystemFolder, System64Folder, StartMenuFolder, StartupFolder, …
        - See *A gentle introduction to Windows Installer XML Toolset*
      - *User-defined*: make sure it does not conflict with system names
    - *Name/LongName* (of the folder: 8.3/long name)
      - Not required for system directories
      - *LongName* requires *Name* to be defined
  - Caveats
    - 32-bit vs 64-bit (see *Advanced Installer Package Types*)
    - User profile-specific folders ("All Users", etc)
- **DirectoryRef element**
  - References already defined directory

Features

- **Feature element**
  - Can contain other features
  - Properties
    - *Id* (text)
    - *Level* (0 – disables feature; 1, 2, … - enables feature; can be modified by conditions)
    - *Title* (short description/name)
    - *Description* (longer description)
    - *ConfigurableDirectory* (non-default directory; must be ID of a PUBLIC property)
    - *Absent* (allow to exclude from installation: `disallow`, `allow`)
    - *AllowAdvertise* (`yes`, `no`, `system` [=yes, except…])
- **FeatureRef element**
  - References already defined feature

Install levels

- **Purpose**
  - Allow selective feature installation
- **Values**
  - `0` : feature will not be installed
  - `1` : default installation level
  - `>1` : use for selective installation

Feature advertisement

- **Purpose**
  - Installs features on demand (when user first tries to run them)
  - See *Windows Installer – Overview of Windows Installer - Advertisement*

Components

- **Component element**
  - One (key) installable (file, registry value, …) per component
  - Properties
    - *Id* (text)
    - *Guid*
- **ComponentRef element**
  - References already defined component
- **ComponentGroup element**
  - Groups related components

Files

- **File element**
  - One key file per component
  - Properties
    - *Id* (text)
    - *Name* (name of the file once it is installed)
      - If *ID* specifies file name, *Name* can be omitted
    - *Source* (location on the build system)
    - *KeyPath* (should be *"yes"*; otherwise it will not be repaired)
    - *Assembly*
      - *".net"* – will copy assembly file (.dll) to GAC
        - It will ignore the Directory element settings
      - *"win32"* – will not copy file to GAC
    - *AssemblyApplication*
      - Defines the main executable for a .NET DLL; if it is specified, the .NET DLL will be copied next to this executable

Files (continued)

- **Example**

```
<Component …>
 <File Id="MyApp.exe"        KeyPath="yes" Vital="yes" Assembly=".net" Checksum="yes" />
</Component>
<Component …>
 <File Id="MyPrivateLib.dll" KeyPath="yes" Vital="yes" Assembly=".net" AssemblyApplication="MyApp.exe" />
</Component>
<Component …>
 <File Id="MySharedLib.dll"  KeyPath="yes" Vital="yes" Assembly=".net" />
</Component>
<Component …>
 <File Id="readme.txt"       KeyPath="yes" Vital="yes" />
</Component>
```

Shortcuts

- **Shortcut element**
  - Under the `Start` menu (or `Desktop`) folder
  - Types
    - Advertised (references a component; launches setup; self-repairs)
    - Non-advertised (default; references a file)
  - Must include
    - `RemoveFolder` element
    - `RegistryValue` element (defining `KeyPath` for component)
  - Properties
    - *Id* (text)
    - *Name* (as it appears)
    - *Description* (tooltip text; optional)
    - *Target* (references a file on target system)
    - *Arguments* (optional command-line arguments)
    - *Icon* (references `Icon` element that must be defined; optional)
      - Icon element must be defined and point to a file on the build system

Icons

- **Icon element**
  - Properties
    - *Id* (text; must contain an extension, e.g. "`program`**`.exe`**")
    - *SourceFile* (on the build system; can be .exe, .ico file)
  - Caveats
    - If icon element ID does not contain extension, it may not work
    - Icons are not always necessary

Shortcuts (continued)

