Kapitel 3

Windows Installer und 64-Bit-Betriebssysteme

In diesem Kapitel:

Andreas Kerl, Inside Windows Installer 4.5, © 2009 Microsoft Press Deutschland, ISBN 978-3-86645-431-6

Viele Unwägbarkeiten führten in der Vergangenheit dazu, dass die 64-Bit-Plattform nicht den Stellenwert eingenommen hat, den sie vom technischen und funktionellen Standpunkt aus hätte einnehmen können. Unterschiedliche und inkompatible Prozessorarchitekturen, teure Hardware und fehlende Software waren einige der Gründe dafür. Im Gegensatz dazu war der Markt durchaus vorhanden, denn Software, die eine hohe Rechenleistung fordert oder sehr viel Arbeitsspeicher benötigt, waren und sind absolut keine Seltenheit. Vereinfacht dargestellt bedeutet 64 Bit, dass die Prozessoren durch ihre Bauart so ausgelegt sind, dass 64 Bit (also 8 Byte) während eines Prozessortaktes verarbeitet werden können. Hierdurch sind effektivere Rechenleistungen gerade im Bereich hoher Ganzzahlen möglich, sodass Vorteile bei Verschlüsselungsalgorithmen, bei grafischen Berechnungen und im Multimediaumfeld zu verzeichnen sind. Weitere Vorteile liegen in der Größe des Adressraums, da hierbei nicht mehr die Begrenzung bei 4 Gigabyte liegt, sondern theoretisch bei 2^{64} Byte, also 16 Exabyte, die direkt adressiert werden können. Dieses ist jedoch nur ein theoretischer Wert, denn praktisch ist die Verwendung eines so großen Adressraums noch nicht möglich. Aktuelle Betriebssysteme in 64-Bit-Architektur wie der Windows Server 2008 unterstützen Adressräume bis zu 2 Terabyte. Die Vorzüge sind dennoch offensichtlich und auch die erforderliche Hardware wird nun zu erschwinglichen Preisen angeboten, sodass die breite Nutzung perspektivisch möglich scheint. Die Verfügbarkeit geeigneter Betriebssysteme und Anwendungen sollte die Verbreitung von 64-Bit-Systemen zusätzlich vorantreiben.

Architekturen

Bei der Erstellung eines Installationspaketes ist festzulegen, welche Plattform unterstützt werden soll. Diese Informationen werden in der Eigenschaft *PID_TEMPLATE* des *Summary Information Streams* abgelegt. Es ist denkbar einfach, eine 32-Bit-Plattform als Ziel für ein Installationspaket auswählen. Bei einer 64-Bit-Plattform sieht das Ganze auf den ersten Blick doch etwas verwirrend aus, was ihren Ursprung in dem Namenswirrwarr bei den um 64-Bit-Funktionen erweiterten x86-Prozessoren findet. Es stellt sich zunächst die Frage, was sich hinter Begriffen wie *x86*, *AMD64*, *IA64*, *EM64T* oder *IA64* verbirgt und welche Unterschiede vorhanden sind:

- **x86** Bei *x86* handelt es sich um die klassische Prozessorarchitektur, also 32-Bit-Prozesse die auf einem 32-Bit-Windows-Betriebssystem ausgeführt werden.
- **AMD64** Hierbei handelt es sich um die von *Advanced Micro Devices, Inc.* geschaffene Möglichkeit, 32-Bit-Code auf einem 64-Bit-System nativ auszuführen. Das bedeutet, dass ein 32-Bit-Windows-Betriebssystem auf einer AMD64-Maschine nativ ausgeführt werden kann, also nicht emuliert wird. Technisch gesehen ist der Chip ein vollwertiger 32-Bit-Prozessor, dessen Register im 64-Bit-Modus verbreitert wurden. Er ist dadurch uneingeschränkt zu heutiger 32-Bit- und sogar alter 16-Bit-Software kompatibel. Zusätzlich steht ein 64-Bit-Modus zur Verfügung, der vor allem die Adressierung eines größeren Speicherbereichs ermöglicht und Performanceverbesserungen durch breite Register mit sich bringt. Mit *AMD64* leitete AMD daher einen sanften Übergang von 32- auf 64-Bit-Umgebungen ein. Früher wurde diese Architektur auch als *x86-64* bezeichnet.
- **IA64** *IA64* steht für *Intel Architecture 64-Bit* und ist eine 64-Bit-Architektur und ein Befehlssatz von Intel für die Prozessorgenerationen *Itanium* und *Itanium 2*. Hierbei wird 64-Bit-Code nativ ausgeführt, 32-Bit-Code wird emuliert. Es ist somit nicht möglich, ein 32-Bit-Betriebssystem auf einem *IA64-System* zu installieren, sondern es müssen Betriebssysteme wie Windows *Server 2003 for 64-bit Itanium-based Systems* oder *Windows Server 2008 for Itanium-based Systems* verwendet werden.

Andreas Kerl, Inside Windows Installer 4.5, © 2009 Microsoft Press Deutschland, ISBN 978-3-86645-431-6

- **Intel64** *Intel64* (früher auch *Extended Memory 64 Technology*, abgekürzt *EM64T*), bezeichnet die Erweiterung der Architektur um die Fähigkeit, mehr als 4 Gigabyte Speicher direkt zu adressieren und die 64-Bit-*AMD64-Befehle* auszuführen. Dieser Prozessor unterstützt ebenso die native Ausführung von 32-Bit-Code wie ein *AMD64*.
- **x64** Dieses ist die von Microsoft verwendete Bezeichnung, um Prozessoren zu definieren, die 32- und 64-Bit-Code nativ, also ohne Emulation ausführen können. Somit sind hierunter die beiden Prozessoren *Intel64* (*EM64T*) und *AMD64* zu verstehen.

Nun kommt wieder der Windows Installer ins Spiel und die Verwirrung wird noch größer. Windows Installer 4.5 kennt zur Definition der Zielplattform die Eigenschaften *Intel*, *Intel64*, *AMD64* und *x64*. Der Windows Installer unterstützt seit der Version 2.0 bereits die Installation auf der *IA64-Plattform* und seit der Version 3.0 auf der *AMD64-Plattform*. Damals konnte noch nicht vorhergesagt werden, dass Intel eine weitere 64-Bit-Architektur unterstützen würde, sodass zum damaligen Zeitpunkt passende Eigenschaftsbezeichnungen gewählt wurden, die bis heute aus Gründen der Kompatibilität beibehalten wurden. Um ein Installationspaket für die Ausführung auf einer *IA64-Plattform* zu konfigurieren, ist die Eigenschaft *PID_TEMPLATE* des *Summary Information Streams* auf »Intel64« zu setzen; für die Installation auf der *AMD64-Plattform* hingegen auf »AMD64«. Mit dem Windows Installer 3.1 wurde der zusätzliche Wert *x64* eingeführt. Da Intel jedoch durch die *EM64T-Architektur* kompatibel zur *AMD64-Architektur* ist, handelt es sich hierbei nur um Kosmetik, um hiermit die Unabhängigkeit zum Hersteller der Prozessoren darzustellen. Es ist somit unerheblich, ob der Eigenschaftswert *x64* oder *AMD64* gesetzt wird; das Paket lässt sich auf einer *x64-Plattform* anwenden. Der Eigenschaftswert *AMD64* ist durch den generischen Wert *x64* überflüssig geworden und befindet sich nur noch aus Gründen der Kompatibilität mit älteren Installationspaketen im Eigenschaftsvorrat von Windows Installer.

