Windows Installer XML

Extended Install Language

Windows-Installer-Dateien erleichtern die ungeliebte Aufgabe, Installationsroutinen bereitzustellen. Die Toolsammlung Windows Installer XML bietet besondere Vorteile beim Erstellen von msi-Dateien: Sie ermöglicht die Arbeit in Teams und die Integration der Technologie in den Build-Prozess.

Die Anzahl der verfügbaren Tools und Anwendungen zum Erstellen von Windows-Installer-Paketen und anderen Windows-Installer-Dateien hat in den vergangenen Jahren ständig zugenommen. Es sind Tools verfügbar, die nahezu allen Anforderungen und Vorlieben entgegen kommen. Diese Werkzeuge verfügen über komfortable und intuitiv zu bedienende Benutzeroberflächen, wodurch die Erstellung von Windows-Installer-Dateien auf einfache Weise realisiert werden kann.

Dennoch existieren Anforderungskriterien, bei denen diese Tools an ihre Grenzen stoßen oder nicht wirkungsvoll in den Entwicklungsprozess des jeweiligen Softwareproduktes integriert werden können. Schwierigkeiten bereitet etwa der Einsatz dieser Tools in Entwicklungsteams. Hierfür müsste eine Möglichkeit existieren, die vom Entwickler definierten Installationseinstellungen in mehreren physischen Dateien abzulegen.

Das ist insofern relevant, als ein Softwareentwickler die Installationseinstellungen für die von ihm entwickelte Komponente selbst definieren sollte und die Gesamtheit dieser Installationsbeschreibungen erst im tatsächlichen Build-Vorgang zum Erzeugen des Installationspaketes verwendet wird.

Nachteile verfügbarer Tools

Weitere Probleme bereiten auch die proprietären Dateiformate, die die Tools zur Speicherung der Installationsbeschreibungen verwenden. Da es sich hierbei im überwiegenden Fall um Binärformate handelt, ist eine effektive Verwaltung in Versionskontrollsystemen wie Concurrent Versioning System (CVS) nur äußerst schwierig realisierbar. Beim Binärformat ist es nicht möglich, eine exakte Visualisierung der Änderungen zu erhalten, die im Entwicklungszeitraum vorgenommen werden. Eine solche Funktionalität ist nur mit Dateien im Textformat problemlos erreichbar.

Ein weiterer gravierender Nachteil der verfügbaren Tools liegt in der unzureichenden Integration in automatisierte Build-Prozesse. Die Werkzeuge stellen gar keine oder nur rudimentär implementierte Schnittstellen zu diesem Zweck zur Verfügung. In der heutigen Zeit ist es jedoch vielfach erforderlich, die Erstellung der entsprechenden Windows-Installer-Dateien in einen automatisierten Build-Prozess zu integrieren, sodass eine solche Integration nur durch eine individuelle Lösung realisiert werden kann, die zu diesem Zweck das Windows-Installer-API direkt verwendet.

Jeder, der bereits Erfahrungen mit diesem API gesammelt hat, wird dabei aber festgestellt haben, dass die Umsetzung einer solchen Lösung zwar möglich ist, aber ein sehr hohes Technologieverständnis erfordert.

Bei Windows Installer XML (WiX) handelt es sich um eine Sammlung von Tools und Spezifikationen, mit denen die Inhalte eines Windows-Installer-Paketes (msi), eines Merge-Moduls (msm) oder eines Patch Creation Property Files (pcp) in XML-Dokumenten definiert und anschließend in die entsprechende Windows-Installer-Datei überführt werden.

Der große Vorteil dieser Toolsammlung liegt in der Verwendung von Textdateien zur Speicherung der Installationsinformationen, der Aufteilung des Installationsprojektes in mehrere physische Dateien sowie der hervorragenden Integrationsmöglichkeit in automatisierte Build-Prozesse.

Die Installationsbeschreibung erstellen

Zur Erstellung einer Windows-Installer-Datei unter Einsatz von Windows-Installer-XML ist zunächst der Programmcode zu verfassen. Der Programmcode wird hierbei in einem wohl geformten (well-formed) XML-Dokument beschrieben. Es verfügt über die Dateiendung *wxs* und enthält eine Auflistung aller Elemente, die in das spätere Installationspaket integriert werden sollen.

Von einem wohl geformten XML-Dokument wird gesprochen, wenn dieses den W3C-Empfehlungen entspricht, wozu die nachfolgenden Bedingungen erfüllt sein müssen:

- *Ein XML-Dokument kann nur ein Stammelement enthalten*: Beim Stammelement eines XML-Dokumentes handelt es sich um ein einzelnes Element, das den gesamten, als Teil des Dokuments geltenden Inhalt enthält. Das Stammelement ist das erste Element, das hinter dem

Auf einen Blick

Autor

Andreas Kerl ist Application Development Consultant im Premier Support for Developers der Microsoft Deutschland GmbH. Er hat das erste Buch zum Windows Installer 3.1 geschrieben. Sie erreichen ihn unter andreaskerl@msn.com.

dotnetpro.code
A0510Installer

Sprachen Windows Installer XML

Technik Windows Installer 3.1

Voraussetzungen Visual Studio .NET 2003, Windows Installer, Windows Installer XML