- **Example**

```
<Directory Id="TARGETDIR" Name="SourceDir">
  <Directory Id="ProgramFilesFolder">
    <Directory Id="APPLICATIONFOLDER" Name="WiX Demo"/>
  </Directory>
  <Directory Id="ProgramMenuFolder">
    <Directory Id="ShortcutsFolder" Name="WiX Demo"/>
  </Directory>
</Directory>
…
<DirectoryRef Id="ShortcutsFolder">
  <Component Id="ProgramShortcut" Guid="ABCDEF01-2345-6789-ABCDEF01234576" …>
    <Shortcut Id="ClientShortcut" Name="WiX Demo Client GUI" Description="Launch WiX Demo Client GUI"
        Target="[APPLICATIONFOLDER]Client.exe" WorkingDirectory="APPLICATIONFOLDER" Icon="Client.exe">
      <Icon Id="Client.exe" SourceFile="..\..\Release\bin\Client.exe"/>
    </Shortcut>

    <RemoveFolder Id="ClientShortcutRemoveFolder" On="uninstall"/>

    <RegistryValue Root="HKCU" Key="Software\WiX Demo\Client" Name="installed" Type="integer"
        Value="1" KeyPath="yes"/>
  </Component>
</DirectoryRef>
```

Registry

- **RegistryKey element**
  - Properties
    - *Id* (text)
    - *Action* (`none, create, createAndRemoveOnUninstall`)
    - *Root* (`HKLM, HKCU, …`)
    - *Key* (path, e.g.: `SOFTWARE\MyCompany\MyProduct`)
- **RegistryValue element**
  - Defined under `RegistryKey` element (or must specify the key)
  - One key path per component
  - Properties
    - *Id* (text)
    - *Action* (append, prepend, write)
    - *Name* (name of the value)
    - *Value* (path, e.g.: `SOFTWARE\MyCompany\MyProduct`)
    - *Type* (data type: `string, multiString, integer, binary, …`)
    - *KeyPath* (yes/no; one key path per component)

Registry (continued)

- **RemoveRegistryKey element**
  - Removes key and all of its subkeys from the Registry
  - Properties
    - *Id* (text)
    - *Action* (removeOnInstall, removeOnUninstall)
    - *Root* (HKLM, HKCU, …)
    - *Key* (path, e.g.: SOFTWARE\MyCompany\MyProduct)
- **RemoveRegistryValue element**
  - Removes value from registry key
  - Properties
    - *Id* (text)
    - *Root* (HKLM, HKCU, …)
    - *Key* (path, e.g.: SOFTWARE\MyCompany\MyProduct)
    - *Name* (name of the value)

Registry (continued)

- **Example**

```
<Component …>
  <RegistryKey
      Root="HKLM"
      Key="Software\MyCompany">

    <RegistryValue
        Action="write"
        Name="Some Name"
        Type="string"
        Value="Some Value"
        KeyPath="yes"
    />
 </RegistryKey>
</Component>
```

Windows services

- **ServiceInstall element**
  - Creates a Windows service for the specified executable
  - Properties
    - *Id* (text)
    - *Name* (service name)
    - *DisplayName* (service name displayed in the Services Control Panel)
    - *Description* (service description)
- **ServiceControl element**
  - Controls when a service must get started, stopped or uninstalled
  - Properties
    - *Id* (text)
    - *Name* (name of the service; same as in ServiceInstall)
    - *Start, Stop, Remove* (control when service must be started, stopped or uninstalled: install, uninstall, both)

Windows services (continued)

- **Example**

```
<Component …>
  <File …/>

  <ServiceInstall
      Id="SI_MyService"
      Name="mysvc"
      DisplayName="My Service"
      Description="Demo service that does not do anything"
      Start="auto"
      ErroControl="normal"
      Type="ownProcess"
  />

  <ServiceControl
      Id="SC_MyService"
      Name="mysvc"
      Start="install"
      Stop="both"
      Remove="uninstall"
      Wait="yes"
  />
</Component>
```

What else can you install?