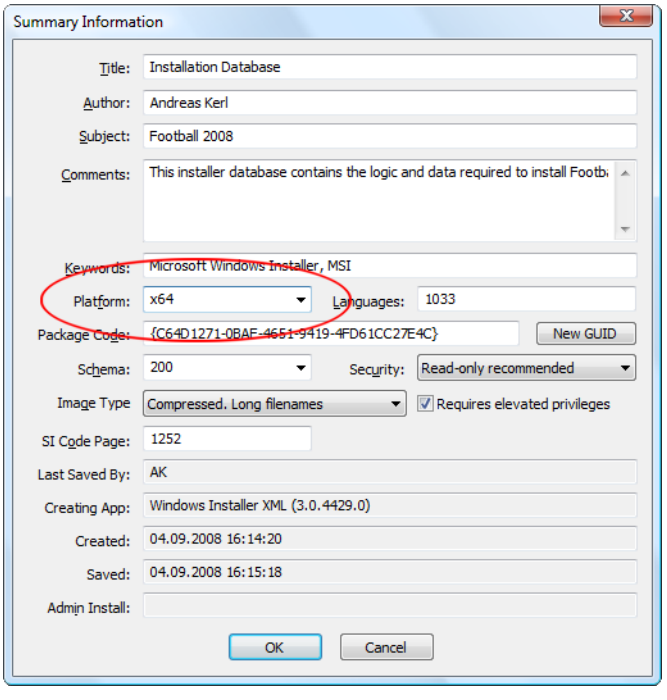

Abbildung 3.1 Plattform-Architektur im »Summary Information Stream«

HINWEIS Auch bei Mergemodulen muss die Eigenschaft *PID_TEMPLATE* des *Summary Information Streams* auf die entsprechende Prozessorarchitektur gesetzt werden. Hierbei ist zu beachten, dass sich Mergemodule, die mit *Intel64* gekennzeichnet sind, auch nur in die identisch gekennzeichneten Installationspakete integrieren lassen. Gleiches gilt für die Kennzeichnung mit *x64* Und *AMD64*. Hingegen lassen sich Mergemodule, die für die 32-Bit-Verwendung gekennzeichnet sind, in jedes Installationspaket integrieren.

Bei der Erstellung eines Paketes mit Windows Installer-XML ist die zu verwendende Architektur dem Attribut *Platform* des Elementes <*Package/*> anzufügen, wie dieses auch in Listing 3.1 dargestellt wird.

```
<?xml version="1.0" encoding="Windows-1252"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
  <!-- Definition des Produktes -->
  <Product UpgradeCode="{DBC3A49F-67B4-4bce-A596-660D365DBFD8}" Manufacturer="Andreas Kerl"
    Language="1033" Version="1.00.0000" Name="Football 2008" Id="{0BC4E7F4-8840-4d23-9AF9-C68CFFDC7C0E}"
    Codepage="1252">

    <!-- Paketdefinition -->
    <Package Keywords="Microsoft Windows Installer, MSI" Description="Football 2008"
      Manufacturer="Andreas Kerl" InstallerVersion="200" Platform="x64" Languages="1033"
      Compressed="yes" SummaryCodepage="1252" InstallPrivileges="elevated" />

    <!-- Weitere Daten -->

  </Product>
</Wix>
```

Listing 3.1 Festlegen der Plattform-Architektur mit Windows Installer-XML

Zur Festlegung der Architektur mit Windows Installer-XML stehen die Werte *Intel*, *Intel64*, *x86*, *x64* und *ia64* zur Verfügung. Beim Kompilieren des WXS-Dokuments wird bei Verwendung von *Intel* und *Intel64* eine Warnung ausgegeben und darauf hingewiesen, dass diese Werte nicht mehr verwendet werden sollten.

```
D:\Draft\Product.wxs(12) : warning CNDL1111 : The value "intel64" for the Package/@Platform attribute
   has been deprecated.  Please use "ia64" instead.
D:\Draft\Product.wxs(12) : warning CNDL1111 : The value "intel" for the Package/@Platform attribute
   has been deprecated.  Please use "x86" instead.
```

Diese Maßnahme trägt ganz entscheidend zur Übersichtlichkeit bei und stellt die Konsistenz zu den regulären Bezeichnungen wieder her, wie auch in der Gegenüberstellung der Bezeichnungen aufgezeigt wird.

Architektur	Windows Installer	Windows Installer-XML
x86	*Intel*	*x86*
AMD64	*AMD64* oder *x64*	*x64*
Intel64	*AMD64* oder *x64*	*x64*
IA64	*Intel64*	*ia64*
x64	*AMD64* oder *x64*	*x64*

Tabelle 3.1 Gegenüberstellung der Kennzeichnungen bei den Plattform-Architekturen

Ein Installationspaket muss als 32-Bit-oder 64-Bit-Paket spezifiziert werden. Die Möglichkeit der Kennzeichnung als neutrales Paket über einen generischen Eigenschaftswert besteht nicht. Es wird somit deutlich, dass es nicht möglich ist, ein Installationspaket für unterschiedliche Zielplattformen zu definieren. Auch die Kombination mehrerer Eigenschaftswerte ist nicht erlaubt.

- Es ist nicht möglich, ein Windows Installer-Paket so zu kennzeichnen, dass die Plattformen *Intel64* und *x64* unterstützt werden. Der Eigenschaftswert »*Intel64*,*x64*« ist ungültig.
- Es ist nicht möglich, ein Windows Installer-Paket so zu kennzeichnen, dass die 32-Bit- und die 64-Bit-Plattform unterstützt werden. Der Eigenschaftswert »Intel,x64« oder »Intel,Intel64« ist ungültig.

Falls versucht wird, ein 64-Bit-Paket auf einer 32-Bit-Plattform zu installieren, wird der Installationsversuch durch den Fehler 1633 (*ERROR_INSTALL_PLATFORM_UNSUPPORTED*) abgebrochen. Auf einer 64-Bit-Plattform wird der Windows Installer in einem 64-Bit-Prozess ausgeführt, sodass sowohl 32-Bit- als auch 64-Bit-Pakete installiert werden können. Hierbei gibt es drei Arten ein Installationspaket zu definieren und zu konstruieren.

- Ein 32-Bit-Paket, das ausschließlich 32-Bit-Komponenten enthält.
- Ein 64-Bit-Paket, das einige 32-Bit-Komponenten enthält.
- Ein 64-Bit-Paket, das ausschließlich 64-Bit-Komponenten enthält.

Hier stellt sich nun die Frage, warum eine plattformspezifische Kennzeichnung überhaupt vorgenommen werden muss. Die Installation eines 32-Bit-Paketes auf einer 64-Bit-Plattform scheint ja auf den ersten Blick zu funktionieren, denn es kommt zu keinen diesbezüglichen Installationsfehlern. Wäre dieses nicht ein geeigneter generischer Ansatz?

Dateisystem und Systemregistrierung

Die Frage des letzten Abschnitts kann nicht pauschal beantwortet werden, denn dies ist in erster Linie von der Anwendung und dem Installationspaket abhängig. Die Installation sollte problemlos erfolgen, allerdings kann es geschehen, dass Komponenten mitunter nicht richtig installiert werden, sodass die Ausführung der Anwendung nicht sichergestellt werden kann. Hiermit ist nicht der tatsächliche Installationsprozess gemeint, sondern der Zugriff auf Ordner und die Systemregistrierung von der Anwendung aus. Hier findet sich auch die gravierende Anforderung zur Verwendung unterschiedlicher Pakete für unterschiedliche Plattformen. Nur durch die geeignete Kennzeichnung wird sichergestellt, dass 64-Bit-Dateien in den richtigen Verzeichnissen abgelegt und der Zugriff auf die Systemregistrierung durch 64-Bit-Komponenten ermöglicht wird. Bei der Installation einer 32-Bit-Anwendung auf einem 64-Bit-Betriebssystem werden die Ordnerzugriffe und Zugriffe auf die Systemregistrierung umgeleitet, sodass von der Anwendung auch auf die zutreffenden Bereiche zugegriffen werden kann.