Listing I

Ein WiX-Dokument fasst alle Informationen für das Erstellen eines Installationspaketes zusammen.

```
<?xml version="1.0" encoding="windows-1252"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2003/01/wi">
  <!- Definition des Produktes ->
  <Product Id="AA387173-BADA-4261-8B81-40207CDD14EB" UpgradeCode="4BE21D77-
    BA5D-42EC-9A1F-BCE9F3B5D82B"
    Manufacturer="Microsoft Deutschland GmbH" Language="1031"
    Version="1.00.0000" Name="Darwin Descriptor 1.0" Codepage="1252">

    <!- Paketdefinition ->
    <Package Id="????????-????-????-????-????????????"
      Keywords="Microsoft Windows Installer, MSI"
      Description="Darwin Descriptor" Comments="Installation mit WiX."
      Manufacturer="Andreas Kerl" InstallerVersion="300"
      Platforms="Intel" Languages="1031"
      Compressed="yes" SummaryCodepage="1252" />

    <!- Medien ->
    <Media Id="1" EmbedCab="yes" Cabinet="Data.cab" />

    <!- Ordner, Komponenten und Ressourcen ->
    <Directory Id="TARGETDIR" Name="SourceDir">
      <Directory Id="ProgramFilesFolder">
        <Directory Id="INSTALLLOCATION" Name="DARWIN"
          LongName="Darwin Descriptor 1.0">
          <Component Id="DD.exe"
            Guid="6ABBEDC4-3D20-4901-BBAE-D6D16127571A">
            <File Id="DD.exe" src=".\Binaries\" Name="DD.exe"
              KeyPath="yes" AssemblyManifest="DD.exe"
              AssemblyApplication="DD.exe"
              Assembly=".net" DiskId="1" />
          </Component>
          <Component Id="Common.dll"
            Guid="332C7990-C503-4992-8494-4151FEF834AB">
            <File Id="Common.dll" src=".\Binaries\" Name="Common.dll"
              KeyPath="yes" AssemblyManifest="Common.dll"
              AssemblyApplication="Common.dll"
              Assembly=".net" DiskId="1" />
          </Component>
        </Directory>
      </Directory>
    </Directory>

    <!- Definition der Featurestruktur ->
    <Feature Id="Application" Title="Anwendung"
      Description="Installiert die Anwendung" Level="1">
      <ComponentRef Id="DD.exe" />
      <ComponentRef Id="Common.dll" />
    </Feature>
    <Property Id="INSTALLLEVEL" Value="3"/>
    <Property Id="ARPHELPLINK" Value="www.microsoft.com/germany" />

    <!- Systemvoraussetzung ->
    <Condition Message="Das Microsoft .NET Framework ist zur Ausführung
der Applikation unbedingt erforderlich.">MsiNetAssemblySupport</Condition>
  </Product>
</Wix>
```

Prolog des Dokuments angezeigt wird. Das Stammelement wird auch als Dokumentelement bezeichnet.

- *Alle XML-Elemente müssen End-Tags enthalten*: Obgleich End-Tags in bestimmten HTML-Dokumentelementen optional sind, müssen alle Elemente in einem XML-Dokument ein End-Tag enthalten.
- *Die Namen des Start- und End-Tags eines Elements müssen identisch sein*: Bei XML wird die Groß-/Kleinschreibung beachtet, daher muss der Name eines End-Tags exakt mit dem Namen des entsprechenden Start-Tags übereinstimmen.
- *XML-Elemente dürfen sich nicht überlappen*: Falls das Start-Tag eines Elements innerhalb eines anderen Elements auftritt, muss es innerhalb desselben Elements enden.
- *Alle Attributwerte müssen in Anführungszeichen gesetzt sein*: Attributwerte müssen in Hochkommas oder in Anführungszeichen gesetzt sein.
- *Die Zeichen <, > und & dürfen nicht im Text eines XML-Dokumentes verwendet werden*: Hierbei handelt es sich um Sonderzeichen, die für XML-Parser eine besondere Bedeutung haben. Für diese Zeichen sollten Sie vordefinierte Zeichen oder Entitätsverweise verwenden.

Für das WiX-Dokument bedeutet dies, dass das Stammelement über die Bezeichnung <*Wix*> verfügen und eines der Elemente <*Product*>, <*Module*>, <*PatchCreation*> oder <*Fragment*> als untergeordnetes Element enthalten muss. Die Auswahl des zu verwendenden untergeordneten Elementes richtet sich nach dem Verwendungszweck der Windows-Installer-Datei. Das Element <*Product*> dient zur Erstellung eines Windows-Installer-Paketes, das Element <*PatchCreation*> zur Erstellung einer PCP-Datei und das Element <*Module*> zur Erstellung eines Windows-Installer-Merge-Moduls. Unterhalb dieser Elemente werden wie in Listing 1 die weiteren Informationen zur Definition der endgültigen Windows-Installer-Datei angeordnet.

In Listing 1 ist zu erkennen, dass das Element <*Product*> unterhalb des Stammelementes verwendet wird, sodass hierdurch ein Windows-Installer-Paket erzeugt wird. Das Element <*Product*> enthält hierbei weitere Attribute, die das Installationsprodukt definieren und zu einem späteren Zeitpunkt in die Tabelle *Property* der Installationsdatenbank übertragen werden.

Die Informationen, die in den *Summary Information Stream* der zu erstellenden Windows-Installer-Datei übertragen werden sollen, werden durch das Element <*Package*> beschrieben. In dem Listing ist weiterhin zu erkennen, dass einige Eigenschaften in Form einer GUID angegeben werden. Windows Installer XML erwartet, dass GUIDs ohne geschweifte Klammern verwendet werden. Zur Generierung solcher GUIDs enthält Microsoft Visual Studio .NET ein Befehlszeilentool mit der Bezeichnung *uuidgen.exe*. Darüber hinaus besteht auch die Möglichkeit, eine GUID bei jedem Kompiliervorgang durch Windows-Installer-XML automatisch erzeugen zu lassen. Hierzu ist die Zeichenfolge *????????-????-????-????-????????????* dem jeweiligen Attribut zuzuweisen.

Die wesentlichen Elemente innerhalb eines Installationspaketes sind die Features, die Komponenten und die zu installierenden Ressourcen. Das Festlegen der Komponenten und der Ressourcen erfolgt in einem WiX-Dokument innerhalb der zu verwendenden Ordnerstruktur. Hierdurch

ist eine bessere Übersicht gewährleistet, da sehr schnell das Installationsverzeichnis der jeweiligen Ressourcen ermittelt werden kann.

In Listing 1 ist zu erkennen, dass alle Komponenten unterhalb des Verzeichnisses *INSTALLLOCATION* angeordnet werden, sodass die enthaltenen Ressourcen bei der Installation in dieses Verzeichnis kopiert werden. Jede Komponente ist mit einer *Id* versehen, durch die Verknüpfungen zu Features ermöglicht werden. Die zu installierenden Ressourcen sind wiederum innerhalb der Komponente definiert, wodurch die Zuordnung zu der Komponente realisiert wird. Zur Festlegung der Dateien steht eine Vielzahl von Attributen bereit, wobei die Attribute *Id* und *Name* immer verwendet werden müssen. Weiterhin ist festzulegen, in welches Installationsmedium diese Dateien integriert werden sollen. Zu diesem Zweck steht das Attribut *DiskId* zur Verfügung, das auf Komponenten- oder Dateiebene verwendet werden kann und auf einen Eintrag des Elementtyps <*Media*> verweisen muss.

