Windows Installer 3.1

Architektur • Troubleshooting • Patchmanagement

Sprache: **deutsch**
Preis: **39,90 EUR**
Autor: **Andreas Kerl**
ISBN: **3-86063-547-6**
Ausstattung: **Hardcover**
Produktbereich: **Diverse**
Umfang: **448 Seiten**
Lieferbar: **ab 07/2005**

Kurzbeschreibung: Der Windows Installer ist ein in das Betriebssystem integrierter Dienst zum Installieren und zum Verwalten von Anwendungen. Diese Erläuterung zu dem Windows Installer ist sehr einfach gehalten, sie stellt jedoch die wesentlichen Aspekte der Technologie in den Vordergrund. Die Installation stellt den ersten Schritt dar, mit dem Ziel, die Anwendung in einen lauffähigen Zustand zu versetzen. Die Verwaltung der Anwendung stellt sicher, dass diese in einem lauffähigen Zustand verbleibt.

In diesem Buch werden die gerade erläuterten Aspekte im Detail betrachtet, wobei sich ein Großteil der Betrachtungen auf die neuen Funktionalitäten des Windows Installers 3.0 und 3.1 bezieht. Zunächst werden jedoch die internen Abläufe im Installationsprozess detailliert aufgezeigt, wodurch die notwendigen Grundlagen vermittelt werden, um effizient auf Problemsituationen reagieren zu können. Hierbei beginnt dieses Buch wo andere Bücher aufhören, denn Begriffe wie »Darwin Descriptoren«, »SQUID«, »Custom Action Server« und »Installationsskripte« bleiben hiervon nicht ausgenommen.

Der Zweite Teil dieses Buches befasst sich mit der Erweiterung des Windows Installer-Servicemodells durch die neuen Funktionalitäten des Windows Installer 3.0 und 3.1. Auch hierbei werden auf einer sehr tiefen technischen Ebene die internen Abläufe bei der Anwendung von Patches betrachtet, wodurch alle Voraussetzungen für den effektiven Einsatz von Windows Installer-Patches geschaffen werden.

Abgerundet wird der Umfang dieses Buches durch die Betrachtung verwandter Technologien wie das *Microsoft Driver Installations Framework (DifX)* oder die Toolsammlung *Windows Installer-XML (WiX)*.

KAPITEL 1: DER WINDOWS INSTALLER

- Windows Installer-Technologie
  - Überblick
  - Dateiformate
  - Verteilbare Dateien
- Neue Funktionen und Implementierungen in den Versionen 3.0 und 3.1
  - Servicemodell
  - Befehlszeilenoptionen
  - Troubleshooting
  - Externe Benutzeroberfläche
  - Änderung des Benutzer-SID
  - Validierung
  - Weiteres
- Tools und Anwendungen
  - Windows Installer-SDK
  - Windows Installer Tabellen Editor (Orca)
  - Windows Installer-XML (WiX)

KAPITEL 2: ANATOMIE DES WINDOWS INSTALLER-PAKETES

- Interne Betrachtung
  - Summary Information Stream

- Windows Installer-Datenbank
- Installierbare Ressourcen

- Logische Betrachtung
  - Ressourcen
  - Komponenten
  - Features
  - Produkt
- Einstellungen für den Installationsprozess
  - Eigenschaften
  - Bedingungen
- Paketerstellung mit Windows Installer-XML

KAPITEL 3: DER INSTALLATIONSPROZESS

- Installationsarten
  - Clientinstallation
  - Administrative Installation
  - Angekündigte Installation
- Installationsphasen
  - Acquisition-Phase
  - Execution-Phase
  - Vorteile der Unterteilung
- Prozesse
  - Client-Prozess
  - Server-Prozess
- Skriptdateien
  - Metadaten für die Skriptausführung
  - Aktualisierung der Systemregistrierung
  - Registrierung der Installationskomponenten
  - Aktualisierung von Dateien
  - Ausführung von benutzerdefinierten Code
- Konfigurationsdaten
  - Darstellungsformen einer GUID
  - Darwin Descriptor
  - Undokumentierte Verwendungsmöglichkeiten
- Analyse
  - Windows Installer-Protokollierung
  - Analyse der Skriptdateien

KAPITEL 4: BENUTZERDEFINIERTE AKTIONEN

- Custom Action-Server
  - Architektur
  - Erzeugen des Custom Action-Servers
  - Verwenden einer Objektbibliothek
  - Verwenden eines Skriptes
  - Spezielle Anwendungsfälle
- Grundlegende Betrachtungen
  - Einordnen in den Installationsprozess
  - Rückgabewerte
  - Bedingungen und Deinstallation
  - Debuggen von benutzerdefinierten Aktionen
- Definieren von benutzerdefinierten Aktionen

- Anatomie eines Patches
  - Summary Information Stream
  - Eingebettete Transformationen
  - Kabinettdateien
  - Datenbanktabellen
- Anwendungsreihenfolge von Patches
  - Theoretische Aspekte
  - Patchfamilie
  - Ausführungssequenzen
  - Gültigkeitsdauer von Patches
- Komplexe Aktualisierungsszenarien
  - Zurückbeordern von existierenden Patches
  - Sequenzen in »Multi-Target-Patches«
  - Verwenden mehrerer Patchfamilien
  - Bedingungen in Sequenzen
  - Änderung des Modells der Patchfamilien
- Migration und Kompatibilität
  - Integration mit »älteren« Patches
  - Installation von Patches der Version 3.0 unter älteren Versionen des Windows Installers

KAPITEL 8: ERSTELLEN VON WINDOWS INSTALLER-PATCHES

- Vorüberlegungen
  - Problemquellen bei Patches der Version 2.0
  - Konfliktbeseitigung durch Verwenden des »Baseline-Cache«
- Struktur des Patch Creation Property File
  - Tabelle »Properties«
  - Tabelle »ImageFamilies«
  - Tabelle »UpgradedImages«
  - Tabelle »TargetImages«
  - Tabelle »UpgradedFiles_OptionalData«
  - Tabelle »FamilyFileRanges«
  - Tabelle »TargetFiles_OptionalData«
  - Tabelle »ExternalFiles«
  - Tabelle »UpgradedFilesToIgnore«
  - Tabelle »PatchSequence«
  - Tabelle »PatchMetadata«
- Vorgehensweise beim Erstellen eines Patches
  - Festlegen der erforderlichen Informationen
  - Erstellen des Patchpaketes
  - Problematische Szenarien
- Besondere Anwendungsfälle
  - Automatische Generierung von Sequenzinformationen
  - Produktlinien und Produktversionen
  - Patchen von bestimmten Regionen einer Datei
  - Verwenden von Symboldateien
  - Cachen von Informationen

KAPITEL 9: ANWENDUNGSPROZESS BEI WINDOWS INSTALLER-PATCHES

- Anwenden von Patches
  - Dynamische Anwendung

3 Der Installationsprozess

Der Windows Installer-Service ist der Kern der gesamten Windows Installer-Technologie. Der Windows Installer-Service ist der Baustein, der die eigentlichen Installationsaufgaben ausführt. Hierzu verwendet er die Informationen des Windows Installer-Paketes. Zusätzlich findet im Rahmen des Installationsprozesses eine intensive Kommunikation mit dem Benutzer statt, um weitere Daten dynamisch zu ermitteln. Dieses Kapitel stellt den internen Ablauf des Installationsprozesses dar und zeigt welche Informationen der Windows Installer zur Konfiguration des Produktes auf dem System ablegt. Weiterhin enthält dieses Kapitel detaillierte Informationen zur Analyse des Installationsprozesses und zeigt Möglichkeiten auf, eine effektive Fehlersuche zu realisieren.