WOW64-Subsystem

Gerade die letzte Aussage sollte noch ein wenig verdeutlicht werden. Alle 64-Bit-Versionen von Windows enthalten ein Subsystem zur Ausführung von 32-Bit-Anwendungen, das als *WOW64* (Windows-On-Windows 64-Bit) bezeichnet wird. Der hauptsächliche Zweck liegt in der Bereitstellung einer 32-Bit-Umgebung, in der sämtliche Schnittstellen zur Verfügung stehen, die 32-Bit-Windows-Anwendungen benötigen, um ohne Anpassungen auf einem 64-Bit-System zu laufen. Dieses ist der ganz entscheidenden Punkt, denn eine

32-Bit-Anwendung muss nicht explizit verändert werden. Das Betriebssystem stellt automatisch fest, wenn es sich um eine 32-Bit-Anwendung handelt und führt diese dann im Subsystem aus.

Mitunter kann es innerhalb einer Anwendung erforderlich werden, herauszufinden, ob sie als 64-Bit-oder 32-Bit-Prozess ausgeführt wird. Dieses kommt besonders bei .NET-Anwendungen zum Tragen, die nicht für eine spezifische Architektur kompiliert wurden, sondern für die generische Verwendung auf unterschiedlichen Plattformen. Hierbei wird beim Kompilieren lediglich ein Zwischencode (Microsoft Intermediate Language) erzeugt, der im Rahmen JIT-Kompilierung (Just-In-Time) erst auf dem Zielsystem für die jeweilige Architektur angepasst wird. Ob ein Assembly für die plattformunabhängige Verwendung konfiguriert wurde oder ob bereits beim Kompilieren eine spezifische Architektur festgelegt wurde, lässt sich über die Eigenschaft *System.Reflection.Assembly.ProcessorArchitecture* abrufen. Diese Eigenschaft kann die in Tabelle 3.2 aufgeführten Werte zurückliefern.

Wert	Beschreibung
MSIL	Neutrales Format zur plattformunabhängigen Verwendung
X86	Kann auf einer 32-Bit-Plattform oder im *WOW64*-Subsystem einer 64-Bit-Plattform verwendet werden
IA64	Kann auf einer *IA64*-Plattform verwendet werden
Amd64	Kann auf einer *X64*-Plattform verwendet werden

Tabelle 3.2 Kennzeichnung der Architektur bei einem .NET-Assembly

Wird ein plattformunabhängiges Assembly (MSIL) auf einem 64-Bit-System ausgeführt, wird es auch als 64-Bit-Anwendung kompiliert und entsprechend verwendet. Identisches gilt für die Ausführung auf einem 32-Bit-System. Das Betriebssystem stellt die Kernel-Funktion *IsWow64Process()* für solche Prüfmechanismen zur Verfügung. Allerdings existiert die Funktion nur bei 64-Bit-Betriebssystemen, sodass eine plattformunabhängige Prüfmethode nur sehr aufwendig und unter Berücksichtigung von Fehlercodes möglich ist. Einfacher gestaltet sich eine Prüfung, bei der die Größe bestimmter Objekte geprüft wird. Die Struktur *System.IntPtr* ist plattformabhängig und wird zur Repräsentation eines Zeigers oder eines Handles verwendet, sodass sich die Größe der Struktur in Abhängigkeit zur verwendeten Architektur ändert. Innerhalb eines 64-Bit-Prozesses ist die Struktur 8 Byte groß; innerhalb eines 32-Bit-Prozesses hingegen nur 4 Byte. Dieser Umstand lässt sich ausnutzen und auf einfache Weise zur Prüfung der Architektur heranziehen, wie auch Listing 3.2 zeigt.

```
// Gibt True zurück, falls es sich um einen 64-Bit-Prozess handelt
public static bool Is64Bit
{
    get { return (IntPtr.Size == 8); }
}
```

Listing 3.2 Ermittelt die Architektur des Prozesses

Das *WOW64*-Subsystem isoliert 32-Bit-Binärdateien von den 64-Bit-Binärdateien, indem es spezifische Registrierungs- und Dateisystemaufrufe umleitet. Das *WOW64*-Subsystem isoliert diese Dateien, um zu verhindern, dass eine 32-Bit-Binärdatei zufällig auf Daten aus einer 64-Bit-Binärdatei zugreift. Eine 32-Bit-Anwendung, die eine Objektbibliothek aus dem Ordner *%systemroot%\system32* ausführt, könnte beispielsweise zufällig versuchen, auf eine 64-Bit-Objektbibliothek zuzugreifen, die allerdings mit der Anwendung nicht kompatibel ist. Um dies zu verhindern, leitet das Subsystem den Zugriff vom Ordner *%system-*

root%\system32 auf den Ordner *%systemroot%\SysWOW64* um. Hierdurch werden Kompatibilitätsprobleme vermieden, da die Objektbibliothek in diesem Fall speziell für die Verwendung mit 32-Bit-Programmen entwickelt wurde. Was hinsichtlich der Namensgebung auf den ersten Blick merkwürdig aussieht, ist aus Gründen der Kompatibilität erforderlich. Der Ordner *%systemroot%\system32* beinhaltet die 64-Bit-Komponenten und die 32-Bit-Komponenten werden im Ordner *%systemroot%\SysWOW64* abgelegt.

Ein ähnliches Bild ergibt sich auch beim *Global Assembly Cache*, der Assemblys in Abhängigkeit zur Architektur isoliert betrachtet und verwaltet. So existieren Speicherbereiche für Assemblys, die zur Ausführung auf einer 64-Bit-Plattform kompiliert wurden, und auch für solche, die für die 32-Bit-Plattform vorgesehen sind. Zusätzlich existiert ein Speicherbereich, der plattformunabhängige Assemblys aufnimmt. Die Zuordnung, welcher Speicherbereich verwendet wird, erfolgt automatisch und orientiert sich an der bereits beschriebenen Kennzeichnung innerhalb des Assemblys. Die jeweilige Zuordnung wird auch im Windows Explorer dargestellt, wie dieses in Abbildung 3.2 gezeigt wird.