Handelt es sich bei den Dateien um Assemblies (.NET oder Win32), sind die Attribute *Assembly* und *AssemblyManifest* zu definieren, wodurch automatisch die entsprechenden Eintragungen in den Tabellen *MsiAssembly* und *MsiAssemblyName* vorgenommen werden. Während des Erstellungsvorgangs werden die weiteren Informationen wie die Größe der Datei, die Dateiversion et cetera automatisch ermittelt und in die Tabelle *File* übertragen.

Handelt es sich um eine Datei ohne Versionsangabe, werden automatisch die Eintragungen in der Tabelle *MsiFileHash* vorgenommen. Die Referenzierung der physischen Dateien erfolgt durch das Attribut *src*, dem ein absoluter oder relativer Pfad auf die jeweiligen Dateien zugewiesen wird. In dem Listing wird hierzu die Zeichenfolge .*Binaries*\ verwendet, sodass die erforderlichen Ressourcen aus dem Unterverzeichnis *Binaries* verwendet werden. Die Vorgehensweise zur Verwendung von anderen Ressourcenarten wie Dateiverknüpfungen, Eintragungen der Systemregistrierung, Betriebssystemdiensten und anderen ist entsprechend umzusetzen. Eine genaue Auflistung der verfügbaren Attribute für die einzelnen Elementtypen finden Sie in der Hilfe zu Windows Installer XML.

Jedes Installationspaket muss über mindestens ein Feature verfügen, dem die jeweiligen Komponenten zugewiesen werden. Diese Zuordnung von Komponenten zu Features wird durch das Element <*ComponentRef*> realisiert. Diese Informationen werden später in die Tabelle *FeatureComponents* übertragen.

Im Wesentlichen ist das Installationspaket nun definiert und lässt sich erstellen und später auch auf dem System installieren. Es ist natürlich noch eine Vielzahl weiterer Installationseinstellungen möglich, wie beispielsweise die Festlegung von Systemvoraussetzungen über das Attribut <*Condition*> oder die Festlegung weiterer Eigenschaften mit Hilfe der Attribute <*Property*>.

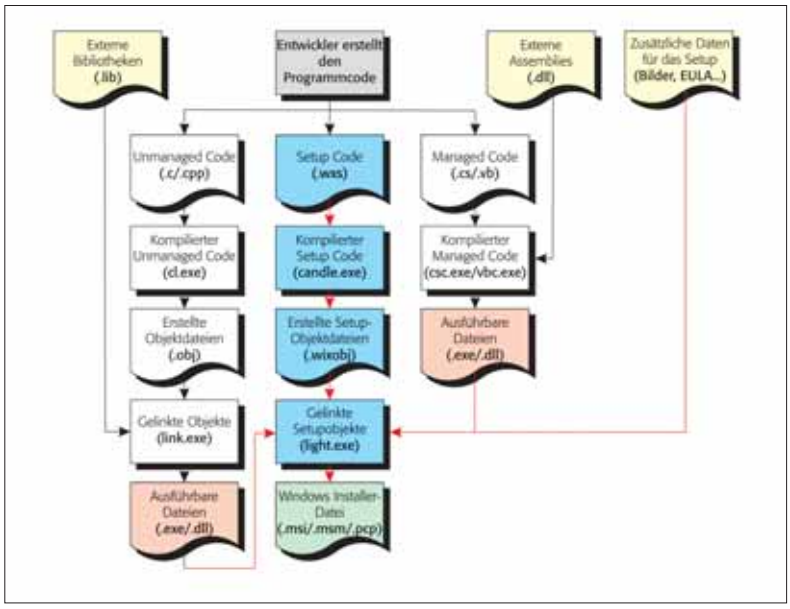

Abbildung 1 Windows Installer XML fügt sich nahtlos in den Entwicklungsprozess ein.

Kompilieren und Linken

Nachdem Sie die erforderlichen Informationen in ein WiX-Dokument übertragen haben, sind alle Voraussetzungen geschaffen, diese in ein funktionsfähiges Installationspaket zu überführen. Die Toolsammlung Windows Installer XML enthält zu diesem Zweck diverse Anwendungen, Objektbibliotheken und weitere Dateien. Tabelle 1 fasst die in der Toolsammlung enthaltenen Komponenten zusammen.

Der große Vorteil von Windows Installer XML liegt darin, dass es sich am klassischen Erstellungsprozess für Software orientiert, wie es auch Abbildung 1 verdeutlicht.

Der erste Schritt hierbei umfasst das Erstellen des Programmcodes, also das

Tabelle 1

Bestandteile der Toolsammlung Windows Installer XML.

Tool	Beschreibung
wix.xsd	Schema für Windows-Installer-Pakete, Merge-Module und PCP-Dateien
wixloc.xsd	Schema für Windows-Installer-XML-Sprachdateien
candle.exe	Compiler
light.exe	Linker
wix.chm	Hilfedatei
wixca.dll / sca*.dll	Objektbibliotheken zur Integration von vordefinierten benutzerdefinierten Aktionen (IIS, SQL-Server, Benutzerkonten, Performancecounter etc.)
dark.exe	Decompiler
tallow.exe	Befehlszeilentool, mit dem Dateien eines bestimmten Verzeichnisses in ein WiX-Dokument integriert werden können
wix.dll	Bibliothek, die alle Funktionalitäten enthält und daher für programmtechnische Ansätze verwendet werden kann (C#).

Erstellen des WiX-Dokumentes, wie es bereits beschrieben wurde. Im nächsten Schritt findet der Kompiliervorgang statt. Hierbei werden die Informationen des XML-Dokumentes auf Gültigkeit geprüft, die Variablen aufgelöst und das Ergebnis in Objektdateien (*wixobj*) abgelegt.