Installationsarten

Beim Windows Installer-Service handelt es sich um einen Betriebssystemdienst, der für die Installation, Deinstallation und Modifikation von Windows Installer-Paketen verantwortlich ist. Darüber hinaus wird der Windows Installer-Service ebenfalls verwendet, um Windows Installer-Patches zu installieren und zu deinstallieren. Dieser Betriebssystemdienst trägt die Bezeichnung »MSIServer« und verwendet die ausführbare Datei *msiexec.exe*. Der Dienst wird nicht automatisch vom System gestartet, sondern er muss manuell durch einen Aktivierungsmechanismus aufgerufen werden. Diese Aktivierung kann auf unterschiedliche Arten erfolgen:

- Durch die Verwendung der Befehlszeilenoptionen des Windows Installer-Dienstes.
- Durch das Doppelklicken auf eine Datei mit der Endung `.msi` oder `.msp`, da diese Dateinamenerweiterungen mit dem Windows Installer-Service verknüpft sind.
- Durch Funktionen des Windows Installer-API oder des Windows Installer-Automationsmodells.
- Durch Softwareverteilungstechnologien wie *Microsoft Systems Management Server (SMS)* oder die auf Gruppenrichtlinien basierenden Technologien, die in *Microsoft Windows 2000 Server* und *Microsoft Windows Server 2003* integriert sind (Active Directory).
- Durch die Option *Software* der Systemsteuerung von *Microsoft Windows 2000*, *Microsoft Windows XP* und *Microsoft Windows Server 2003*.
- Durch Anwendungen, die Windows Installer-Technologien direkt nutzen. Hierunter ist z.B. *Microsoft Word 2003* anzusiedeln, da über den Menüpunkt *Erkennen und Reparieren* direkt auf den Windows Installer-Service zugegriffen wird.

Unabhängig davon, auf welche Art der Windows Installer-Service aufgerufen und somit ein Installationsprozess gestartet wird, wird bei dem Aufruf immer eine Referenz auf das jeweilige Installationspaket übergeben. Die ersten Schritte im Installationsprozess erstrecken sich auf die Auswertung der Informationen des Summary Information Streams. So wird an dieser Stelle zunächst geprüft, ob das Installationspaket auf dem aktuellen System überhaupt verwendet werden kann, indem die definierte Version des Windows Installers und die jeweilige Plattform auf Übereinstimmung geprüft werden. Im Anschluss wird das Installationspaket in den Arbeitsspeicher geladen, so dass direkt auf die notwendigen Informationen der Windows Installer-Datenbank zugegriffen werden kann. Nachfolgend werden nun die Aufrufparameter ausgewertet und darüber hinaus geprüft, ob sich das Produkt bereits auf dem System befindet. Anhand dieser Informationen wird die tatsächliche Installationsart festgelegt. In Abhängigkeit zu der ermittelten Installationsart werden Patches und Transformationen auf das Paket angewendet, weitere Systemkonfigurationen ausgewertet und schließlich die in der Windows Installer-Datenbank definierten Installationsabfolgen ausgeführt.

Dieser sehr schematische Installationsablauf wird an späterer Stelle in diesem Kapitel weiter präzisiert. Allerdings sind diese Informationen für den Moment ausreichend um die Aufrufmöglichkeiten der einzelnen Installationsarten zu erläutern.

Clientinstallation

Die Clientinstallation ist die am häufigsten verwendete und auch die relevanteste Installationsart. Es handelt sich hierbei um den bekannten Prozess, in dem Ressourcen vom Quellmedium auf den Zielcomputer übertragen werden, um dort eine funktionsfähige Anwendung zu erstellen. Eine spezielle Form der Clientinstallation ist der Wartungsmodus. In diesen Wartungsmodus wird automatisch verzweigt, falls eine Clientinstallation gestartet wird und das jeweilige Produkt auf dem System bereits vorhanden ist.

Der Wartungsmodus wird ebenfalls verwendet, falls ein Minor- oder Small-Update auf das Produkt angewendet wird, eine Reparatur des Produktes veranlasst wird oder das Produkt deinstalliert wird. Im Installationsprotokoll lässt sich eine Clientinstallation am einfachsten erkennen, indem die Eigenschaft ACTION ausgewertet wird. Bei einer Clientinstallation muss für diese Eigenschaft der Wert »INSTALL« festgelegt sein.

```
Property(S): ACTION = INSTALL
```

Wird eine Clientinstallation von einem administrativen Installationspunkt ausgeführt (Post-Admin-Install), so wird während der Installation die Eigenschaft IsAdminPackage auf den Wert »1« festgelegt. Diese Information kann ebenfalls der Eigenschaftsauflistung des Installationsprotokolls entnommen werden.

Basisinstallation

Eine Clientinstallation wird immer über das Argument /i gesteuert, wobei das Zeichen »i« für »Install« steht. Zusätzlich ist eine Referenz auf das zu verwendende Installationspaket erforderlich. Bei einer Basisinstallation ist hierzu der vollständige Pfad zum Installationspaket nach dem folgenden Schema zu verwenden. Diese Referenz kann auch als *UNC-Name (Universal Naming Convention)* oder als *URL (Uniform Resource Locator)* übergeben werden.

- `msiexec /i <Pfad zum Installationspaket>`

Seit dem Windows Installer 3.0 stehen darüber hinaus Standardbefehlszeilenoptionen zur Verfügung. Die Installation eines Produktes unter Verwendung dieser Optionen kann nach dem folgenden Schema erfolgen:

- `msiexec /package <Pfad zum Installationspaket>`

Die Clientinstallation kann ebenfalls über die Funktion MsiInstallProduct() des Windows Installer-API oder über die Methode Installer.InstallProduct des Windows Installer- Automationsmodells gestartet werden.

Im Installationsprotokoll ist die Durchführung einer Basisinstallation anhand der nachfolgenden Eintragungen zu erkennen:

```
MSI (s) (E8:B4) [14:15:32:375]: Product not registered: beginning first-time install
MSI (s) (E8:B4) [14:15:32:375]: PROPERTY CHANGE: Adding ProductState property. Its value is '-1'.
```


Wartungsmodus

In den Wartungsmodus wird automatisch verzweigt, falls die Installation nach dem vorherigen Schema gestartet wird und das Produkt sich bereits auf dem System befindet. Optional kann hierbei anstelle des Pfades zum Installationspaket auch der ProductCode des jeweiligen Produktes angegeben werden. Darüber hinaus ist es auch möglich, eine Reparatur oder eine Reinstallation des Produktes über einen Befehlszeilenaufruf zu starten, der letztlich wieder im Wartungsmodus mündet. Der Reparaturmodus wird hierbei durch das Argument /f eingeleitet, wobei das Zeichen »f« für »Fix« steht, und um die durchzuführenden Reparaturoptionen ergänzt wird.

- `msiexec /f[Reparatur Optionen] <Pfad zum Paket oder ProductCode>`

Die Festlegung der Reparaturoptionen wird durch eine Verkettung der nachfolgend dargestellten Zeichen ermöglicht.

Argument	Beschreibung
p	Ausschließliche Reinstallation der fehlenden Dateien.
o	Reinstallation wenn eine Datei fehlt oder in einer älteren Version vorliegt.
e	Reinstallation von fehlenden Dateien oder von Dateien, deren Versionsnummer kleiner oder gleich ist.
d	Reinstallation von fehlenden Dateien oder von Dateien mit abweichenden Versionsnummern.
c	Reinstallation von fehlenden Dateien oder von Dateien, bei denen die gespeicherte *Checksumme* nicht mit dem berechneten Wert übereinstimmt. Dies gilt nur für Dateien, bei denen das Attribut msidbFileAttributesChecksum in der Tabelle File gesetzt worden ist.
a	Reinstallation aller Dateien ohne Beachtung der Versionen und *Checksummen*.
u	Wiederherstellung aller anwenderspezifischen Registrierungseinträge unter HKEY_CURRENT_USER und HKEY_USERS.
m	Wiederherstellung aller computerspezifischen Registrierungseinträge unter HKEY_LOCAL_MACHINE und HKEY_CLASSES_ROOT. Schreiben aller Informationen der Tabellen Class, Verb, PublishComponents, ProgID, MIME, Icon, Extension und AppID. Reinstallation aller qualifizierten Komponenten.
s	Reinstallation aller Verknüpfungen im Startmenü.
v	Erzwingt das erneute Ausführen vom Originalmedium und aktualisiert das im Cache vorhandene Windows Installer-Paket.