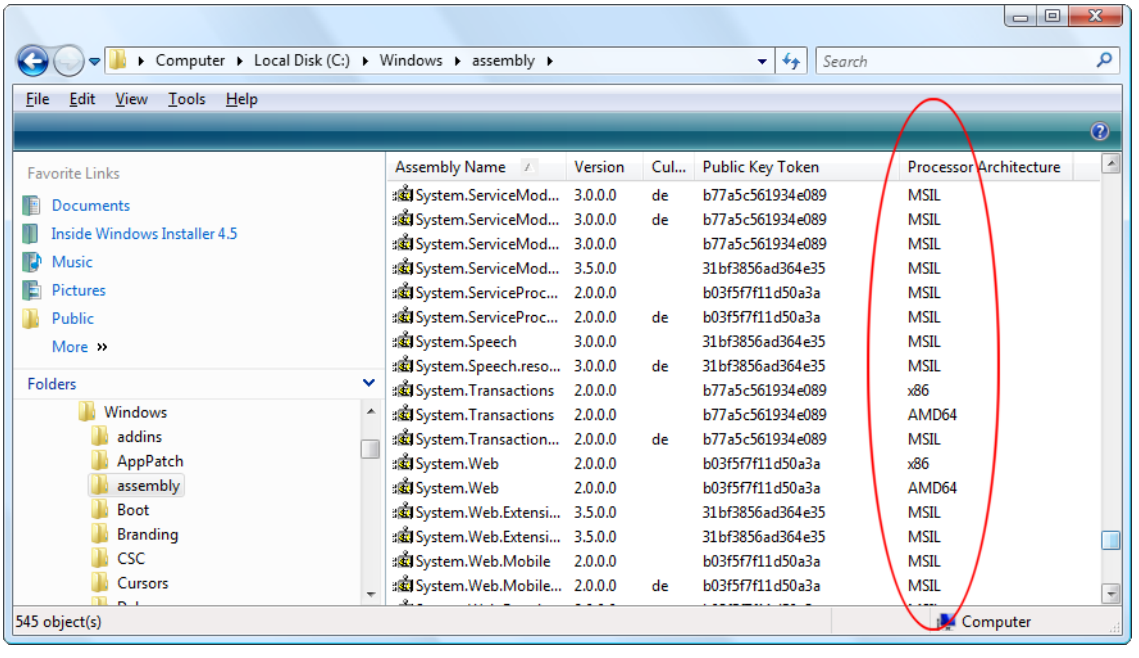

Abbildung 3.2 Kennzeichnung der Architektur im »Global Assembly Cache«

Was für das Dateisystem gilt, gilt auch für die Systemregistrierung. Auch hier wurde das Layout geändert, sodass die Einträge wiederum isoliert voneinander abgelegt werden. Das bedeutet, dass ein 32-Bit-Prozess auf einen anderen Bereich der Systemregistrierung zugreift, als ein 64-Bit-Prozess. Greift ein 64-Bit-Prozess auf den Registrierungsschlüssel *HKEY_LOCAL_MACHINE\SOFTWARE* zu, werden die Informationen auch aus diesem Schlüssel verwendet. Greift hingegen ein 32-Bit-Prozess auf diesen Schlüssel zu, findet eine Umleitung statt und die Informationen von *HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432node* werden verwendet. Dieses funktioniert automatisch und wird als *Registry Redirection* bezeichnet. Diese Unterteilung lässt sich hervorragend mit dem Registrierungs-Editor betrachten. Starten Sie diesen auf einem 64-Bit-System, indem Sie den Befehl »regedit.exe« im Dialogfeld *Ausführen* (Aufruf über ⊞+R) eingeben. Wechseln Sie nun zu *HKEY_LOCAL_MACHINE\SOFTWARE*. Starten Sie anschließend den 32-Bit-Registrierungs-Editor, indem Sie im Dialogfeld *Ausführen* den Befehl »%systemroot%\SysWOW64\regedit.exe –m« eingeben.

Das Argument *–m* ermöglicht das Öffnen mehrerer Instanzen des Editors. Bei der Betrachtung des identischen Schlüssels, sind die Unterschiede offensichtlich, wie auch Abbildung 3.3 zeigt.

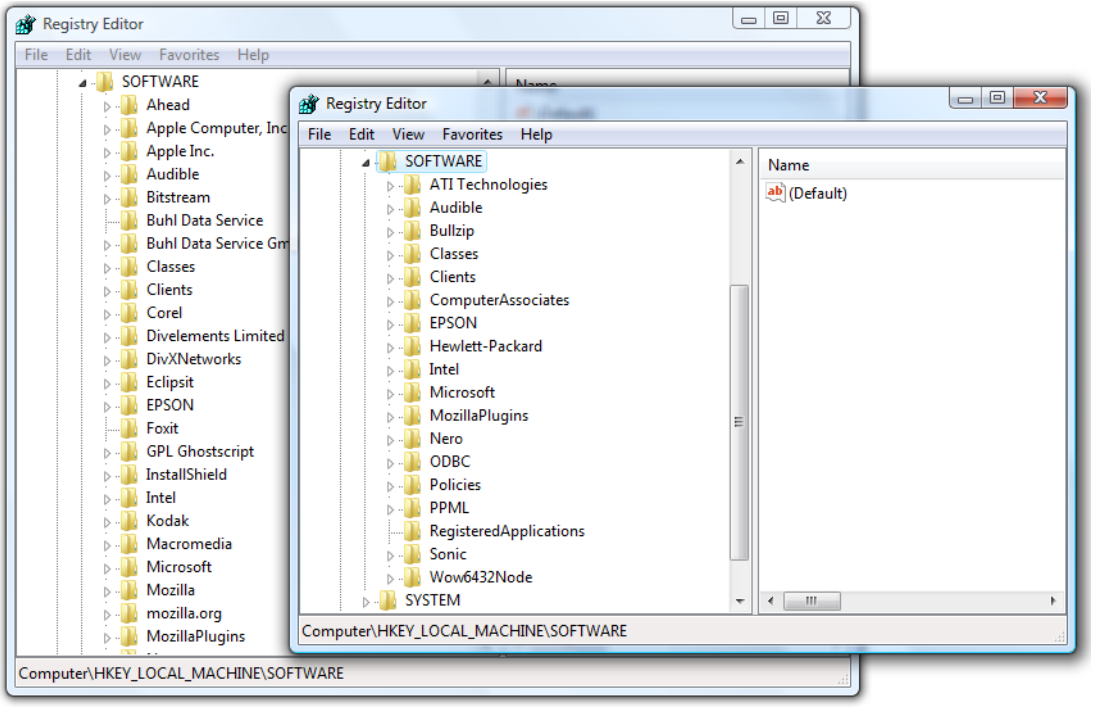

Abbildung 3.3 Unterschiedliche Ansichten mit dem Registrierungs-Editor

Die Systemregistrierung verfügt noch über weitere interessante Erweiterungen, wie die Verwendung gemeinsamer Einträge und die so genannte Spiegelung von Einträgen. Die gemeinsamen Einträge gelten sowohl für den 32-Bit-Bereich, als auch für den 64-Bit-Bereich. Dies bedeutet, dass für diese Einträge keine Umleitung durchgeführt wird. Die gemeinsam verwendeten Einträge sind alle unter *HKEY_LOCAL_MACHINE\SOFTWARE* zu finden. Nachfolgend findet sich ein Auszug aus den Registrierungsschlüsseln, die für beide Sichten gelten:

- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Driver Signing*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSMQ*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Non-Driver Signing*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Software\Microsoft\Shared Tools\MSInfo*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SystemCertificates*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Setup*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Policies*

Die Zielsetzung der Spiegelung (Registry Reflection) ist eine andere und liegt in der Duplizierung eines Eintrages, der in beiden Sichten synchron gehalten wird. Wird also ein Eintrag im 64-Bit-Bereich angelegt, wird automatisch ein Duplikat im 32-Bit-Bereich der Registrierung abgelegt. Dieses Verfahren gilt bei den folgenden Registrierungsschlüsseln:

- *HKEY_LOCAL_MACHINE\SOFTWARE\Classes*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\COM3*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\EventSystem*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Ole*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc*
- *HKEY_USERS\{SID}\Software\Classes*
- *HKEY_USERS\{SID}_Classes*

Hierbei ist aber zu beachten, dass dieses Szenario immer von beiden Seiten betrachtet werden muss. Das bedeutet, dass sich ein Eintrag im 64-Bit-Bereich auf den 32-Bit-Bereich genauso auswirkt wie die umgekehrte Reihenfolge. Also ein Eintrag im 32-Bit-Bereich wird auch im 64-Bit-Bereich vorgenommen. Es gewinnt immer der letzte Eintrag.

Verhalten bei der Installation

Im Rahmen der Entwicklung des Installationspaketes sind ebenfalls einige Faktoren zu beachten, damit die Installation auch wie erwartet durchgeführt wird. Zunächst ist für das gesamte Paket die zu verwendende Plattform-Architektur zu bestimmen. Das bedeutet, dass die Eigenschaft *PID_TEMPLATE* des *Summary Information Streams* auf den Wert »x64« oder »Intel64« festzulegen ist. Die Vorgehensweise dafür wurde bereits erläutert. Durch diese Definition wird der Installationsprozess nicht wesentlich beeinflusst. Allerdings führt diese Änderung bereits dazu, dass die Registrierung des Produktes im 64-Bit-Bereich der Systemregistrierung vorgenommen wird. Weiterhin wird diese Eigenschaft automatisch beim Starten der Installation ausgewertet, um eine Prüfung der Systemarchitektur durchzuführen. Hiermit kann beispielsweise ausgeschlossen werden, dass das Produkt auf einem 32-Bit-System installiert wird.