Der wesentliche Aspekt innerhalb dieses Vorgangs ist das Prüfen der Daten auf Gültigkeit, wozu die Schemadatei *wix.xsd* verwendet wird. Sie starten den Kompiliervorgang, indem Sie den Compiler *candle.exe* über die folgende Befehlszeile aufrufen und der Befehlszeile die erforderlichen Argumente, wie etwa die Referenz auf das WiX-Dokument, übergeben.

```
candle.exe [-out outputFile] sourceFile
[sourceFile ...]
```

Nachdem der Kompiliervorgang erfolgreich abgeschlossen ist, kann aus den erstellten Objektdateien schließlich die Windows-Installer-Datei erzeugt werden. Hierbei werden die Inhalte einer oder mehrerer Objektdateien (*wixobj*) ausgewertet, mit den Metainformationen externer Dateien kombiniert und in die Windows-Installer-Datei übertragen. Weiterhin werden während dieses Vorgangs die Kabinettdateien erstellt und diese und weitere Ressourcen wie beispielsweise die Bitmaps zur Darstellung der Benutzeroberfläche in die Windows-Installer-Datei integriert.

Das Zusammenführen der Dateien und das Schreiben der notwendigen Informationen realisieren Sie durch einen Linker, der in Windows Installer XML die Bezeichnung *light.exe* trägt. Den Linker rufen Sie über folgende Befehlszeile auf:

```
light.exe [-out outputFile] objectFile
[objectFile ...]
```

Nach dem Linken ist der Erstellungsprozess abgeschlossen und es wurde die entsprechende Windows-Installer-Datei erzeugt. Sollte es zu einem späteren Zeitpunkt erforderlich werden, das Windows-Installer-Paket wieder in ein WiX-Dokument zu konvertieren, steht ein Decompiler mit der Bezeichnung *dark.exe* zur Verfügung.

Aus dieser Beschreibung über den Einsatz von Windows Installer XML wird deutlich, dass das Erzeugen von Installationspaketen hervorragend in automatisierte Build-Prozesse integriert werden kann. Einerseits bietet die Verwendung von XML-Dokumenten zur Beschreibung der Inhalte eines Installationspaketes einen sehr hohen Automatisierungsgrad und zum anderen können Sie die benötigten Tools vollständig von der Befehlszeile aus steuern.

Darüber hinaus stellt Windows Installer XML eine Bibliothek mit der Bezeichnung *wix.dll* zur Verfügung, die Sie auch in eigene Anwendungen integrieren oder zu Automatisierungszwecken verwenden können. Diese in C# entwickelte Bibliothek stellt die gerade beschriebenen Funktionalitäten durch die Klassen *Compiler*, *Linker* und *Decompiler* zur Verfügung, die sich alle im Namensraum *Microsoft.Tools.WindowsInstallerXml* befinden.

Präprozessor

Zur Erlangung der größtmöglichen Flexibilität bei der Gestaltung der WiX-Dokumente bietet Windows Installer XML die Möglichkeit, diverse Präprozessorelemente in dem Dokument zu verwenden. Die Präprozessorelemente werden durch den Compiler aufgelöst, bevor die eigentliche Objektdatei erzeugt wird. Somit wird es hierdurch unter anderem möglich, mit einem WiX-Dokument mehrere Zielprodukte zu erstellen, wobei Sie die individuellen Produktkonfigurationen über den Befehlszeilenaufruf steuern können. Ein Typ dieser Präprozessorelemente sind die Variablen, von denen Windows Installer XML die folgenden Typen kennt:

- Umgebungsvariablen
- Systemvariablen
- benutzerdefinierte Variablen

Unabhängig von der zu verwendenden Art der Variablen sind diese nach dem Schema *$(Typ.VarName)* im Dokument zu definieren, wobei als Typ für Umgebungsvariablen die Zeichenfolge *env*, für Systemvariablen *sys* und für benutzerdefinierte Variablen *var* zu verwenden ist. Soll beispielsweise die Umgebungsvariable *%Systemdrive%* im WiX-Dokument referenziert werden, so ist dies durch Verwendung der Zeichenfolge *$(sys.Systemdrive)* umsetzbar. Systemvariablen werden vom Windows Installer XML zur Verfügung gestellt und enthalten Informationen über das jeweilige WiX-Projekt.

In der momentanen Version von Windows Installer XML stehen die Systemvariablen *CURRENTDIR*, *SOURCEFILEPATH* und *SOURCEFILEDIR* zur Verfügung. Zu beachten ist hierbei, dass Systemvariablen ausschließlich in Großbuchstaben defi-

Listing 2

Benutzerdefinierte Variablen mit Include-Datei verwalten.

```
<?xml version="1.0" encoding="utf-8"?>

<Include>

   <?define SourceFolder     = ".\Binaries\" ?>
   <?define HelpGuid         = "079D5F69-A2B5-4760-AEF9-6644F8F246A0" ?>
   <?define ReadmeGuid       = "4E7D6BDD-6B35-4731-9474-1C028F446F65" ?>

    <?if $(var.Flavor) = "debug" ?>

        <?define ProductName     = "Darwin Descriptor 1.0 (Debug)"?>
        <?define ProductId       = "C418DE3A-7B13-426A-A054-E9793DD583FA" ?>
        <?define AppBinariesGuid = "17F1A33F-BF41-4681-8E4E-2DA613244015" ?>
        <?define LibBinariesGuid = "26518869-8F1F-4A9F-9EB1-AF6E240423D8" ?>
        <?define BinaryFolder    = ".\Binaries\Debug\" ?>

    <?else ?>

        <?define ProductName     = "Darwin Descriptor 1.0"?>
        <?define ProductId       = "4916154D-6FA2-4729-945B-C34018467549" ?>
        <?define AppBinariesGuid = "B7656789-A32E-463D-B7E7-9920BB9765AE" ?>
        <?define LibBinariesGuid = "82B76300-2A08-4B6A-8773-1CACD9D1D8D1" ?>
        <?define BinaryFolder    = ".\Binaries\Release\" ?>

    <?endif ?>

</Include>
```