Tabelle 3.1: *Befehlszeilenargumente zur Reparatur einer Installation*

Die Reparatur eines Produktes kann darüber hinaus durch die Funktion `MsiReinstallProduct ()` des Windows Installer-API oder die Methode `Installer.ReinstallProduct` des Windows Installer-Automationsmodells realisiert werden. Weiterhin ist es möglich, durch Optionen innerhalb des Dialogs *Software* der Systemsteuerung, die Reparatur eines installierten Produktes durchzuführen. In diesem Fall wird die Reparatur unter Verwendung der Optionen »ocmusv« ausgeführt.

HINWEIS: Die Reparaturmöglichkeit eines Produktes durch die Optionen im Dialog *Software* kann nur angewendet werden, falls die Eigenschaft `ARPNOREPAIR` bei der Produktinstallation nicht gesetzt wurde.

Die Verwendung des Wartungsmodus lässt sich anhand des Installationsprotokolls feststellen. Die relevanten Eintragungen finden Sie in den Protokolleinträgen des Server-Prozesses. Diese sind an dem Präfix »MSI (s)« zu erkennen. Die verwendeten Reparaturoptionen können der Eigenschaft `REINSTALLMODE` entnommen werden, die dem Eintrag mit dem Präfix »Command Line:« angefügt ist.

```
MSI (s) (E8:AC) [14:18:34:078]: Command Line: REINSTALL=ALL REINSTALLMODE=omus CURRENTDIRECTORY=\\eagle\setup
CLIENTUILEVEL=3 CLIENTPROCESSID=1636
```

Der Wartungsmodus ist darüber hinaus explizit durch die Zeichenfolge »entering maintenance mode« gekennzeichnet.

```
MSI (s) (40:60) [16:11:00:859]: Product registered: entering maintenance mode
MSI (s) (40:60) [16:11:00:859]: PROPERTY CHANGE: Adding ProductState property. Its value is '5'.
```

Sie erhalten sogar die notwendigen Informationen, ob das Produkt im Benutzer- oder Maschinenkontext installiert wurde.

```
MSI (s) (40:60) [16:11:00:859]: Determined that existing product (either this product or the product being
upgraded with a patch) is installed per-machine.
```

Interessant ist auch die Eigenschaft `Installed`, die das Installationsdatum des Produktes enthält.

```
Property(S): Installed = 2004/12/07 23:08:48
```


Deinstallation

Die Deinstallation eines Produktes wird durch das Argument `/x` realisiert, wobei die Referenz auf das Produkt durch die Pfadangabe zum Installationspaket oder den `ProductCode` festgelegt werden kann.

- `msiexec /x <Pfad zum Installationspaket oder ProductCode>`

Mit dem Windows Installer 3.x kann hierzu auch die Standardbefehlszeilenoption »/uninstall« verwendet werden.

- `msiexec /uninstall <Pfad zum Installationspaket oder ProductCode >`

Die Deinstallation eines Produktes ist ebenfalls anhand des Installationsprotokolls zu erkennen. Der relevante Eintrag wird wiederum vom Server-Prozess erstellt und durch den Präfix »Command Line:« eingeleitet. Das wesentliche Argument ist hierbei die Eigenschaft `REMOVE`, die auf den Wert »ALL« festgelegt wird.

```
MSI (s) (40:64) [16:07:26:437]: Command Line: REMOVE=ALL CURRENTDIRECTORY==\\eagle\setup CLIENTUILEVEL=3
CLIENTPROCESSID=3624
```

Die Deinstallation kann ebenfalls über die Funktion `MsiInstallProduct()` des Windows Installer-API oder über die Methode `Installer.InstallProduct` des Windows Installer- Automationsmodells ausgeführt werden. Allerdings ist es hierzu erforderlich, die Zeichenfolge »REMOVE=ALL« der Eigenschaftsauflistung der Funktion anzufügen.

Administrative Installation

Bei der administrativen Installation wird nicht die ausführbare Anwendung auf dem Zielcomputer installiert. Der Windows Installer legt bei einer solchen Installationsart das Windows Installer-Paket und alle zugehörenden Dateien auf einem freigegebenen Netzwerkverzeichnis ab. Das Zielverzeichnis dieser Installation wird als `Installation Point` bezeichnet. Anwender können das Installationsprogramm direkt vom `Installation Point` ausführen, um eine Clientinstallation zu starten. Während der administrativen Installation ändert der Installer bestimmte Eigenschaften (Properties) in der Datenbank. Dateien, die entweder in Form einer internen oder externen Kabinettdatei (`.cab`) vorliegen, werden während des administrativen Installationsprozesses im unkomprimierten Zustand in einer Ordnerstruktur abgelegt. Eine administrative Installation kann von der Befehlszeile über den Parameter `/a` gestartet werden.

- `msiexec /a <Pfad zum Paket>`

Im Installationsprotokoll lässt sich eine administrative Installation anhand der Eigenschaft `ACTION` erkennen. Bei dieser Installationsart wird die Eigenschaft auf den Wert »ADMIN« festgelegt.

```
Property(S): ACTION = ADMIN
```

Eine administrative Installation kann durch die Funktion `MsiInstallProduct()` des Windows Installer-API oder über die Methode `Installer.InstallProduct` des Windows Installer- Automationsmodells ausgeführt werden. Es ist allerdings erforderlich, die Zeichenfolge »ACTION=ADMIN« der Eigenschaftsauflistung der Funktion anzufügen.

Angekündigte Installation

Die angekündigte Installation bezeichnet die Möglichkeit Installationsteile anzumelden, ohne aktuell benötigte Dateien zu installieren. Hierbei wird zwischen dem Zuweisen (Assign) und dem Veröffentlichen (Publish.) unterschieden. Um ein Produkt anzukündigen, müssen Sie die Option `/j` mit Angabe zusätzlicher Parameter und Angabe des Windows Installer-Paketes verwenden. Ein wesentlicher Punkt ist hierbei die Angabe des Kontexts, für den das Produkt angekündigt werden soll. Aus diesem Grund ist das Argument `/j` weiter zu qualifizieren, um dadurch den Benutzer- oder Maschinenkontext zu bestimmen. Die möglichen Befehlszeilenaufrufe können wie folgt definiert werden.

- `msiexec /j[u|m] <Pfad zum Paket>`
- `msiexec /j[u|m] <Pfad zum Paket> /t <Transformation>`
- `msiexec /j[u|m] <Pfad zum Paket> /g <Sprach-ID>`

Zu erkennen ist hierbei dass die Installation bei Bedarf immer mit dem Argument `/j` eingeleitet wird, wobei das »j« für »Just-In-Time« steht. Hieran muss der Kontext angefügt werden, wobei »u« für »User« steht und somit den aktuellen Benutzer kennzeichnet und »m« (maschine) den Maschinenkontext.

Im Installationsprotokoll lässt sich eine Installation bei Bedarf anhand der Eigenschaft `ACTION` erkennen. Bei dieser Installationsart wird die Eigenschaft auf den Wert »ADVERTISE« festgelegt.

```
Property(S): ACTION = ADVERTISE
```

Eine Installation bei Bedarf kann darüber hinaus durch die Funktionen `MsiAdvertiseProduct()` und `MsiAdvertiseProductEx()` des Windows Installer-API gestartet werden.

Installationsphasen

Nach dem Aufruf des Installationspaketes beginnt der Windows Installer die notwendigen Informationen zur Durchführung der Installation zu sammeln. Dieser Abschnitt der Installation wird als *Acquisition-Phase* bezeichnet. Während dieser Phase wird die Benutzeroberfläche angezeigt und der Anwender führt die Eingaben durch. Nach dieser Phase werden die gesammelten Daten an den Server-Prozess übergeben, der anschließend die entsprechenden Installationsskripte generiert und die eigentliche Installation durchführt. Dieser Abschnitt der Installation wird als *Execution-Phase* bezeichnet. Sollte es während der Installation zu Problemen kommen, wird der Wiederherstellungsmodus (Rollback) aufgerufen, wodurch der Computer wieder in den Zustand vor der Installation zurückgesetzt wird. Die folgende Abbildung demonstriert das Zusammenspiel der beiden Phasen.