Darüber hinaus muss die Eigenschaft *PID_PAGECOUNT* des *Summary Information Streams* auf einen Wert von mindestens »200« gesetzt werden. Dieser Wert besagt, dass zur Installation der Windows Installer mindestens in der Version 2.0 vorhanden sein muss. Diese Version ist erforderlich, da vorherige Versionen von Windows Installer keine 64-Bit-Unterstützung enthielten. Dieses waren die Definitionen auf Paketebene, die unbedingt durchzuführen sind. Weitere Festlegungen beziehen sich auf Komponentenebene, wobei der Umfang dieser Einstellungen vom Aufbau der Anwendung und des Gesamtpaketes abhängig sind. Hierbei muss jede 64-Bit-Komponente auch als solche gekennzeichnet werden, damit die Komponentenregistrierung an der richtigen Position ausgeführt wird. Zur Kennzeichnung ist das Attribut *msidbComponentAttributes64bit* der Spalte *Attributes* der Tabelle *Component* anzufügen. Die Definition bei der Verwendung von Windows Installer-XML erfolgt durch das Attribut *Win64* wie das folgende Beispiel zeigt.

```
<Component Id="C__DD.exe" Guid="6ABBEDC4-3D2O-49O1-BBAE-D6D16127571A" Win64="yes">
  <File Id="F__DD.exe" Source=".\Binaries\" Name="DD.exe" KeyPath="yes" DiskId="1" />
</Component>
```

Diese Einstellungen sind für die Zugriffe auf die Systemregistrierung äußerst relevant. Beim Zugriff auf das Dateisystem ist dies nicht der Fall, denn die Definition der Ablagestruktur erfolgt durch die folgenden Eigenschaften:

- *CommonFiles64Folder*
- *CommonFilesFolder*
- *ProgramFiles64Folder*
- *ProgramFilesFolder*
- *System64Folder*
- *SystemFolder*

Bei der Betrachtung der Eigenschaften fällt auf, dass es sich um die 32-Bit- und 64-Bit-Repräsentationen bestimmter Systemordner handelt. Vom logischen Gesichtspunkt her sollte davon ausgegangen werden, dass durch die Eigenschaft *ProgramFiles64Folder* der Ordner *C:\Program Files* und durch *ProgramFilesFolder* der Ordner *C:\Program Files (x86)* definiert werden. Vom Prinzip her ist diese Annahme richtig, dennoch entspricht sie nicht der Realität, denn es muss ebenfalls die Paketdefinition einbezogen werden. Das bedeutet, dass bei einer 64-Bit-Komponente, dessen Installationsverzeichnis auf *ProgramFiles64Folder* festgelegt wurde, eine Umleitung auf *ProgramFilesFolder* stattfindet, falls es sich um ein 32-Bit-Installationspaket handelt. Dies wird bei der Betrachtung des Installationsprotokolls besonders deutlich. Hier ist erkennbar, dass der *ProgramFiles64Folder* zunächst richtig gesetzt, dieser aber anschließend wieder auf das äquivalente 32-Bit-Verzeichnis zurückgesetzt wird.

```
MSI (s) (24:18) [18:20:30:922]: WIN64DUALFOLDERS: 'C:\Program Files (x86)\' will substitute 17 characters
   in 'C:\Program Files\' folder path. (mask argument = 0, the folder pair's iSwapAttrib member = 0).
MSI (s) (24:18) [18:20:30:922]: PROPERTY CHANGE: Modifying ProgramFiles64Folder property. Its current value
   is 'C:\Program Files\'. Its new value: 'C:\Program Files (x86)\'.
```

Dieses Verhalten zeigt die Relevanz, die die Festlegung der jeweiligen Plattform auf Ebene der Paketdefinition einnimmt. Hier ist noch anzumerken, dass dieses Verhalten für alle der oben skizzierten Ordner gilt, wie auch Tabelle 3.3 aufzeigt.

Ordner	32-Bit-Paket auf x86-Plattform	32-Bit-Paket auf x64-Plattform	64-Bit-Paket auf x64-Plattform
CommonFiles64Folder	Null	*C:\Program Files (x86)\Common Files*	*C:\Program Files\Common Files*
CommonFilesFolder	*C:\Program Files\Common Files*	*C:\Program Files (x86)\Common Files*	*C:\Program Files (x86)\Common Files*
ProgramFiles64Folder	Null	*C:\Program Files (x86)*	*C:\Program Files*
ProgramFilesFolder	*C:\Program Files*	*C:\Program Files (x86)*	*C:\Program Files (x86)*
System64Folder	Null	*C:\Windows\SysWOW64*	*C:\Windows\system32*
SystemFolder	*C:\Windows\system32*	*C:\Windows\SysWOW64*	*C:\Windows\SysWOW64*

Tabelle 3.3 Ordnerpfade in Abhängigkeiten zur Zielplattform und zum Installationspaket

Der Zugriff auf die Systemregistrierung gestaltet sich einfacher, da hierbei keine speziellen Variablen zu verwenden sind. Der tatsächliche Ablageort orientiert sich ausschließlich an dem Attribut *msidbComponentAttributes64bit*. Schreibt eine so gekennzeichnete Komponente unter *HKEY_LOCAL_MACHINE\SOFT-*

WARE, werden die Informationen auch an dieser Stelle abgelegt. Handelt es sich um eine Komponente ohne diese Kennzeichnung, also eine 32-Bit-Komponente, wird der Eintrag unter *HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node* abgelegt. Eine 64-Bit-Komponente schreibt somit in den 64-Bit-Bereich und eine 32-Bit-Komponente entsprechend in den 32-Bit-Bereich der Systemregistrierung. Im Installationsprotokoll lassen sich diese Unterscheidungen an den Parametern der Operationsanweisung *RegOpenKey* nachvollziehen. Bei 32-Bit-Komponenten ist dem Parameter *BinaryType* der Wert »0« zugeordnet, bei 64-Bit-Komponenten hingegen der Wert »1«.

```
MSI (s) (F8:C4) [18:52:52:776]: Executing op: RegOpenKey(Root=-2147483646,Key=SOFTWARE\MsiTest,,BinaryType=0,)
MSI (s) (F8:C4) [18:52:52:776]: Executing op: RegAddValue(Name=Typ,Value=32-Bit-Komponente,)
…
MSI (s) (F8:08) [18:52:03:782]: Executing op: RegOpenKey(Root=-2147483646,Key=SOFTWARE\MsiTest,,BinaryType=1,)
MSI (s) (F8:08) [18:52:03:782]: Executing op: RegAddValue(Name=Typ,Value=64-Bit-Komponente,)
```

An einer vorherigen Stelle wurde bereits die Spiegelung von Registrierungsschlüsseln (*Registry Reflection*) erläutert. Hierbei werden Registrierungsschlüssel automatisch dupliziert und synchronisiert. Falls dieses nicht gewünscht ist, kann der Mechanismus außer Kraft gesetzt werden. Hierzu ist die entsprechende Komponente mit dem Attribut *msidbComponentAttributesDisableRegistryReflection* zu versehen. Es gilt zu beachten, dass dieses Attribut erst mit dem Windows Installer 4.0 bereitgestellt wurde und auch nur für Windows Vista und Windows Server 2008 gültig ist. Auf allen anderen Betriebssystemen und auf 32-Bit-Plattformen wird das Attribut ignoriert.