Listing 3

Sprachdatei für die Lokalisierung.

```
<?xml version="1.0" encoding="utf-8"?>
<WixLocalization xmlns='http://schemas.microsoft.com/wix/2003/11/localization' Codepage="1252" >
   <String Id="Lang">1033</String>
   <String Id="Name">Darwin Descriptor 1.0 (English)</String>
   <String Id="Comment">This installer database contains the logic and data required to install
                       "Darwin Descriptor 1.0".</String>
   <String Id="InstallFolder">Darwin Descriptor 1.0 (English)</String>
   <String Id="FeatureTitle">Complete</String>
   <String Id="FeatureDescription">Installed all Features</String>
</WixLocalization>
```

niert sind und dass hierbei die Groß- und Kleinschreibung zu berücksichtigen ist. Windows Installer XML bietet auch die Möglichkeit, benutzerdefinierte Variablen im jeweiligen Dokument zu verwenden. So ist es beispielsweise möglich, die Quelldateien beziehungsweise den Ordner, der die Quelldateien enthält, durch eine benutzerdefinierte Variable zu definieren:

```
<File Id="DD.exe" src="$(var.
SourceFolder)\DD1.exe" Name="DD.exe"
KeyPath="yes" AssemblyManifest="DD.exe"
AssemblyApplication="DD.exe" />
```

Die verwendete Variable muss jedoch zuvor im Dokument definiert werden, wozu das Element *<? define ?>* nach dem Schema *<?define SourceFolder = ".\Binaries\" ?>* zu verwenden ist. Zur effizienten Verwaltung der benutzerdefinierten Variablen bietet Windows Installer XML zusätzlich die Möglichkeit, so genannte Include-Dateien zu verwenden. Diese Dateiart ist vergleichbar mit den Header-Dateien (h) in C++ und hat die Dateierweiterung *wxi*.

Listing 2 zeigt ein Beispiel für eine Include-Datei, in der wiederum die benutzerdefinierte Variable *$(var.Flavor)* verwendet wird. Der Wert dieser Variablen ist in dem Beispiel jedoch nicht definiert, sondern wird erst beim tatsächlichen Kompiliervorgang gesetzt, indem der Wert an den Befehlszeilenaufruf angefügt wird:

```
candle.exe  Product.wxs -dFlavor=debug
```

Zusätzlich zu den Variablen stehen unter Windows Installer XML auch die Präprozessorelemente *Bedingung* und *Iteration* zur Verfügung, wobei Bedingungen durch die folgenden Elemente definiert werden können:

<?if ?>
<?ifdef ?>
<?ifndef ?>
<?else?>
<?elseif ?>
<?endif?>

Durch die Verwendung der Bedingungen ist es beispielsweise möglich, den Umfang der zu installierenden Ressourcen anhand der Projektkonfiguration einzuschränken oder zu erweitern:

```
<!– Falls Debug die Symboldatei einbinden –>
<?if $(var.Flavor) = "debug" ?>
   <File Id="DD.pdb" src="$(var.BinaryFolder)"
    Name="DD.pdb" KeyPath="no" DiskId="1" />
<?endif ?>
```

Eine Iteration erreichen Sie durch *For-Each*-Statements. Das folgende Beispiel, in dem dem Installationspaket für jede unterstützte Sprache eine neue Komponente hinzugefügt wird, zeigt die Implementierung dieser Möglichkeit.

```
<?define LcidList=1033;1041;1055?>
<?foreach LCID in $(var.LcidList)?>
      <DirectoryRef Id='TARGETDIR'>
            <Component
             Id='MyComponent.$(var.LCID)' />
      </DirectoryRef>
<?endforeach?>
```

Die dargestellten Präprozessorelemente sind natürlich miteinander kombinierbar. So ist es möglich, die benutzerdefinierte Variable *LcidList* des Iterationsbeispiels in eine Include-Datei auszulagern oder die zu verwendenden Elemente durch Bedingungen einzuschränken oder zu erweitern.

Lokalisierung

Die Lokalisierung, also die Bereitstellung eines Installationspaketes in mehreren Sprachen, ist mit den gerade bezeichneten Möglichkeiten der Variablen und Include-Dateien problemlos möglich. Dennoch stellt Windows Installer XML für diese Zwecke spezielle Sprachdateien zur Verfügung, die durch die Dateierweiterung *wxl* gekennzeichnet sind und gegen das Schema *wixloc.xsd* validiert werden. Diese speziellen Sprachdateien sind vergleichbar mit Ressourcendateien der klassischen Softwareentwicklung.

Der große Vorteil der Sprachdateien begründet sich darin, dass sie erst während des Linkens benötigt werden. Eine benutzerdefinierte Variable muss hingegen bereits zur Kompilierzeit definiert sein, sodass die erzeugte Objektdatei (*wixobj*) bereits den endgültigen Wert dieser Variablen enthält. Bei der Verwendung von Sprachdateien wird die Definition der Variablen hingegen erst während des Linkens geprüft und ausgewertet. Hierdurch ist es möglich, eine sprachneutrale Objektdatei durch den Compiler erstellen zu lassen und diese zu einem späteren Zeitpunkt als Quelle für die lokalisierten Pakete zu verwenden, was vom zeitlichen Aspekt aus gesehen natürlich äußerst effizient ist. Werden Sprachdateien im Projekt verwendet, müssen Sie diese der Befehlszeile des Linkers mit dem Argument *-loc* übergeben.

```
candle.exe  Product.wxs
light.exe -out 1031.msi Product.wixobj -loc
1031.wxl
light.exe -out 1033.msi Product.wixobj -loc
1033.wxl
```

Sprachdateien sind ebenfalls im XML-Format zu definieren, wobei das Stammelement *<WixLocalization/>* zu verwenden ist, wie es Listing 3 zeigt.

Die lokalisierten Informationen werden im Hauptdokument wie Variablen verwendet, wobei als Typ *loc* anzugeben ist. Um beispielsweise auf die Zeichenfolge *Name* der Sprachdatei aus Listing 3 zuzugreifen, verwenden Sie im Hauptdokument die Referenz *$(loc.Name)*.