- **Installables**
  - Folders, files, shortcuts
  - Windows services
  - Registry keys, registry values
  - COM registration info
  - COM+ applications
  - Drivers
  - File extensions
  - ODBC data sources
  - Config files
  - IIS web sites
  - IIS virtual directories
  - …

GUIDs

- **Importance**
  - Identify product, upgrade path, features, components
  - NEVER CHANGE UPGRADE CODE (GUID)… unless…
- **Explicit** (hard-coded)
  - Generate via Visual Studio's Tools – Create GUID menu
    - Use the registry format (can be used with or without braces)
- **Implicit** (auto-generated)
  - Use the asterisk (`Guid="*"`)
    - For *product*: generated anew on each build.
    - For *component*: hashes the full key path (stable as long as inputs do not change).
      - *File*: path + file name (can use well-known and computed names).
      - *Registry*: key [+ value name].
      - Hashing algorithm can change if you enable/disable FIPS mode.
  - Use with care during development.

Fragments

- **Purpose**
  - Let you to split elements of the WiX document between multiple wxs files
- **Example**

```
<?xml version="1.0" encoding="UTF-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
<Fragment…>
  <Component Id="C_ID1" Guid="ABCDEF01-2345-6789-ABCDEF01234576" …><File …/></Component>
  <Component Id="C_ID2" Guid="01ABCDEF-2345-6789-ABCDEF01234576" …><File …/></Component>
</Fragment>
```

ClientFiles.wxs

```
<?xml version="1.0" encoding="UTF-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
<Product …>
  …
  <DirectoryRef Id="APPLICATIONFOLDER">
    <ComponentRef Id="C_ID1"/>
    <ComponentRef Id="C_ID2"/>
  </DirectoryRef>

  <Feature Id="F_ID" …>
    <ComponentRef Id="C_ID1"/>
    <ComponentRef Id="C_ID2"/>
  </Feature>
</Product>
```

Product.wxs

Properties

- **Definition**
  - Global variables that Windows Installer uses during installation
  - IDs are case-sensitive (MyProperty ≠ MYPROPERTY)
- **Types**
  - Public
    - Can be set and changed outside of MSI database (e.g. via command line or UI)
    - ID in UPPERCASE: `MY.CUSTOM.PROPERTY`
  - Private
    - Set and used only internally (in the MSI database)
    - ID in MixedCase: `My.Custom.Property`
- **Common Windows Installer properties**
  - TARGETDIR, ALLUSERS, Installed, SourceDir, …
  - Reference
    - See *Windows Installer Guide – Properties - Property Reference*

Properties (continued)

- **Property element**
  - `<Property Id="Greeting" Value="Hello!"/>`
- **Using properties**
  - [PropertyX] : Returns value of PropertyX
  - [[PropertyX]] : Returns value of the property named by value of PropertyX
    - `<Property Id="PropertyX" Value="PropertyY"/>`
    - `<Property Id="PropertyY" Value="Hello!"/>`
    - `[[PropertyX]] = "Hello!"`

Formatted strings

- **About**
  - Resolved during installation
  - *Windows Installer Reference – Installer Database – Installer Database Reference – Database Tables – Column Data Types – Formatted*
- **Examples**
  - `[%EnvVariableName]` - value of environment variable
  - `[#FileId]` - full path of file (if installed)
  - `[!FileId]` - full path of file in 8.3 format (if installed)
  - …

Variables

- **About**
  - Resolved at compile/build time
  - Format: `$(`*`TYPE`*`.Name)`
- **Types**
  - **var** (defined in WiX source in `define` element; referenced projects)
    - `$(var.ProductVersion) ← <?define ProductVersion="…"/>`
  - **env** (environment variable)
    - `$(env.PATH)` : gets %PATH% environment variable
  - **sys** (pre-defined system variable)
    - $(sys.CURRENTDIR) : current dir of build process
    - $(sys.SOURCEFILEDIR) : dir containing file being processed
    - $(sys.SOURCEFILEPATH) : full path to file being processed
    - $(sys.BUILDARCH) : `Package` element's `Platform` attribute (Intel, x64, Intel64)
  - **wix** (defined via `WixVariable` element or passed via command line)
    - $(wix.Foo) ← `<WixVariable Id="Foo" Value="Bar"/>`
  - **loc** (gets a localized string from a loc[alization] file)
    - $(loc.WelcomeMessage)