Die Definition mit Windows Installer-XML ist wiederum sehr einfach, wie nachfolgend auch dargestellt wird. Hierbei ist das Attribut *DisableRegistryReflection* des Elements *Component* zu verwenden.

```
<Component Id="C__C1" Guid="26076C98-F5F3-4BD6-9547-F0252D6DB801" Win64="yes" DisableRegistryReflection="no">
  <RegistryValue Id="R__Typ" Root="HKLM" Key="SOFTWARE\Classes\Msi.Demo" Value="Demo"
    Type="string"></RegistryValue>
</Component>
```

Das dargestellte Beispiel konstruiert eine 64-Bit-Komponente, bei der das Attribut *DisableRegistryReflection* nicht gesetzt wurde und somit das Standardverhalten repräsentiert. Dies bedeutet, dass in dem Beispiel die folgenden Einträge in der Systemregistrierung erzeugt und synchron gehalten werden:

- *HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Msi.Demo*
- *HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Classes\Msi.Demo*

Das gleiche Ergebnis wird auch erreicht, wenn es sich um eine 32-Bit-Komponente handelt. Wird nun bei der 64-Bit-Komponente das Attribut *DisableRegistryReflection* aktiviert, wird der folgende Schlüssel erzeugt:

- *HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Msi.Demo*

Wird hingegen bei einer 32-Bit-Komponente das Attribut *DisableRegistryReflection* aktiviert, wird auch nur der folgende 32-Bit-Schlüssel erzeugt:

- *HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Classes\Msi.Demo*

Die bisherigen dargestellten Möglichkeiten bezogen sich auf den schreibenden Zugriff auf die Systemregistrierung, also auf die Modifikation. Soll jedoch im Rahmen der Installation lesend auf die Systemregistrierung zugegriffen werden, steht zu diesem Zweck die Tabelle *RegLocator* zur Verfügung. Die Einträge dieser Tabelle sind mit keiner Komponente verknüpft, sodass über diesen Mechanismus nicht der bitspezifische Teilbereich der Systemregistrierung bestimmt werden kann. Aus diesem Grund muss der Eintrag der Tabelle

RegLocator weiter qualifiziert werden, um somit eine Unterscheidung zwischen dem 32-Bit-Bereich und dem 64-Bit-Bereich zu treffen. Zu diesem Zweck steht das Attribut *msidbLocatorType64bit* zur Verfügung, das mit Windows Installer-XML wie folgt umgesetzt wird:

```
<Property Id="DOTNET64BITENABLED">
  <RegistrySearch Id="RegSearch" Root="HKLM" Key="SOFTWARE\Microsoft\.NETFramework"
    Name="Enable64Bit" Type="raw" Win64="yes"/>
</Property>
```

Listing 3.3 Durchsuchen des 64-Bit-Bereichs der Systemregistrierung

Eigenschaften

Zur Beeinflussung des Installationsprozesses und zur Modellierung des Installationsergebnisses stehen auch Eigenschaften zur Verfügung, die plattformspezifische Informationen enthalten. Somit ist es möglich, diese in Bedingungen zu verwenden und somit Fallunterscheidungen zu treffen. Mitunter kann es erforderlich werden, die Installation einer Komponente von der Plattformarchitektur abhängig zu machen, sodass diese mit einer Bedingung zu versehen ist. Im einfachsten Fall können hierzu die Eigenschaften *VersionNT* oder *VersionNT64* verwendet werden. Beide Eigenschaften werden auf die Versionsnummer des Betriebssystems gesetzt. Die Eigenschaft *VersionNT* ist auf jeder Plattform verfügbar. Die Eigenschaft *VersionNT64* wird hingegen nur auf einer 64-Bit-Plattform gesetzt; auf einer 32-Bit-Plattform ist diese Eigenschaft gar nicht vorhanden.

Ist diese Unterscheidung nicht ausreichend und wird es erforderlich, die tatsächliche Prozessorarchitektur innerhalb der Bedingung zu prüfen, muss auf andere Eigenschaften ausgewichen werden. Bei der Installation werden vom Windows Installer spezifische Eigenschaften gesetzt, die Auskunft über die verwendeten Prozessorarchitekturen geben. Der angefügte Wert wird auf den numerischen Prozessor-Level (*wProcessorLevel*) der Windows-Struktur *SYSTEM_INFO* gesetzt, also »4« für 486, »5« für P5 und »6« für P6. Die folgenden Eigenschaften können hierfür verwendet werden:

- **Intel** Diese Eigenschaft wird gesetzt, wenn die Installation auf einem Intel-Prozessor (32-Bit) ausgeführt wird. Wird die Installation auf einem x64-Prozessor ausgeführt, wird diese Eigenschaft ebenfalls gesetzt, da der 64-Bit-Prozesor die 32-Bit-Befehlssätze nativ unterstützt.
- **Intel64** Diese Eigenschaft wird nur bei einem Itanium-Prozessor, also IA64 gesetzt. Bei einem x64-Prozessor wird die Eigenschaft hingegen nicht gesetzt. Diese Eigenschaft ist verfügbar seit Windows Installer 2.0.
- **MsiAMD64** Wird gesetzt, falls die Installation auf einem x64-Prozessor ausgeführt wird. Diese Eigenschaft ist verfügbar seit Windows Installer 3.0. Sie ist jedoch nur noch aus Gründen der Kompatibilität vorhanden.
- **Msix64** Wird gesetzt, falls die Installation auf einem x64-Prozessor ausgeführt wird. Diese Eigenschaft ist verfügbar seit Windows Installer 3.1.

Entscheidend wird hierbei in den wenigsten Fällen der tatsächliche Eigenschaftswert sein, sondern lediglich die Existenz der Eigenschaft. Wird die Installation beispielsweise auf einem x64-System durchgeführt, sind die Eigenschaften *Intel*, *MsiAMD64* und *Msix64* gesetzt. Die Eigenschaft *Intel64* ist hingegen gar nicht vorhanden.

Allgemeine Hinweise

Darüber hinaus existieren noch allgemeine Hinweise, die sich auf die Verwendung von Komponenten beziehen. Beim Design eines Installationspaketes, das sowohl 32-Bit- als auch 64-Bit-Dateien enthält, sind die Verzeichnispfade so zu wählen, dass es hierdurch zu keiner Beeinträchtigung kommt. Hiermit ist gemeint, dass sich einerseits die Dateien im Rahmen der klassischen Installation nicht überschreiben. Auf der anderen Seite ist auch die Festlegung der Quellverzeichnisse nicht uninteressant, denn im Rahmen einer administrativen Installation werden diese verwendet. Somit ist auch hierbei darauf zu achten, dass sich die Dateien nicht gegenseitig überschreiben. Weiterhin ist zu beachten, dass es sich bei 32-Bit- und 64-Bit-Komponenten aus Sicht von Windows Installer um unterschiedliche Komponenten handelt, sodass sie auch über eine unterschiedliche Komponenten-ID verfügen müssen. Identisches gilt auf Ebene des Produktes; auch hier sind unterschiedliche Produktcodes zu verwenden.

Bei Mergemodulen muss ebenfalls die Eigenschaft *PID_TEMPLATE* des *Summary Information Streams* auf die entsprechende Prozessorarchitektur gesetzt werden. Hierbei ist zu beachten, dass sich Mergemodule, die mit *Intel64* gekennzeichnet sind, auch nur in die identisch gekennzeichneten Installationspakete integrieren lassen. Gleiches gilt für die Kennzeichnung mit *x64*. Mergemodule, die für die 32-Bit-Verwendung gekennzeichnet sind, lassen sich hingegen in jedes Installationspaket integrieren.

Benutzerdefinierte Aktionen

Die bisherigen Ausführungen haben gezeigt, dass es problemlos möglich ist, eine 32-Bit-Datei in ein 64-Bit-Verzeichnis zu kopieren und umgekehrt. Weiterhin stellte es auch kein großes Problem dar, die Zugriffe auf die Systemregistrierung durch die Eigenschaft *msidbComponentAttributes64bit* zu beeinflussen. Diese Vorgehensweisen sind problemlos im Rahmen der Installation durchzuführen, sodass rein formal eine funktionsfähige Anwendung installiert werden könnte. Dieses kann jedoch nicht verallgemeinert werden, denn im Rahmen der Installation fehlt die Betrachtung des tatsächlichen Ausführungsverhaltens der Anwendung. So kann es geschehen, dass die Anwendung fehlerfrei installiert wurde, sich aber in keinem funktionsfähigen Zustand befindet, da das Installationslayout ungeeignet war. Anders verhält es sich bei der Verwendung von benutzerdefinierten Aktionen, denn hierbei tauchen solche Probleme direkt bei der Installation auf. Das bedeutet auch, dass ein nicht ordentlich konzipiertes Installationspaket zwangsläufig zu einem Installationsfehler führen wird. Eine 64-Bit-Datei kann problemlos auf ein 32-Bit-System kopiert werden, aber eine 64-Bit-Datei kann nicht als benutzerdefinierte Aktion bei der Installation auf einem 32-Bit-System verwendet werden.