Fragmente

Eine Anforderung an ein geeignetes Tool zum Erstellen von Installationspaketen betrifft – gerade beim Einsatz in Entwicklungsteams – die Aufteilung des Gesamtdokumentes in einzelne physische Dateien. Die involvierten Entwickler können bei solchen Voraussetzungen die installationsspezifischen Beschreibungen der von ihnen entwickelten Komponente selbst vornehmen. Erst im späteren

Listing 4

Das Gesamtdokument lässt sich in Fragmente aufteilen.

```
<?xml version="1.0" encoding="windows-1252" ?>
<Wix xmlns="http://schemas.microsoft.com/wix/2003/01/wi">
   <!- Definition des Produktes ->
   <Fragment Id="Common">
      <!- Ordner, Komponenten und Ressourcen ->
      <DirectoryRef Id="INSTALLLOCATION">
         <Component Id="C_Common" Guid="332C7990-C503-4992-8494-4151FEF834AB">
            <File Id="Common.dll" src=".\Binaries\" Name="Common.dll" KeyPath="yes"
                  AssemblyManifest="Common.dll"
                  AssemblyApplication="Common.dll" Assembly=".net" DiskId="1" />
         </Component>
      </DirectoryRef>
      <!- Definition der Featurestruktur ->
      <FeatureRef Id="Application">
         <ComponentRef Id="C_Common" />
      </FeatureRef>
   </Fragment>
</Wix>
```

Build-Prozess fließen diese Informationen in das Gesamtdokument ein.

Windows Installer XML unterstützt diese Anforderungen, indem es den Dokumenttyp *<Fragment>* zur Verfügung stellt. Dieser Dokumenttyp ist vergleichbar mit der Funktionalität der Partial Classes im Microsoft .NET Framework 2.0. Zur Definition eines Fragmentes sind wiederum der Prolog und das Stammelement in einem neuen WiX-Dokument zu definieren. Anschließend ist das Element *<Fragment>* einzufügen. Hierdurch wird der Bereich gekennzeichnet, der während des Linkens in das Hauptdokument übertragen wird. Listing 4 zeigt ein Beispiel für ein Fragment.

In dem Listing ist gut zu erkennen, dass innerhalb des Fragmentes die benötigte Installationsstruktur durch Referenzen zum Hauptdokument realisiert wird. So wird beispielsweise die Komponente *C_Common* dem Installationsverzeichnis zugeordnet, das im Hauptdokument mit *INSTALLLOCATION* bezeichnet wurde.

Die Zuordnung der Komponente zu einem Feature wird ebenfalls über Referenzelemente realisiert. Nachdem diese Informationen definiert wurden, muss das Hauptdokument noch Kenntnis über die Verwendung des Fragments erhalten, wozu das Element *FragmentRef* benötigt wird.

Um das Fragment aus Listing 4 in ein Dokument einzubinden müssen Sie das Statement

```
<FragmentRef Id=' Common' />
```

in das Hauptdokument einfügen. Beim Kompilieren und beim Linken ist weiterhin zu beachten, dass die physische Datei, in der das Fragment definiert wurde, an die entsprechenden Befehlszeilen angefügt wird.

```
candle.exe Product.wxs Fragment.wxs
light.exe -out Setup.msi Product.wixobj
Fragment.wixobj
```

Fragmente sind vielseitig verwendbar und eignen sich ebenfalls hervorragend zur Definition einer Benutzeroberfläche, da hierdurch eine strukturierte Verwendung in späteren Installationsprojekten ermöglicht wird.

Benutzerdefinierte Aktionen

Eines der vielfältigsten Probleme beim Erstellen eines Installationspaketes betrifft die Integration von Funktionalitäten, die über den normalen Funktionsumfang wie das Kopieren und Löschen von Dateien, das Schreiben von Informationen in die Systemregistrierung oder das Installieren von Betriebssystemdiensten hinausgehen. Vielfach besteht die Notwendigkeit, komplexe Aktionsabläufe, wie beispielsweise das Anlegen von Benutzerkonten oder das Erstellen von Datenbanken auf dem Microsoft SQL Server, in den Installationsprozess zu integrieren.

Der Windows Installer stellt für solche Zwecke eine Schnittstelle zur Verfügung, durch die es möglich ist, individuellen Programmcode während der Installation auszuführen. Diese Schnittstelle wird als benutzerdefinierte Aktion oder Custom Action bezeichnet.

Die Integration von benutzerdefinierten Aktionen in das Installationspaket stellt eine der anspruchvollsten Tätigkeiten im Entwicklungsprozess dar, da eine mangelhafte Implementierung den gesamten Installationserfolg negativ beeinflussen kann.

Eine Zielsetzung bei der Entwicklung von Windows Installer XML betrifft die Bereitstellung von vordefinierten Aktionen für die am häufigsten benötigten Funktionsanforderungen. Die Toolsammlung Windows Installer XML stellt aus diesem Grund mehrere Kategorien von benutzerdefinierten Aktionen zur Verfügung. Die benutzerdefinierten Aktionen der Kategorie „Standard“ erweitern die Funktionalität von Standardaktionen der Windows-Installer-Technologie. Diese Art von benutzerdefinierten Aktionen wird durch die Objektbibliothek *wixca.dll* zur Verfügung gestellt, wodurch die folgenden Tätigkeiten ermöglicht werden:

- *Sicherheit*: Absicherung von Objekten durch Zugriffssteuerlisten (ACL), wobei diese Aktionen weit über die Möglichkeiten der Tabelle *LockPermissions* hinausgehen;
- *Dienste*: Festlegen von speziellen Konfigurationseinstellungen für Betriebssystemdienste, die unter Verwendung der Tabellen *ServiceInstall* und *ServiceControl* nicht möglich sind;
- *Konsolenanwendungen*: Ausführen von Konsolenanwendungen, wobei die Anzeige des Konsolenfensters verhindert wird;
- *Drucken*: Ausdrucken der EULA oder sonstiger Dokumente während des Installationsprozesses.

Bei der Verwendung dieser Aktionen müssen Sie dem Aufruf des Linkers einen Verweis auf die Objektdatei *wixca.wixlib* anfügen.

Die zweite Kategorie enthält benutzerdefinierte Aktionen, die speziell im Serverumfeld anzusiedeln sind und daher vielfältige und komplexe Konfigurations- und Überwachungseinstellungen vornehmen können.