Include files

- **Purpose**
  - Share code between multiple projects
- **Usage**

```
<?xml version="1.0" encoding="UTF-8"?>
<Include>
  <?define ProductVersion="1.0.0.0"/>
</Include>
```

<Path_to_Solution>\Common\Common.wxi

```
<?xml version="1.0" encoding="UTF-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
  <?include $(sys.SOURCEFILEDIR)..\Common\Common.wxi/>
  …
  <Product Version="$(var.ProductVersion)" …>
    …
  </Product>
</Wix>
```

<Path_to_Solution>\ProjectDir\Product.wxs

Preprocessor directives

- **Define variable**
  - `<?define Foo="bar"?>`
- **Conditional blocks**

  -
    ```
    <?if …?>
    <?elseif …?>
    <?else?>
    <?endif?>
    ```
- **Checking for defined variables**
  - `<?ifdef Foo?>`
  - `<?ifndef Foo?>`
- **Include a file**
  - `<?include File.wxs?>`
- **Iteration block**

  -
    ```
    <?foreach Foo in "x;y;z"?>
    <?endforeach?>
    ```

Preprocessor expressions

- **When using with `if`, `ifelse`, `ifdef` and `ifndef`**
  - Literals (can be in quotes [quotes are optional])
  - Grouping (using parentheses)
  - Variables (using regular variable syntax)
  - Comparison (`< > <= >= = != ~= not and or`)
- **Example**

```
<?define myValue="3" ?>
<?define system32=$(env.windir)\system32 ?>
<?define B="good var" ?>
<?define C=3 ?>
<?define IExist ?>

<?if $(var.Iexist) ?><!-- true --><?endif?>
<?if $(var.myValue)=6 ?><!-- false --><?endif?>
<?if $(var.myValue)!=3 ?><!-- false --><?endif?>
<?if not "x"="y"?><!-- true --><?endif?>
<?if $(env.systemdrive)=a ?><!-- false --><?endif?>
<?if 3<$(var.myValue)?><!-- false --><?endif?>
<?if $(var.B)="good VAR"?><!-- false --><?endif?>
```

```
<?if $(var.A) and not $(env.MyEnvVariable) ?>
  <!-- false -->
<?endif?>

<?if $(var.A) Or ($(var.B) And $(var.myValue) >=3)?>
  <!-- true -->
<?endif?>

<?ifdef IExist ?>
  <!-- true -->
<?else?>
  <!-- false -->
<?endif?>
```

Launch conditions

- **Purpose**
  - Use rules to enforce prerequisites for installing product, features or components
- **Condition element**
  - **Product** conditions
    - *Message* attribute (message displayed when condition is not met)

```
<Product …>
 <Condition Message="…"><![CDATA[Installed OR VersionNT >= 600]]></Condition>
</Product>
```

  - **Feature** conditions
    - *Level* attribute (if condition is met, this level will be set for the parent feature)

```
<Feature Level="1" …>
  <Condition Level="0">%PROCESSOR_ARCHITECTURE = "x86"</Condition>
</Component>
```

  - **Component** conditions

```
<Component …>
  <Condition><![CDATA[VersionNT < 500]]></Condition>
</Component>
```

Action states

- **Purpose**
  - Determine whether the user requested a particular feature or component to be installed
- **Syntax**
  - Ampersand + ID of feature
    - `&FeatureA = 3`
  - Dollar sign + ID of component
    - `$ComponentX = 1`
- **Values**
  - -1 : Unknown (no action will be taken)
  - 1 : Advertised (feature will be installed on demand)
  - 2 : Absent (feature/component will not be installed)
  - 3 : Local (feature/component will be installed on a local hard drive)
  - 4 : Source (feature/component will run from source, e.g. CD)