Benutzerdefinierte Aktionen werden nicht im eigentlichen Installationsprozess ausgeführt, sondern in einem für diese Fälle speziellen Hostprozess, der als *Custom Action-Server* bezeichnet wird. Auf einer 64-Bit-Plattform existieren Custom Action-Server sowohl in 32-Bit als auch in 64-Bit-Ausprägung. Die Zuordnung zum jeweiligen Host-Prozess wird nicht durch die Konfiguration des Paketes vorgenommen, sondern orientiert sich am tatsächlichen Format der hierfür zu verwendenden Datei. Eine 32-Bit-Objektbibliothek wird demzufolge automatisch im 32-Bit Custom Action-Server ausgeführt und eine 64-Bit-Bibliothek im 64-Bit-Server. Diese automatische Zuordnung funktioniert jedoch nur für Dateien, die über ein PE-Header (Portable Executable File Format) verfügen, also auch bei Objektbibliotheken. Mit Hilfe des Windows Task-Mangers kann festgestellt werden, welcher Custom Action-Server hierfür verwendet wird. Wie bereits in Kapitel 1 erläutert wurde, existieren mehrere Prozesse mit der Bezeichnung *msiexec.exe*. Die Hostprozesse zum Ausführen der benutzerdefinierten Aktion verfügen über die identische Bezeichnung. Zur Identifizierung ist der

Befehlszeilenaufruf auszuwerten, denn diesen Prozessen wurde das Argument »-Embedding« übergeben. Auf einem 64-Bit-System werden die 32-Bit-Prozesse durch die Zeichenfolge »*32« gekennzeichnet. Dieses gilt entsprechend auch für die Custom Action-Server, wie die folgende Abbildung 3.4 zeigt.

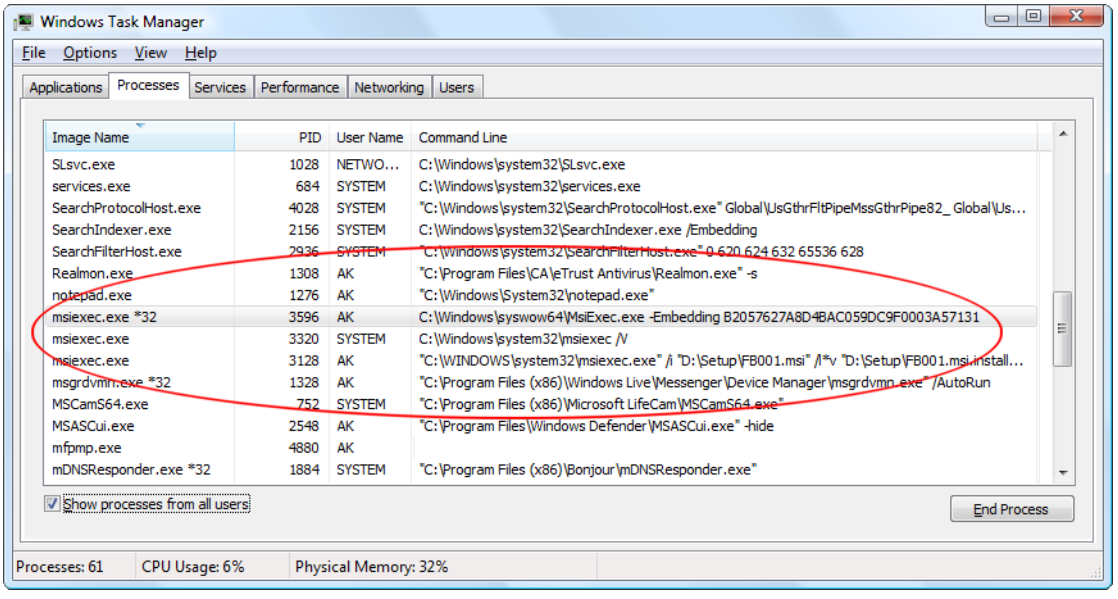

Abbildung 3.4 Kenzeichnung eines 32-Bit Custom Action-Servers im Windows Task-Manger

Das Installationsprotokoll ist an dieser Stelle auch eine sehr große Hilfe, denn hieraus kann direkt abgelesen werden, in welchem Custom Action-Server der Code ausgeführt wird:

```
MSI (s) (74:14) [11:04:43:203]: Executing op: CustomActionSchedule(Action=CA_OutputString32,
   ActionType=1025,Source=BinaryData,Target=OutputString,)
MSI (s) (74:14) [11:04:43:203]: Creating MSIHANDLE (1) of type 790536 for thread 3604
MSI (s) (74:44) [11:04:43:203]: Invoking remote custom action. DLL: C:\WINDOWS\Installer\MSI243.tmp,
  Entrypoint: OutputString
MSI (s) (74:2C) [11:04:43:203]: Generating random cookie.
MSI (s) (74:2C) [11:04:43:219]: Created Custom Action Server with PID 2260 (0x8D4).
MSI (s) (74:64) [11:04:43:250]: Running as a service.
MSI (s) (74:64) [11:04:43:250]: Hello, I'm your 32bit Impersonated custom action server.
…
MSI (s) (74:14) [11:04:43:250]: Executing op: CustomActionSchedule(Action=CA_OutputString64,
  ActionType=1025,Source=BinaryData,Target=OutputString,)
MSI (s) (74:14) [11:04:43:250]: Creating MSIHANDLE (3) of type 790536 for thread 3604
MSI (s) (74:8C) [11:04:43:266]: Invoking remote custom action. DLL: C:\WINDOWS\Installer\MSI244.tmp,
  Entrypoint: OutputString
MSI (s) (74:2C) [11:04:43:266]: Generating random cookie.
MSI (s) (74:2C) [11:04:43:266]: Created Custom Action Server with PID 3480 (0xD98).
MSI (s) (74:08) [11:04:43:282]: Running as a service.
MSI (s) (74:64) [11:04:43:282]: Hello, I'm your 64bit Impersonated custom action server.
```

Listing 3.4 Kennzeichnung der Custom Action-Server im Installationsprotokoll

Wie bereits weiter oben erläutert, funktioniert die automatische Zuordnung des Custom Action-Servers nur für Dateien, die über einen PE-Header verfügen. Bei Skriptdateien oder der Verwendung von Skriptcode

funktioniert diese Automatik somit nicht, sodass eine manuelle Zuordnung erfolgen muss. Hierfür steht das Attribut *msidbCustomActionType64BitScript* zur Verfügung, das der Spalte *Type* der Tabelle *CustomAction* anzufügen ist. Bei der Verwendung von Windows Installer-XML erfolgt die Zuordnung über das Attribut *Win64* des Elementes *<CustomAction/>*, wie auch Listing 3.5 zeigt. Wie bei den Objektdateien, wird die Verwendung des jeweiligen Custom Action-Servers dem Installationsprotokoll ebenfalls angefügt.

```
<!--Visual Basic-Skript verwenden-->
<Binary Id="VBScript" SourceFile=".\Binaries\GetInfos.vbs" />

<!--Definition der benutzerdefinierten Aktion-->
<CustomAction Id="CheckEnvironment" BinaryKey="VBScript" VBScriptCall="Main"
  Win64="yes" Execute="immediate"/>

<!--Einordnung in die Ausführungssequenz-->
<InstallExecuteSequence>
  <Custom Action="CheckEnvironment" Before="LaunchConditions"/>
</InstallExecuteSequence>
```

Listing 3.5 Definition einer benutzerdefinierten Aktion vom Typ »Skript«

Ausführbare Dateien, die als benutzerdefinierte Aktionen verwendet werden, benötigen keine besondere Kennzeichnung, denn sie werden im eigenen Prozessraum ausgeführt und sind somit auf keinen Hostprozess angewiesen.