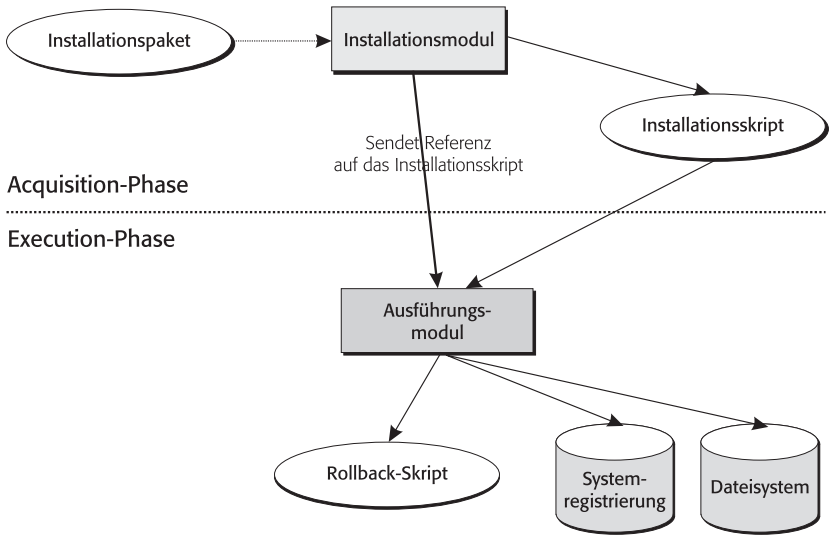

Abbildung 3.1: *Darstellung der Installationsphasen*

Acquisition-Phase

Die *Acquisition-Phase* dient im Wesentlichen dazu, Informationen für den Installationsprozess zu beschaffen und diese dem Ausführungsmodul (Execution Engine) zur Verfügung zu stellen. Die *Acquisition-Phase* wird immer mit den Privilegien des aktuellen Benutzers durchgeführt. Die Zuweisung von Systemprivilegien ist für diese Phase nicht möglich.

Sehen Sie zunächst das folgende Beispiel: Der Benutzer führt im *Microsoft Windows Explorer* einen Doppelklick auf eine Datei mit der Endung `.msi` durch. Dateien mit einer solchen Endung sind standardmäßig mit dem Windows Installer-Dienst verknüpft, so dass hierdurch die folgende Befehlszeile konstruiert und aufgerufen wird:

```
%windir%\system32\msiexec.exe /i <Pfad zum Installationspaket>
```

Es wird zunächst der Client-Prozess gestartet und diesem ein Verweis auf das Installationspaket übergeben. Optional ist hierbei auch die Übergabe von Windows Installer-Patches oder Transformationen möglich. Der Client-Prozess lädt das Installationspaket in den Arbeitsspeicher und ruft alle relevanten Informationen daraus ab. Im Folgenden führt er die Initialisierung aus, in der beispielsweise der Speicherbedarf ermittelt und der Status der Features überprüft wird. Anschließend wird die Benutzeroberfläche angezeigt und es findet die Interaktion mit dem Benutzer statt. Nachdem alle Eingaben vorgenommen wurden, wird der Server-Prozess gestartet. Diesem Prozess werden alle bereits ermittelten Informationen in Form einer Befehlszeile übergeben. In einem Installationsprotokoll ist diese Aktion als »Switching to server« bezeichnet.

```
MSI (c) (A8:8C) [17:45:37:453]: Switching to server: TARGETDIR="C:\" USERNAME="Andreas Kerl"
CLIENTPROCESSID="1448" CLIENTUILEVEL="0" CURRENTDIRECTORY="\\eagle\setup" INSTALLDIR="C:\Program Files\Darwin
Descriptor 1.0\" WWWROOT="C:\" GAC="C:\" SOURCEDIR="D:\001\" COMPANYNAME="Microsoft Deutschland GmbH"
ACTION="INSTALL" EXECUTEACTION="INSTALL" ROOTDRIVE="C:\" SECONDSEQUENCE="1" ALLUSERS="1"  ADDLOCAL=Complete
```

Sie erkennen in dem Ausschnitt des Installationsprotokolls, dass ausschließlich die öffentlichen Eigenschaften an den Server übergeben werden. Verfügt der Benutzer über administrative Privilegien werden alle öffentlichen Eigenschaften übergeben. Handelt es sich bei dem Benutzer um keinen Administrator und wird die Installation mit erhöhten Anwenderprivilegien ausgeführt, werden nur die als sicher erachteten Eigenschaften (`SecureCustomProperties`) übergeben.

HINWEIS: Wird die Installation mit der Standardbenutzeroberfläche (Basic-UI) oder ohne Anzeige einer Benutzeroberfläche durchgeführt, wird der Server-Prozess direkt aufgerufen.

Im Server-Prozess werden die vorliegenden Informationen ausgewertet und in Verbindung mit zusätzlichen Daten wie beispielsweise dem Systemstatus kombiniert. Aus diesen verfügbaren Informationen werden Operationsanweisungen generiert und in das Installationsskript übertragen. Diese Anweisungen beschreiben letztlich wie das System aktualisiert werden soll.

Nachdem die Skriptdatei erstellt wurde, wird die Befehlsausführung von der *Acquisition-Phase* an die *Execution-Phase* übergeben. Hierzu wird der Konfigurationsmanager aufgerufen und diesem der Pfad zum Installationsskript zugewiesen.

HINWEIS: Während der *Acquisition-Phase* werden keine Modifikationen am System vorgenommen werden.

Zusammenfassend betrachtet lässt sich erkennen, dass die *Acquisition-Phase* sowohl im Client- als auch im Server-Prozess ausgeführt wird. Das Ergebnis des Client-Prozesses ist eine Befehlszeile, die an den Server übergeben wird. Das Resultat des Server-Prozesses ist ein Installationsskript, das an dem Konfigurationsmanager zugewiesen wird.

Execution-Phase

Die *Execution-Phase* ist der Teil des Installationsprozesses in dem das Zielsystem modifiziert wird. Dem Konfigurationsmanager wird das Installationsskript vom serverseitigen Installationsmodul übergeben. Das Ausführungsmodul wird geladen und die Operationsanweisungen des Skriptes werden ausgeführt. In dieser Phase werden die Modifikationen am Dateisystem und in der Systemregistrierung vorgenommen.

Die *Execution-Phase* wird im Kontext des lokalen Systemkontos (`Local System Account`) ausgeführt, da hierdurch sichergestellt werden kann, dass der Installationsprozess die benötigten Rechte besitzt, um Eintragungen in allen Bereichen der Systemregistrierung und des Dateisystems vorzunehmen. Dieses bedeutet nicht, dass die vollständige Installation mit Privilegien ausgeführt wird, über die der aktuelle Benutzer nicht verfügt. Vielmehr wird für die *Execution-Phase* ein Identitätswechsel (Impersonation) auf den aktuellen Benutzer ausgeführt. Die Installationsaufgaben werden demzufolge mit den Anwenderprivilegien ausgeführt. Der Windows Installer legt jedoch zusätzliche Konfigurationsdaten in

Bereichen des Zielsystems ab, auf die ein Standardbenutzer keinen Zugriff hat. Diese Konfiguration wird mit den Privilegien des lokalen Systems vorgenommen.

HINWEIS: Der Administrator kann veranlassen, dass eine Installation mit erhöhten Anwenderprivilegien ausgeführt wird. Diese erhöhten Privilegien beziehen sich ausschließlich auf die *Execution-Phase*.

Vorteile der Unterteilung

Die Unterteilung des Installationsprozess in die zwei Phasen, sowie die Verwendung des Installationsskriptes zur Definition der Operationsanweisungen, erfolgt aus folgenden Gründen:

- Alle notwendigen Installationsoperationen werden ermittelt, bevor Modifikationen am System vorgenommen werden.
- Die Existenz einer vollständigen Liste von durchzuführenden Operationen ist Voraussetzung für einen transaktionalen Installationsprozess.
- In Fällen in denen ein Computerneustart erforderlich ist oder die Installation unterbrochen wurde, kann die Skriptdatei verwendet werden, um die Ausführung fortzusetzen.
- Konkurrierende Probleme werden durch das transaktionale Modell beseitigt. Gleichzeitige Aktualisierungen werden blockiert.
- Die Basisinstallation wird im Kontext des lokalen Benutzers ausgeführt. In einer abgesicherten Umgebung (Locked Down Environment) fehlen dem Prozess die notwendigen Berechtigungen, um die Aktualisierung des Systems vorzunehmen. Durch die Unterteilung in zwei Phasen kann die Kontrolle an einen Prozess übergeben werden, der die notwendigen Berechtigungen besitzt.
- Die Simulation einer Installation ist einfach durchzuführen, indem der Installationsprozess zwischen den beiden Phasen gestoppt wird. Die darauf folgende Analyse der Skriptdatei ermöglicht die Darstellung der durchzuführenden Operationen.
- Beide Phasen können separat getestet werden.