Installed states

- **Purpose**
  - Determine whether a particular feature or component is installed
- **Syntax**
  - Features: Exclamation mark + ID of feature
    - `!FeatureA = 3`
  - Components: Question mark + ID of component
    - `?ComponentX = 3`
- **Values**
  - -1 : Unknown
  - 1 : Advertised (feature was installed as advertised)
  - 2 : Absent (feature/component was not installed)
  - 3 : Local (feature/component was installed on a local hard drive)
  - 4 : Source (feature/component runs from source, e.g. CD)

Installation phases

- **UI sequence (*InstallUISequence* table)**
  - Preparation steps
    - Searching for installed components
    - Checking conditions
    - Verifying disk space
    - Selecting type of setup (install/upgrade/repair/etc)
- **Execution sequence (*InstallExecuteSequence* table)**
  - System modifications
  - Performed in two sessions
    - **Client** side: UI operations; runs as interactive user
    - **Server** side: system changes (copying files/etc); runs as *LocalSystem* user
  - Phases
    - **Immediate** phase (preparation to execution; generates rollback script)
    - **Deferred** phase (only actions performed in this phase can be rolled back; mutexed)
      - Custom Actions (CAs) that change the system must be performed in this phase

Custom actions (CAs) in Windows Installer

- **Purpose**
  - Execute custom code or operation
- **CAs and installation phases**
  - See *Installation Phases and In-Script Execution Options for Custom Actions in Windows Installer* by Stefan Krueger (pay attention to rules)
- **CA types for various artifacts**
  - Binary table: 1 (DLL), 2 (EXE), 5 (Jscript), 6 (VBScript)
  - Copied during installation: 17 (DLL), 18 (EXE), 21 (Jscript), 22 (VBScript)
  - Referencing directory: 34 (EXE)
  - Referencing property: 50 (EXE), 53 (Jscript), 54 (VBScript)
  - Literal code in MSI database: 37 (Jscript), 38 (VBScript)
  - Display error and terminate setup: 19
  - Set a property: 51
  - …

Custom actions in WiX

- **CustomAction element**
  - Properties
    - *Id* (text)
    - *Return* (*asyncNoWait*: run in parallel with setup; *asyncWait*: run in parallel, but installer will wait for return code at the end of the execution sequence; *check*: run sequentially and check for success; *ignore*: run sequentially, but don't check for success)
- **Caveats**
  - Accessing a property from a CA executed in a deferred phase
  - Conditional execution
  - CAs in merge modules
- **Detailed explanation and examples**
  - See *From MSI to WiX, Part 5 - Custom actions: Introduction* by Alex Shevchuk

Custom Action example

- **Registering a COM executable via a Custom Action (CA)**

```
<CustomAction Id="RegisterComServer" FileKey="MyComSrv.exe" ExeCommand=" /RegServer"
              Execute="deferred" Impersonate="no" Return="check" />
<CustomAction Id="UnregisterComServer" FileKey="MyComSrv.exe" ExeCommand=" /UnRegServer"
              Execute="deferred" Impersonate="no" Return="check" />
…
<InstallExecuteSequence>
  <Custom Action="RegisterComServer" After="InstallFiles">
    NOT Installed
  </Custom>
  <Custom Action="UnregisterComServer" Before="RemoveFiles">
    Installed AND NOT UPGRADINGPRODUCTCODE
  </Custom>
</InstallExecuteSequence>
```

User interface (UI) in WiX

- **Default project**
  - No UI (no dialogs/UI wizards)
- **To add UI wizards**
  - Add reference to `WiXUIExtensions.dll`
  - Add `UIRef` element with name of specific wizard
  - Define properties expected by the wizard
- **Wizards**
  - WixUI_Minimal (EULA)
  - WixUI_Advanced (scope: single or all users; program folder; feature tree)
  - WixUI_FeatureTree (like WixUI_Advanced; default program folder only)
  - WixUI_InstallDIr (allows to change program folder)
  - WixUI_Mondo (Typical, Complete, or Custom setup)

```
<Property Id="ApplicationFolderName" Value="WiX Demo" />
<Property Id="WixAppFolder" Value="WiXxperMachineFolder" />

<UI Id="UISequence"><UIRef Id="WixUI_Advanced"/></UI>
```