- *Internet Information Services (IIS)*: Erstellen und Konfigurieren von Webseiten, virtuellen Verzeichnissen und Webanwendungen;
- *SQL-Server*: Erstellen von Datenbanken und Ausführen von SQL-Skripten;

Listing 5

Ein Benutzerkonto hinzufügen.

```
<?xml version="1.0"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2003/01/wi">
   <Product Id="A53DC573-DC43-4E92-A6E5-2D0AE356DD6A" Name="Account"
            Language="1031" Version="1.0.0.0"
            Manufacturer="Microsoft Deutschland GmbH" Codepage="1252"
            UpgradeCode="A7959C0B-C6A9-4d59-AA6A-A241C93361B1">
      <Package Id="????????-????-????-????-????????????"
               Description="Benutzerkonto hinzufügen" InstallerVersion="300"
               Compressed="yes" />

      <Media Id="1" Cabinet="Data.cab" EmbedCab="yes" />
      <Property Id="ACCOUNTNAME" Value="MSI" Secure="yes"/>

      <!- Nur zu Demonstrationszwecken ->
      <!- Passwort sollte über die Befehlszeile oder UI angegeben werden ->
      <Property Id="PASSWORD" Value="ABCDE*1" Secure="yes" Hidden="yes"/>

      <Directory Id="TARGETDIR" Name="SourceDir">
         <Directory Id="ProgramFilesFolder">
            <Directory Id="INSTALLLOCATION" Name="Konto"
                       LongName="Benutzerkonto">
               <Component Id="C_Account"
                          Guid="08B48FA6-F178-4A4B-861A-81E94F80CB68">
                  <File Id="Dummy.txt" src="Dummy.txt" Name="Dummy.txt"
                        LongName="Dummy.txt" DiskId="1" />

                  <!- Hinzufügen oder aktualisieren des Benutzers ->
                  <User Id="U_User1" Name="[ACCOUNTNAME]"
                        Password="[PASSWORD]" CreateUser="yes"
                        PasswordNeverExpires="yes" RemoveOnUninstall="yes"
                        UpdateIfExists="yes" />
               </Component>
            </Directory>
         </Directory>
      </Directory>

      <Feature Id="ProductFeature" Title="Feature Title" Level="1">
         <ComponentRef Id="C_Account" />
      </Feature>
   </Product>
</Wix>
```

- *Benutzer*: Erstellen und Konfigurieren von Benutzerkonten;
- *Netzwerk*: Erstellen und Konfigurieren von Netzwerkfreigaben;
- *Systemmonitor*: Installieren und Deinstallieren von Leistungsindikatoren (Performance Counter).

Die Metainformationen der benutzerdefinierten Aktionen dieser Kategorie sind in der Objektdatei *sca.wixlib* abgelegt. Die Funktionalität zum Ausführen der Aktionen wird hingegen durch die Laufzeitbibliotheken *scasched.dll* und *scaexec.dll* bereitgestellt, sodass diese Dateien später in das Installationspaket integriert werden müssen. Gerade die benutzerdefinierten Aktionen dieser Kategorie stellen einen sehr umfangreichen Funktionsvorrat zur Verfügung. Unter Verwendung dieser Möglichkeiten ist es sehr einfach, ein neues Benutzerkonto dem lokalen System hinzufügen, wie Listing 5 zeigt. Es ist zu erkennen, dass alle erforderlichen Informationen zum Verwalten eines Benutzerkontos durch das Element <*User*> bereitgestellt werden. Diesem Element können der Name des Benutzers, die Domäne, das Passwort und eine Vielzahl weiterer Argumente übergeben werden.

Alle diese Informationen werden in einer benutzerdefinierten Tabelle mit der Bezeichnung *User* im zu erstellenden Installationspaket abgelegt. Der Zugriff auf diese Informationen wird durch Einträge in der Tabelle *CustomAction* realisiert, die von der Tabelle *InstallExecuteSequence* verwendet werden.

Darüber hinaus müssen die erforderlichen Bibliotheken (*scasched.dll* und *scaexec.dll*) in das Installationspaket integriert werden. Dazu fügen Sie der Befehlszeile zum Aufruf des Linkers eine Referenz auf die Objektdatei an.

```
candle.exe Account.wxs -out Account.wixout
light.exe -out Account.msi Account.wixout
"D:\WiX\ca\sca.wixlib"
```

Stellen Sie sicher, dass sich die Laufzeitbibliotheken im gleichen Verzeichnis befinden wie die entsprechende Objektdatei (*sca.wixlib*).

Zusätzlich zu den bereits erläuterten Kategorien von benutzerdefinierten Aktionen ist es mit Windows Installer XML auch möglich, Gerätetreiber für die Plattformen Microsoft Windows 2000, Microsoft Windows XP, Microsoft Windows Server 2003

Tabelle 2

Präprozessorvariablen in Visual Studio .NET.

Präprozessorvariable	Beispiel	Ergebnis
var.<Project>.ConfigurationName	$(var.App.ConfigurationName)	Debug|.NET
var.<Project>.ProjectDir	$(var.App.ProjectDir)	D:\Setup\App
var.<Project>.ProjectDosFileName	$(var.App.ProjectDosFileName)	APP.CSP
var.<Project>.ProjectExt	$(var.App.ProjectExt)	.csproj
var.<Project>.ProjectFileName	$(var.App.ProjectFileName)	App.csproj
var.<Project>.ProjectName	$(var.App.ProjectName)	App
var.<Project>.ProjectPath	$(var.App.ProjectPath)	D:\Setup\App\App.csproj
var.<Project>.TargetDir	$(var.App.TargetDir)	D:\Setup\App\Bin\Debug
var.<Project>.TargetDosFileName	$(var.App.TargetDosFileName)	APP.EXE
var.<Project>.TargetExt	$(var.App.TargetExt)	.exe
var.<Project>.TargetFileName	$(var.App.TargetFileName)	App.exe
var.<Project>.TargetName	$(var.App.TargetName)	App
var.<Project>.TargetPath	$(var.App.TargetPath)	D:\Setup\App\Bin\Debug\App.exe
var.SolutionDir	$(var.SolutionDir)	D:\Setup\App\
var.SolutionDosFileName	$(var.SolutionDosFileName)	SETUP.SLN
var.SolutionExt	$(var.SolutionExt)	.sln
var.SolutionFileName	$(var.SolutionFileName)	Setup.sln
var.SolutionName	$(var.SolutionName)	Setup
var.SolutionPath	$(var.SolutionPath)	D:\Setup\App\Setup.sln

und Microsoft Windows Codename „Longhorn“ zu installieren. Allerdings enthält Windows Installer XML keine eigene Logik für diese Tätigkeiten, sondern greift auf die notwendigen Ressourcen des Microsoft Driver Installation Frameworks [5] zurück. Die Aktionen werden durch die Bibliotheken *difxapp.dll* und *difxappa.dll* des Frameworks bereitgestellt, wodurch die Installation der folgenden Geräteklassen ermöglicht wird:

- Dateisystem-Treiber, Dateisystem-Filtertreiber, Dateisystem-Minifiltertreiber,
- klassenspezifische Filtertreiber,
- Plug-and-Play-Treiber,
- Netwerktreiber für die Geräteklassen *Net*, *NetClient*, *NetTrans* und *NetService*,
- Kernel-Modus-Treiber.