Nachdem nun der Installationsprozess in einer groben Darstellung anhand der Installationsphasen betrachtet wurde, möchte ich nun auf die hierbei verwendeten Prozesse eingehen.

Prozesse

Wie bereits gerade erläutert, werden zur Durchführung der Installation zwei unabhängige Prozesse verwendet. Diese Unterteilung in einen Client- und einen Server-Prozess ist nur unter den Betriebssystemen *Microsoft Windows NT 4.0*, *Microsoft Windows 2000*, *Microsoft Windows XP* und *Microsoft Windows Server 2003* vorhanden. Bei der Verwendung von *Microsoft Windows 95*, *Microsoft Windows 98* und *Microsoft Windows ME* wird lediglich ein Prozess ausgeführt.

Ungeachtet des verwendeten Betriebssystems existiert immer ein Client-Prozess, der mit den Privilegien des aktuellen Benutzers ausgeführt wird. Auf den Betriebsystemen, die auf der Windows NT Technologie basieren, existiert ein zusätzlicher Server-Prozess, der als Betriebssystemdienst integriert wurde. Unter gewissen Umständen kann dieser Server-Prozess angewiesen werden, Installationsaufgaben auszuführen, für die der aktuelle Benutzer normalerweise keine Berechtigungen hätte. Abbildung 3.2 stellt den Ablauf in den beiden Prozessen bei der Installation dar.

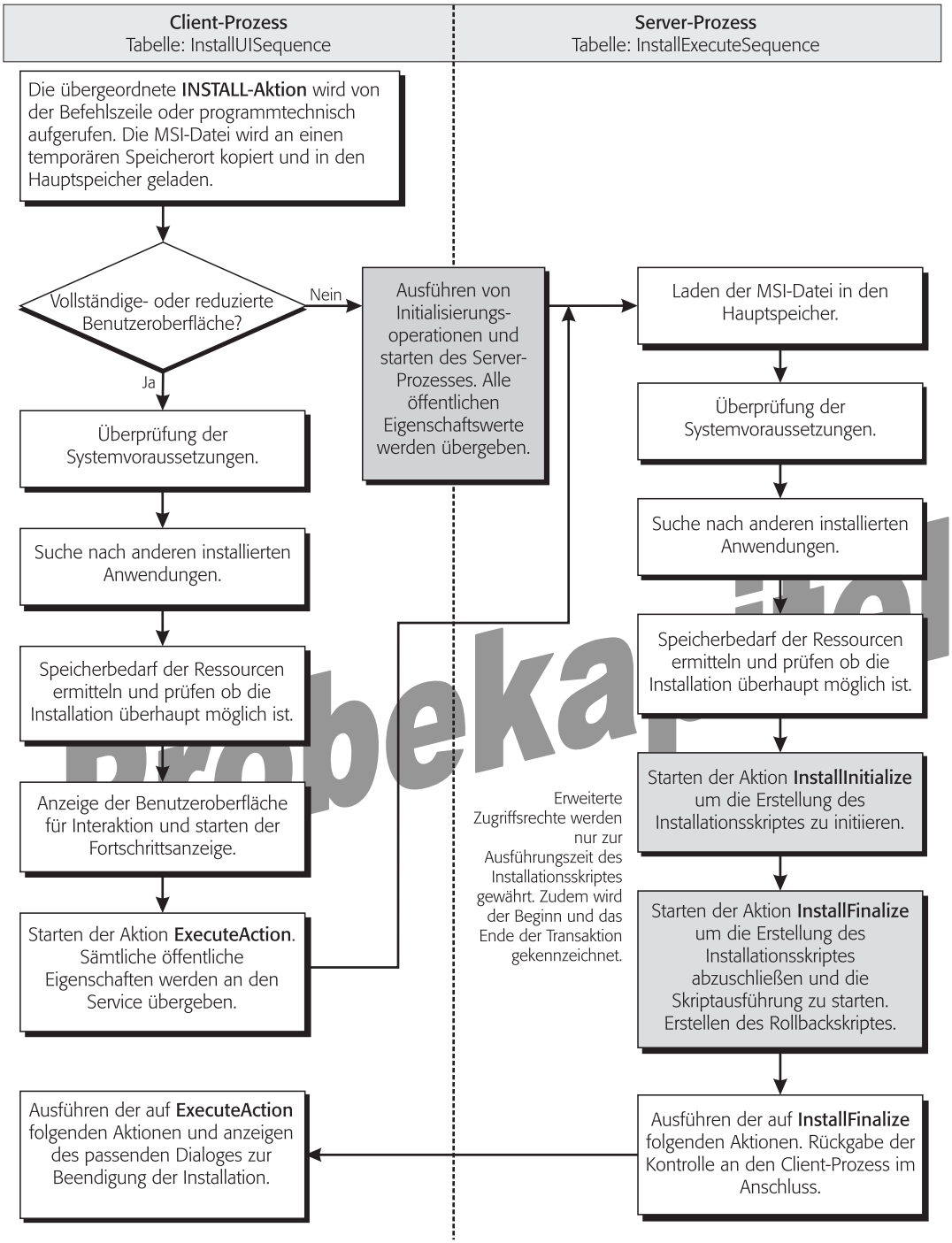

Abbildung 3.2: *Installationsprozesse unter Windows NT 4.0, Windows 2000, Windows XP und Windows Server 2003.*

Die vorherige Abbildung sollte Ihnen einen ersten Überblick über die Installationsprozesse des Windows Installers geben. Nachfolgend möchte ich die beiden Prozesse im Detail erläutern, wobei ich auf die Abläufe unter *Microsoft Windows 95*, *Microsoft Windows 98* und *Microsoft Windows ME* nicht näher eingehen werde, da dieses Buch den Windows Installer 3.x betrachtet und er für die dargestellten Betriebssysteme nicht verfügbar ist.

Client-Prozess

Der Client-Prozess ist für die Interaktion mit dem Benutzer während des Installationsprozesses verantwortlich. Starten Sie eine Installation und wechseln Sie im *Task-Manager* des Betriebssystems zum Register *Prozesse*. In der Liste finden Sie mindestens zwei Prozesse mit der Bezeichnung *msiexec.exe*. Ein

Prozess wird hierbei im Kontext des aktuellen Benutzers und ein Prozess im Kontext des lokalen Systems ausgeführt. Bei dem Prozess im Kontext des aktuellen Benutzers handelt es sich um den Client-Prozess.

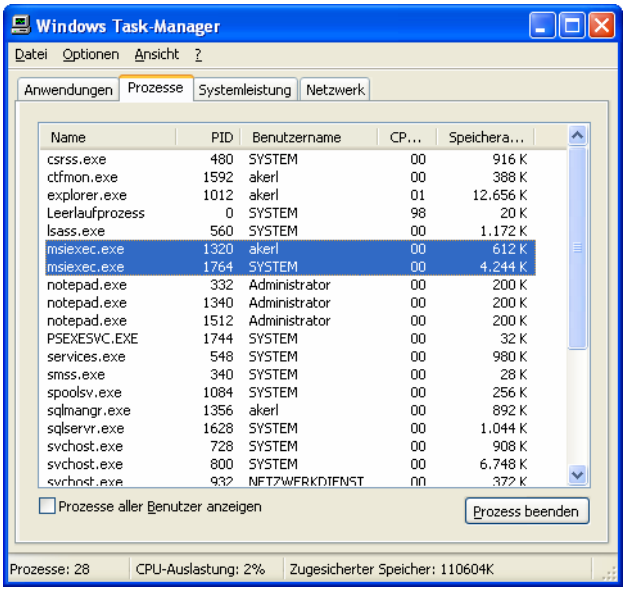

Abbildung 3.3: *Client-Prozess im Windows Task-Manager*

Der Client-Prozess besteht aus mehreren Komponenten, die funktionsspezifische Aufgaben im Installationsprozess verrichten. Eine Darstellung der Kommunikationswege zwischen diesen Objekten finden Sie in der folgenden Abbildung, eine Beschreibung der Objekte finden Sie im Anschluss.