Customizing UI

- **Change existing dialog**
  - Minor adjustments (define bitmaps, license agreements, default folder, etc)
    - Examples: *Customizing Built-in WixUI Dialog Sets*
  - Major changes (remove, add, customize, move controls)
    - Example: *Add a checkbox to the exit dialog to launch the app, or the helpfile*
- **Modify existing wizard (dialog sequence)**
  - Example: *WiX: How to skip LicenseAgreementDlg – more elegant solution*
- **Create your own dialogs**
  - Example: *How do I create a custom dialog in WiX for user input?*

WiX extensions

- **About**
  - Distributed with WiX toolkit (in the `Bin` folder)
  - Similar to .NET class libraries
  - To use, add to project references
- **Popular**
  - WixUIExtension.dll (UI wizards)
  - WixIisExtension.dll (IIS configuration)
  - WixSqlExtension.dll (SQL operations)
  - WixUtilExtension.dll (custom actions and more)

Tools

- **WiX Toolkit**
  - Heat.exe
    - Generates WiX sources from various input formats: projects, DLLs, MSIs, etc
  - See List of Tools
- **Other**
  - Paraffin by John Robbins (alternative to Heat)

Upgrades

- **Minor**
  - Don't use (unless you know why and how)
- **Patch**
  - Don't use (unless you know why and how)
- **Major**
  - Change package code (GUID)
  - Change product version (only first 3 numbers in `ProductVersion` matter)
    - `1.0.0.0 == 1.0.0.1`
    - `1.0.0.1 != 1.0.1.1`
  - Retain original upgrade code (GUID)
  - How-to:
    - StackOverflow: *How to implement WiX installer upgrade?* (read all answers and comments)
    - StackOverflow: *Wix Major Upgrade: how do I prevent Windows service reinstallation?*

Upgrade examples

- **Example #1 (typical)**

```
<Product…>
  <MajorUpgrade
    AllowDowngrade="no"
    AllowSameVersionUpgrades="no"
    Schedule="afterInstallInitialize"
    DowngradeErrorMessage="A later version of [ProductName] is already installed." />
</Product>
```

- **Example #2 (more complex)**

```
<Product…>
  <MajorUpgrade
    AllowDowngrade="no"
    AllowSameVersionUpgrades="no"
    MigrateFeatures="yes"
    Schedule="afterInstallFinalize"
    DowngradeErrorMessage="A later version of [ProductName] is already installed." />
  …
  <InstallExecuteSequence>
    <DeleteServices>NOT UPGRADINGPRODUCTCODE</DeleteServices>
  </InstallExecuteSequence>
</Product>
```

Debugging

- **Use log file**
  - When installing
    - `msiexec /i "<PathToMsi>" /L*v "<PathToLogFile>"`
  - When uninstalling
    - `msiexec /x "<PathToMsi>" /L*v "<PathToLogFile>"`

Limitations

- **Bootstrapper/chainer (setup.exe)**
  - Promised via the BURN tool in WiX ~~3.5~~ 3.6(?) (work in progress)
- **Project output file inclusion**
  - Can be done manually (somewhat automated via the HEAT tool)
- **COM registration info extraction**
  - From executables
- **More**
  - *StackOverflow: What are limitations of WiX and WiX Toolset?*

Resources

- **Book**
  - *WiX: A Developer's Guide to Windows Installer XML* by Nick Ramirez
- **Presentation**
  - *Setup With Windows Installer and WiX* by Jeff Sharp (video and slides)
- **Blogs**
  - *Joy of Setup* by Bob Arnson (check out blog roll)
- **Help**
  - StackOverflow
  - WiX Project Support at SourceForge
- **Tips**
  - StackOverflow: *WiX Tips and Tricks*
- **More**
  - *Learning WiX from ground* up by Alek Davis (see also other parts of the series)