Richtlinien

Es ist erkennbar, dass die Erstellung und Konfiguration von Installationspaketen für unterschiedliche Plattformen und Prozessorarchitekturen als nicht trivial zu bezeichnen ist. Aus diesem Grund sollten die folgenden Dinge bei der Erstellung eines 64-Bit-Paketes beachtet werden:

- Verwenden Sie das Windows Installer-Datenbankschema »200« oder höher. Legen Sie die Version 2.0 als minimale Windows Installer-Version fest, die benötigt wird, um das Paket zu installieren. Hierzu ist die Eigenschaft *PID_PAGECOUNT* im *Summary Information Stream* auf den Wert »200« zu setzen. Frühere Versionen von Windows Installer lassen die Installation von 64-Bit-Paketen nicht zu.
- Legen Sie die erforderliche Prozessorarchitektur für dieses Paket fest. Verwenden Sie den Wert »Intel64« in der Eigenschaft *PID_TEMPLATE* des *Summary Information Streams*, falls das Paket auf einer *IA64-Plattform* ausgeführt werden soll. Verwenden Sie hingegen den Eigenschaftswert »x64«, um die Ausführung auf einem x64 System, also *EM64T* und *AMD64* zu ermöglichen. Ein Paket kann nicht mehrere Plattformen unterstützen. Daher sind die Eigenschaftswerte »Intel64,x64«, »Intel,x64« oder »Intel,Intel64« ungültig.
- Identifizieren Sie jede 64-Bit-Komponente, indem Sie in der Spalte *Attributes* der Tabelle *Component* das Attribut *msidbComponentAttributes64bit* verwenden.
- Verwenden Sie Bedingungen, um die Version des 64-Bit-Betriebssystems zu prüfen, falls dieses erforderlich ist. Hierzu ist die Eigenschaft *VersionNT64* zu verwenden. Diese Eigenschaft wird nur unter 64 Bit gesetzt; bei 32 Bit befindet sich diese Eigenschaft in einem nicht definierten Zustand.
- Verwenden Sie Bedingungen, um die Prozessorarchitektur und den numerischen Prozessor-Level zu bestimmen, falls dieses erforderlich sein sollte. Verwenden Sie hierzu die Eigenschaften *Intel*, *Intel64* oder *Msix64*.

- Verwenden Sie die Tabelle *AppSearch* und die gleichnamige Aktion, um in der Systemregistrierung nach existierenden 64-Bit-Elementen zu suchen, falls dies erforderlich sein sollte. Ergänzen Sie hierzu in der Tabelle *RegLocator* den Inhalt der Spalte *Type* um das Attribut *msidbLocatorType64bit*.
- Verwenden Sie zum Bestimmen der Pfade zu den 64-Bit-Systemordnern die Eigenschaften *System64Folder*, *ProgramFiles64Folder* und *CommonFiles64Folder*. Für die 32-Bit-Systemordner sind die Eigenschaften *SystemFolder*, *ProgramFilesFolder* und *CommonFilesFolder* zu verwenden.
- Stellen Sie sicher, dass die Anwendung die korrekte *GUID* verwendet, falls Sie 64-Bit-Komponenten über den Mechanismus der qualifizierten Komponenten referenzieren. Wenn eine Komponente sowohl in einer 32-Bit- und einer 64-Bit-Variante vorliegt, verfügen diese über unterschiedliche IDs.
- Stellen Sie sicher, dass die Komponente mit »ODBCDriverManager64« bezeichnet wird, falls Sie einen *64-Bit ODBC Driver Manager* installieren möchten. Der ODBC Driver Manager ist dieser Komponente hinzuzufügen. Er wird installiert, falls dies erforderlich sein sollte.
- Stellen Sie sicher, dass die Installation nur 32-Bit-Betriebssystemdienste verwendet, die als ausführbare Datei vorliegen. Ein 64-Bit-Paket kann keine 32-Bit-Dienste aufrufen, die als Objektbibliothek vorliegen.
- Stellen Sie sicher, dass die Einträge in INI-Dateien korrekt zugeordnet werden, falls eine Anwendung koexistierende 32-Bit- und 64-Bit-Varianten einer Komponente installiert.
- Stellen Sie sicher, dass 32-Bit-Patches nur auf 32-Bit-Dateien und 64-Bit-Patches nur auf 64-Bit-Dateien angewendet werden.
- 32-Bit-Anwendungen, die im *WOW64*-Subsystem ausgeführt werden, benutzen eine andere Sicht auf die Systemregistrierung als 64-Bit-Anwendungen. Durch die Spiegelung der Registrierung wird erreicht, dass Einträge der Systemregistrierung im 32-Bit-Bereich und 64-Bit-Bereich synchron gehalten werden. Um diese Funktionalität zu deaktivieren, ist für die entsprechende Komponente in der Tabelle *Component* der Spalte *Attribut* das Attribut *msidbComponentAttributesDisableRegistryReflection* anzufügen. Dieses Attribut steht nur mit dem Windows Installer 4.0 und höher zur Verfügung und wird somit unter Verwendung der 64-Bit-Versionen von Windows 2000 und Windows XP genauso ignoriert wie bei einem 32-Bit-Betriebssystem.

Nachdem das Installationspaket den Vorgaben entsprechend erstellt wurde, sollte einer fehlerfreien Ausführung nichts mehr im Wege stehen. Falls es dennoch zu Fehlern kommen sollte, geben die Fehlernummern darüber Auskunft, ob es sich um eine 64-Bit-Problematik handelt:

- **1633** Das Installationspaket ist für diese Plattform nicht geeignet. Hier wird versucht, ein 64-Bit-Paket auf einer 32-Bit-Plattform oder ein 64-Bit-Paket auf einer inkompatiblen 64-Bit-Plattform zu installieren.
- **2401** Es wird versucht, eine 64-Bit-Operation in der Systemregistrierung eines 32-Bit-Betriebssystems durchzuführen.
- **2943** Es wird versucht, das Advertise-Skript eines 64-Bit-Paketes auf einem inkompatiblen Betriebssystem auszuführen.

Um im Vorfeld alle Faktoren zu berücksichtigen, die bei der Entwicklung von 64-Bit-Paketen zu beachten sind, ist bei einer Validierung die Überprüfungsart *ICE80* besonders zu beachten.

Fazit

Auf den aktuellen 64-Bit-Plattformen können sowohl 32-Bit- als auch 64-Bit-Anwendungen ausgeführt werden. Die Installation einer 32-Bit-Anwendung und die Entwicklung eines geeigneten Installationspaketes sind nahezu identisch mit den Vorgaben, die bei Installationen für die 32-Bit-Plattform gelten. Das bedeutet aber auch, dass einige abweichende Verhaltensmuster existieren, die besonders zu berücksichtigen sind. Diese Verhaltensmuster sind darauf zurückzuführen, dass 32-Bit- und 64-Bit-Anwendungen voneinander isoliert abgelegt werden und somit die Anwendung eine andere Sicht auf das System erhält. Die Erstellung eines Installationspaketes zur Installation einer 64-Bit-Anwendung weicht nicht wesentlich von den bekannten Vorgehensweisen ab. Wie auch bei der Anwendungsentwicklung, sind bei der Erstellung solcher Pakete bestimmte Regeln zu beachten und einige Vorüberlegungen anzustellen. Nur wenn die unterschiedlichen Plattformeigenheiten auch Beachtung finden, kann die Funktionsfähigkeit der zu installierenden Anwendung sichergestellt werden.

Andreas Kerl, Inside Windows Installer 4.5, © 2009 Microsoft Press Deutschland, ISBN 978-3-86645-431-6

Andreas Kerl, Inside Windows Installer 4.5, © 2009 Microsoft Press Deutschland, ISBN 978-3-86645-431-6