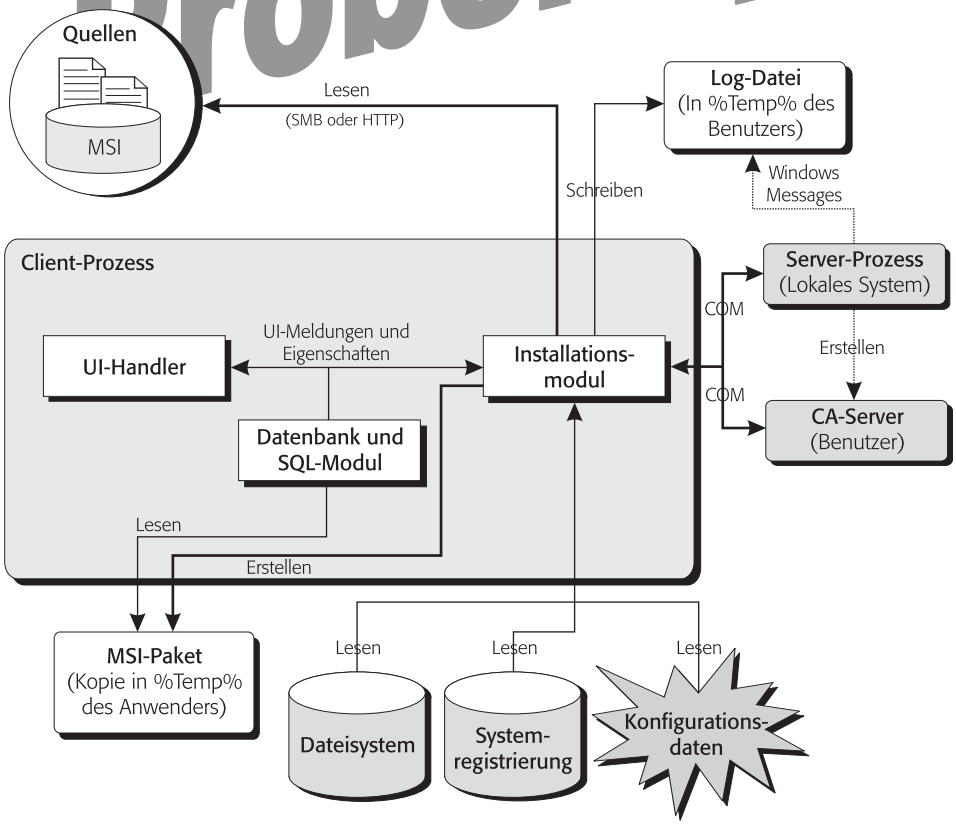

Abbildung 3.4: *Darstellung des Client-Prozesses*

UI-Handler

Der UI-Handler interagiert mit dem Benutzer und stellt Fortschritts- und Fehlerbehandlungsinformationen zu Verfügung. Der UI-Handler kann ebenfalls Eigenschaftswerte auf

Basis der Auswahl in der Dialogsequenz festlegen. Bei diesen Eigenschaften kann es sich auch um den Installationsstatus von Features oder Verzeichnisvorgaben handeln. Die Eigenschaften werden an das Installationsmodul übergeben, damit sie während der Installation verwendet werden können.

Datenbank und SQL-Modul

Das Datenbank- und das SQL-Modul stellen den Datenbankzugriff auf niedriger Ebene zur Verfügung, der benötigt wird, um das Installationspaket zu lesen. Bei dem Datenbankmodul handelt es sich um eine Schicht oberhalb des Interfaces `IStorage`. Das Modul stellt generelle Funktionalität zum Zugriff auf relationale Datenbanken durch die Verwendung von Tabellen und Cursor zur Verfügung. Das SQL-Modul befindet sich auf einer Ebene oberhalb des Datenbankmoduls und dient dazu, komplizierte Abfragen, einschließlich Joins, auf das Installationspaket zu ermöglichen. Vom Windows Installer wird nur ein Teil der vollständigen SQL Abfragesprache unterstützt.

Custom Action-Server (CA-Server)

Im Client-Prozess existiert ein Custom Action-Server, der als Host-Prozess im Kontext des aktuellen Benutzers ausgeführt wird, um jeden benutzerdefinierten Code auszuführen, der im Paket enthalten ist. Handelt es sich bei dem Installationspaket um ein vertrautes Paket (Trusted) werden die benutzerdefinierten Aktionen des Paketes ebenfalls vertraut ausgeführt. Unter 64-Bit Betriebssystemen existiert ein zweiter clientseitiger Custom Action-Server, der diese Architektur unterstützt. Somit existieren ein x86 und ein entsprechender 64-Bit Custom Action-Server, die immer im Kontext des Benutzers ausgeführt werden.

HINWEIS: Detaillierte Informationen zu dem Custom Action-Server finden Sie in einem separaten Kapitel.

Installationsmodul

Nach dem Start der Installation wird zunächst der Systemstatus abgerufen, um zu überprüfen, ob das zu installierende Produkt bereits auf dem lokalen System vorhanden ist. Ist das Produkt bereits installiert, wird das im Verzeichnis *%windir%\installer* zwischengespeicherte Paket für den weiteren Installationsprozess verwendet. Wird festgestellt, dass das Produkt bereits installiert wurde, jedoch das zwischengespeicherte Paket nicht zur Verfügung steht, wird das Originalpaket im Ordner *%windir%\installer* abgelegt. Hierzu wird das Paket aus dem Installationsverzeichnis verwendet, das in den Konfigurationsdaten für das Produkt definiert wurde. Wird hingegen festgestellt, dass das Produkt noch nicht installiert wurde, wird das referenzierte Installationspaket in den Ordner für temporäre Dateien des aktuellen Benutzers (*%userprofile%\Local Settings\Temp*) kopiert. Für Pakete, die über eine *URL* installiert werden, wird der Download des Paketes durch die Funktion `URLDownloadToCacheFile()` gesteuert.

Bevor die Aktionen des Installationspaketes ausgeführt werden, wird das Paket hinsichtlich der *Richtlinien für Softwareeinschränkungen (SAFER)* geprüft. Die Richtlinien für Softwareeinschränkungen ermöglichen es Systemadministratoren, die Ausführung eines Installationsprogrammes in Abhängigkeit zum Pfad des Paketes, der Internetzone, eines Hashes oder eines Zertifikates zu erlauben oder zu verweigern. Alle diesbezüglichen Überprüfungen werden durch die *SAFER-Infrastruktur* ausgeführt, bei der ein Identitätswechsel des angemeldeten Benutzers durchgeführt wird. Hierdurch wird sichergestellt, dass die benutzerbezogenen Richtlinien ordnungsgemäß angewendet werden können. Prozesse, die unter dem lokalen Systemkonto ausgeführt werden, werden durch *SAFER* nicht geprüft. Der Identifizierungslevel, der durch die Funktion `SaferIdentifyLevel()` ermittelt wird, wird für eine spätere Verwendung gespeichert. Das Ziel hierbei besteht in der Implementierung von Custom Action-Servern, die die Richtlinien für Softwareeinschränkungen berücksichtigen. Eine *SAFER-Überprüfung* wird nicht durchgeführt, wenn das Produkt bereits installiert ist, und das zwischengespeicherte Installationspaket verwendet wird. Die SAFER-Überprüfung wird dem Installationsprotokoll wie folgt angefügt:

```
MSI (c) (A8:8C) [17:45:33:515]: SOFTWARE RESTRICTION POLICY: \\eagle\setup\rtm.msi is not digitally signed
MSI (c) (A8:8C) [17:45:33:515]: SOFTWARE RESTRICTION POLICY: \\eagle\setup\rtm.msi is permitted to run at the
'unrestricted' authorization level.
```

HINWEIS: Die Richtlinien für Softwareeinschränkungen stehen unter den Betriebssystemen *Microsoft Windows XP* und *Microsoft Windows Server 2003* zur Verfügung.