Die Integration der Treiberinstallation erfolgt durch spezielle Attribute des Elements <*Component*>. Bei der Verwendung dieser Aktionen muss ein Verweis auf die Objektdatei *difxapp.wixlib* an den Aufruf des Linkers angefügt werden.

Die Integration von benutzerdefinierten Aktionen in die Toolsammlung Windows Installer XML wird permanent weiter entwickelt. So enthält die aktuellste Version eine direkte Implementierung zur Installation und Konfiguration von COM+-Komponenten.

Visual Studio Package (votive)

Windows Installer XML ist nicht nur zur Integration in automatisierte Prozesse geeignet, sondern kann auch als vollwertiges Entwicklungswerkzeug zur Erstellung von Windows-Installer-Dateien verwendet werden.

Um hierbei den Komfort zur Erzeugung der XML-Dokumente und zur Steuerung des Build-Prozesses zu erhöhen, lässt sich Windows Installer XML direkt in die Entwicklungsumgebungen Microsoft Visual Studio .NET 2003 und Microsoft Visual Studio .NET 2005 integrieren.

Nach der Installation des entsprechenden Add-Ins können Sie die WiX-Dokumente direkt in der Entwicklungsumgebung bearbeiten, wobei die Eingabe der erforderlichen Informationen durch IntelliSense vereinfacht wird. Zusätzlich werden – wie es Abbildung 2 zeigt – verschiedene Projekttypen angeboten, bei deren Verwendung die notwendigen Schritte zur Erzeugung des Installationspaketes automatisiert werden.

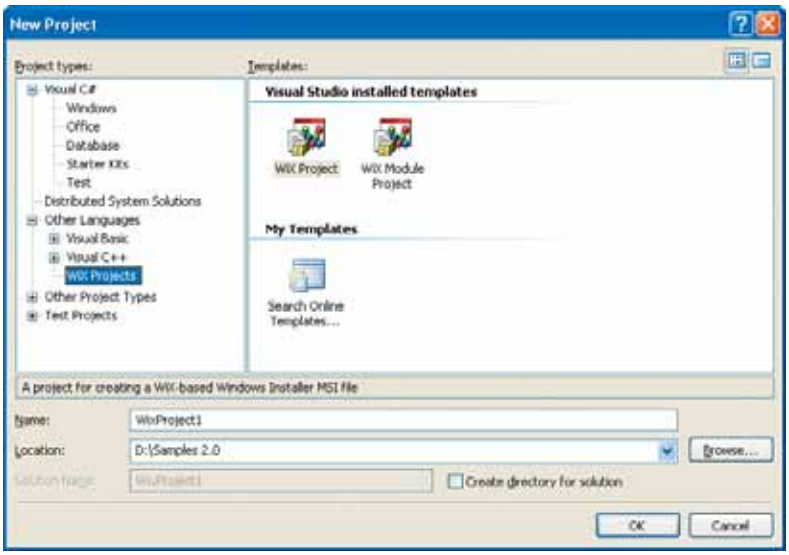

Abbildung 2 Projekttypen zum Erzeugen von Windows-Installer-Dateien.

Bei der Verwendung von Microsoft Visual Studio .NET besteht die Möglichkeit, das WiX-Projekt in eine Projektmappe zu integrieren und auf Elemente der anderen Projekte über spezielle Präprozessorvariablen zuzugreifen. Die Referenzierung ist vergleichbar mit der Verwendung von Variablen, wobei der Name des zu referenzierenden Projektes angegeben werden muss. Soll beispielsweise eine Datei des Projektes *Demo* in ein Installationspaket integriert werden, kann die folgende Definition verwendet werden.

```
<File Id="AppFile" Name="$(var.Demo.
TargetDosFileName)" src="$(var.Demo.
TargetPath)" DiskId="1"/>
```

Tabelle 2 zeigt eine vollständige Auflistung der möglichen Präprozessorvariablen in Visual Studio .NET.

Fazit

Windows Installer XML ist ein mächtiges Werkzeug zum Erzeugen von Installationspaketen, das hinsichtlich der Flexibilität und der Einbindung in bestehende Build-Prozesse seinesgleichen sucht. Die Toolsammlung wurde fast vollständig in der Programmiersprache C# entwickelt und wird als Open-Source-Projekt angeboten. Windows Installer XML und das Visual Studio Package (votive.msi) können von [3] kostenlos geladen werden.

Es ist zu erwarten, dass zukünftig mehrere Installationsprojekte unter Verwendung von Windows Installer XML entwickelt werden und dass dadurch der Markt für Zusatzprodukte langsam größer wird. So bleibt zu hoffen, dass Editoren zur Erzeugung der Benutzeroberfläche im WiX-Format nicht lange auf sich warten lassen oder dass Konverter erscheinen werden, die klassische Ressourcendateien in Windows-Installer-XML-Sprachdateien umwandeln. Sollte dies in naher Zukunft nicht der Fall sein, besteht immer noch die Möglichkeit selbst Hand anzulegen und ein wenig kreative Entwicklungsarbeit zu leisten. |||||||

[1] Andreas Kerl, Windows Installer 3.1, Microsoft Press 2003, ISBN 3-86063-547-6

[2] Andreas Kerl, Inside Windows Installer, Microsoft Press, ISBN 3-86063-099-7

[3] Windows-Installer-XML, sourceforge.net/projects/wix

[4] blogs.msdn.com/robmen

[5] Microsoft Driver Install Frameworks 2.0, www.microsoft.com/whdc/driver/install/difxtools.mspx

Auf einen Blick

Windows Installer XML

Aktuelle Version 2.0.3106.0
Typ Entwicklungstool für Windows-Installer-Setups
Hersteller Microsoft Corporation
Bezugsquelle sourceforge.net/projects/wix/
Lizenz Common Public License
Preis kostenlos