Nachdem die *SAFER-Überprüfung* abgeschlossen ist und das Paket zu Ausführung akzeptiert wurde, wird vom Installationsmodul die Initialisierung durchgeführt, bei der auch die Argumente der Befehlszeile ausgewertet werden. Hierbei werden zunächst die Eigenschaften ausgewertet, die Windows Installer-Patches oder Transformationen referenzieren. Ist das Produkt bereits installiert, werden nur registrierte Transformationen verwendet, da nur im Rahmen der Basisinstallation einem Produkt entsprechende

Transformationen zugeordnet werden können. Windows Installer-Patches können hingegen zu jeder Zeit auf das Produkt angewendet werden.

Transformationen und Windows Installer-Patches werden bei der Auswertung der Befehlszeile in folgender Reihenfolge auf das Produkt angewendet:

- Kompatibilitätstransformationen, die für das Produkt benötigt werden (`APPLYPOINT=1`).
- Instanztransformationen
- Eingebetteten Sprachtransformationen
- Transformationen eines Patches
- Transformationen, die durch die Eigenschaft `TRANSFORMS` definiert wurden
- Kompatibilitätstransformationen, die für den Patch benötigt werden (`APPLYPOINT=2`)

Während der Initialisierungsphase werden Systemeigenschaften wie die Systemordner, der Virtuelle- und Physische Speicher, der Prozessor oder die verwendete Plattform abgerufen. Die Datenbank wird im schreibgeschützten Modus geöffnet, um ein Überschreiben der definierten Werte zu verhindern. Damit trotzdem Modifikationen an den definierten Eigenschaften vorgenommen werden können, wird eine temporäre Tabelle zur Aufnahme der Eigenschaften mit der Bezeichnung `_Property` im Speicher abgelegt. Die Existenz und den Inhalt der Tabelle `_Property` und weiterer temporärer Tabelle können hervorragend mit einem Debugger für Windows Installer-Pakete betrachtet werden. Das Installationsmodul prüft weiterhin die Gültigkeit des Schemas des Installationspaketes, also ob die benötigte Windows Installer-Version auf dem System vorhanden ist und ob es sich um eine gültige Systemarchitektur handelt. Die Installation eines 64-Bit Paketes auf einem 32-Bit Windows wird somit an dieser Stelle abgebrochen.

Als nächstes werden die Eigenschaften ausgewertet, die im Speicherbereich `AdminProperties` abgelegt sind. Dieser Speicherbereich wird im Installationspaket im Rahmen einer administrativen Installation erstellt. Im Folgenden überprüft das Installationsmodul, ob die notwendigen Privilegien zur Installation des Produktes vorhanden sind und ob die Installation mit erhöhten Anwenderprivilegien erfolgen soll. In Abbildung 3.5 wird Ihnen das Schema aufgezeigt, dass der Windows Installer verwendet, um diese Überprüfung durchzuführen.

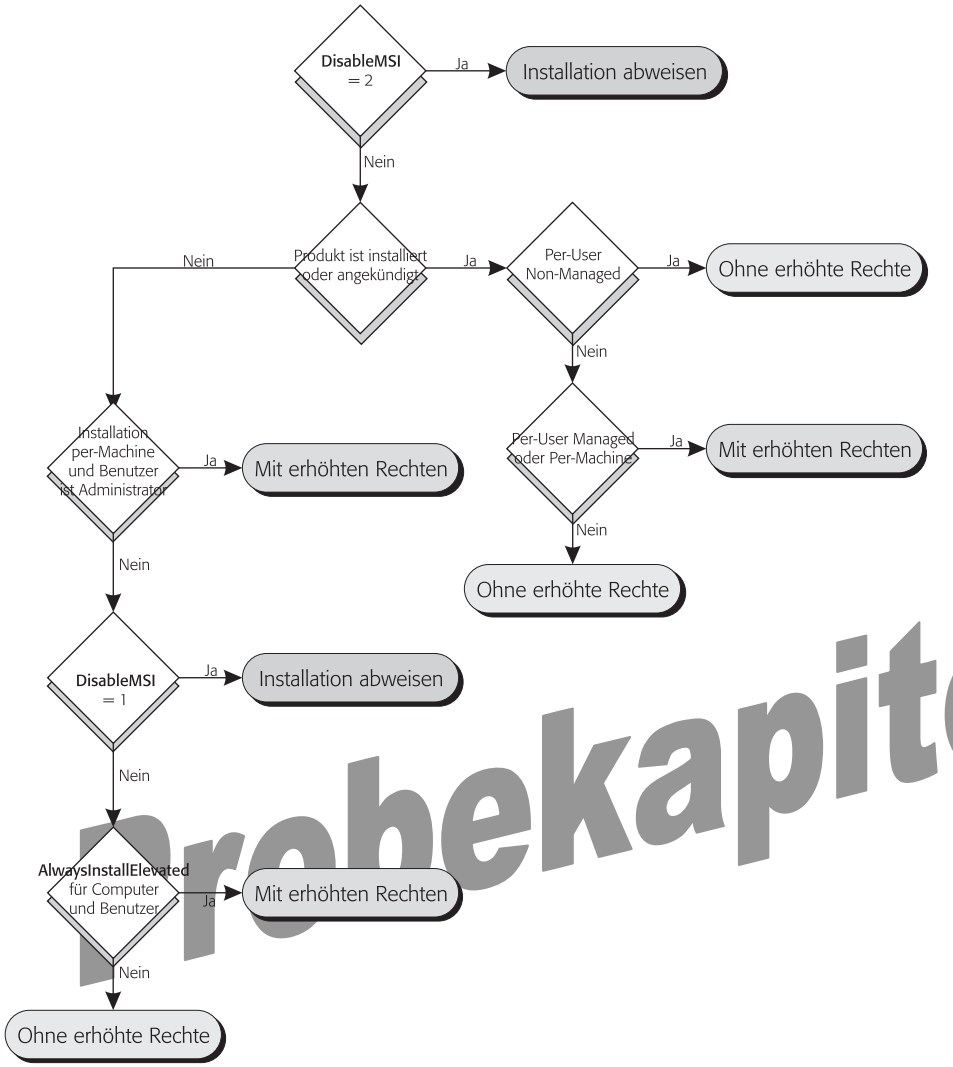

Abbildung 3.5: *Überprüfung der Installationsvoraussetzungen*

Wird festgestellt, dass die Systemrichtlinie `DisableMsi` auf den Wert »2« gesetzt wurde, werden alle bisherigen Aktionen zurückgesetzt und die Installation abgebrochen. Wird festgestellt, dass diese Richtlinie auf den Wert »1« gesetzt wurde, wird die Installation für alle Benutzerinstallationen abgebrochen, die nicht mit erhöhten Rechten ausgeführt werden (`Per-User Non-Managed`). Systemadministratoren können festlegen, dass eine Installation immer mit erhöhten Rechten ausgeführt wird. Hierzu muss die Richtlinie `AlwaysInstallElevated` für den Computer und den Benutzer aktiviert werden.

HINWEIS: Das Aktivieren der Gruppenrichtlinie `AlwaysInstallElevated` stellt ein potentielles Sicherheitsrisiko dar.

Nachdem die Transformationen und Patches angewendet wurden und der Installationskontext bestimmt wurde, werden die weiteren Eigenschaften der Befehlszeile auf das Produkt angewendet. Dieses bedeutet, dass die übergebenen Werte in die temporäre Tabelle `_Property` übertragen werden. Wird die Installation mit erhöhten Rechten ausgeführt, sind hierbei nur die Eigenschaften zulässig, die als sicher erachtet werden. Als sicher erachtete Eigenschaften müssen in der Eigenschaft `SecureCustomProperties` der Tabelle `Property` definiert werden. Der Windows Installer verfügt zusätzlich über den folgenden Standardvorrat an Eigenschaften, die bereits als sicher erachtet werden.

- ACTION, AFTERREBOOT, ALLUSERS, EXECUTEACTION, EXECUTEMODE, FILEADDDEFAULT,
- FILEADDLOCAL, FILEADDSOURCE, INSTALLLEVEL, LIMITUI, LOGACTION, NOCOMPANYNAME,
- NOUSERNAME, MSIINSTANCEGUID, MSINEWINSTANCE, PATCH, PRIMARYFOLDER, PROMPTROLLBACKCOST,
- REBOOT, REINSTALL, REINSTALLMODE, RESUME, SEQUENCE, SHORTFILENAMES,
- TRANSFORMS, TRANSFORMSATSOURCE

Diese Einschränkung kann durch die Eigenschaft `EnableUserControl` oder durch die gleichnamige Systemrichtlinie außer Kraft gesetzt werden. In diesem Fall werden alle öffentlichen Eigenschaften an den Server-Prozess übergeben. Nachdem noch einige weitere Eigenschaften verarbeitet wurden ist die Initialisierungsphase des Client-Prozesses abgeschlossen.

Nach der Initialisierung wird der eigentliche Installationsprozess durch den Aufruf einer der drei Top-Level-Aktionen gestartet, indem alle Aktionen der zugehörigen Sequenztabelle für die Benutzeroberfläche ausgeführt werden. Die folgenden Tabellen werden bei der jeweiligen Aktion verwendet:

- **INSTALL:** `InstallUISequence`
- **ADMIN:** `AdminUISequence`
- **ADVERTISE:** `AdvtUISequence`

HINWEIS: Eine Benutzeroberfläche ist während der Produktankündigung (`ADVERTISE`) nicht verfügbar. Die Tabelle `AdvtUISequence` enthält aus diesem Grund keine Eintragungen.

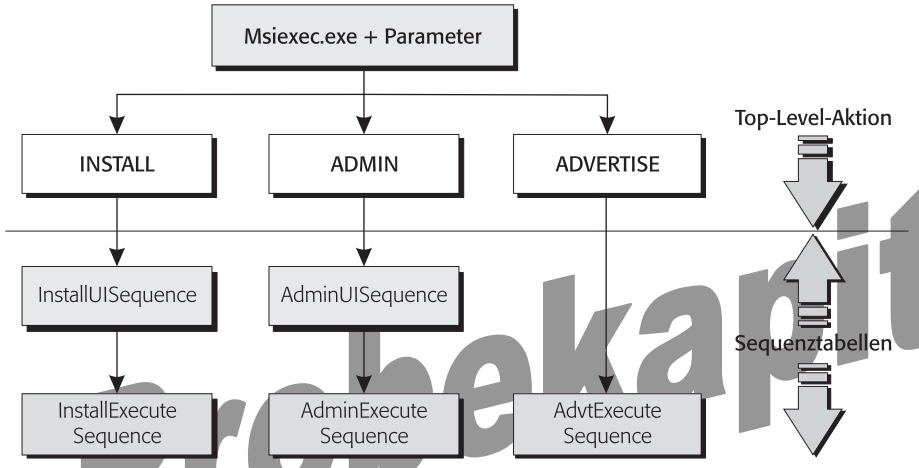

Abbildung 3.6: *Verwendete Tabellen in Abhängigkeit zur Top-Level-Aktion*

Die gerade vorgestellten Sequenztabellen enthalten Informationen zur Anzeige der Benutzeroberfläche, sowie zur Darstellung von Fehler- und anderen Meldungen. Weiterhin können einige Standardaktionen und auch benutzerdefinierte Aktionen in diesen Tabellen verwendet werden. In vielen Installationspaketen sind hier Standardaktionen zur Überprüfung der Systemvoraussetzungen (`LaunchCondition`), zur Suche nach abhängigen Komponenten (`AppSearch`) und zur Berechnung des benötigten Speicherbedarfs (`Costing`) implementiert. Die Berechnung des benötigten Speichers wird durch die nachfolgenden Standardaktionen ausgeführt:

- `CostInitialize`
- `FileCost`
- `CostFinalize`

Durch diese Aktionen wird ebenfalls der Installationsstatus der Features und Komponenten des Windows Installer-Paketes festgestellt und die Installationsverzeichnisse bestimmt. Sie müssen beim Design eines Installationspaketes sicherstellen, dass diese Aktionen ausgeführt wurden, bevor ein Dialog zur Auswahl der Installationsverzeichnisse oder zum Festlegen der Features angezeigt wird.

Nach Abschluss der Aktion `CostFinalize` werden die ermittelten Informationen, sowie ein Dump der Tabelle `Directory` dem Installationsprotokoll angefügt, wie der folgende Ausschnitt zeigt:

```
MSI (c) (A8:8C) [17:45:33:640]: PROPERTY CHANGE: Adding OutOfDiskSpace property. Its value is '0'.
MSI (c) (A8:8C) [17:45:33:640]: PROPERTY CHANGE: Adding OutOfNoRbDiskSpace property. Its value is '0'.
MSI (c) (A8:8C) [17:45:33:640]: PROPERTY CHANGE: Adding PrimaryVolumeSpaceAvailable property. Its value is '0'.
MSI (c) (A8:8C) [17:45:33:640]: PROPERTY CHANGE: Adding PrimaryVolumeSpaceRequired property. Its value is '0'.
MSI (c) (A8:8C) [17:45:33:640]: PROPERTY CHANGE: Adding PrimaryVolumeSpaceRemaining property. Its value is '0'.
MSI (c) (A8:8C) [17:45:33:640]: PROPERTY CHANGE: Adding TARGETDIR property. Its value is 'C:\'.
MSI (c) (A8:8C) [17:45:33:640]: PROPERTY CHANGE: Adding GAC property. Its value is 'C:\'.
MSI (c) (A8:8C) [17:45:33:640]: PROPERTY CHANGE: Adding INSTALLDIR property. Its value is 'C:\Program
Files\Darwin Descriptor 1.0\'.
MSI (c) (A8:8C) [17:45:33:640]: Target path resolution complete. Dumping Directory table...
MSI (c) (A8:8C) [17:45:33:640]: Dir (target): Key: TARGETDIR , Object: C:\
MSI (c) (A8:8C) [17:45:33:640]: Dir (target): Key: PersonalFolder     , Object: C:\Documents and Settings\AK\My
Documents\
```

Die wichtigste Aktion in der jeweiligen Sequenztabelle in diesem Modul ist `ExecuteAction`. Bei Ausführen dieser Aktion wird die Befehlsausführung an den Server-Prozess übergeben. `ExecuteAction` stellt hierzu eine Verbindung zum Konfigurationsmanager des Server-Prozesses her, indem `CoCreateInstance()` aufgerufen wird. Das Installationsmodul des Client-Prozesses prüft daraufhin, ob der Mutex `_MsiExecute` bereits existiert. Ist dies der Fall wird die Installation mit dem Fehler `ERROR_INSTALL_ALREADY_RUNNING` abgebrochen. Dieses ist dadurch bedingt, dass der Windows Installer die Verwendung von mehreren Client-Prozessen unterstützt, jedoch nur die Ausführung eines Server-Prozesses erlaubt. Der Client-Prozess führt einen Remoteprozeduraufruf (Remote-Procedure-Call) zum Server durch, um die eigentliche Installation einzuleiten. Das folgende Listing zeigt, wie die Existenz des Mutex programmtechnisch geprüft werden kann.

```
[DllImport("kernel32.dll", SetLastError=true)]
static extern int OpenMutexA ( int dwDesiredAccess, int bInheritHandle, string lpName);

[DllImport("kernel32.dll", SetLastError=true)]
static extern int CloseHandle ( int hObject);

private const int STANDARD_RIGHTS_REQUIRED = 0x000F0000;
private const int SYNCHRONIZE = 0x00100000;
private const int MUTANT_QUERY_STATE = 0x0001;
private const int MUTEX_ALL_ACCESS = (STANDARD_RIGHTS_REQUIRED | SYNCHRONIZE | MUTANT_QUERY_STATE);

public static bool CheckExecuteMutex()
{
    int handle = OpenMutexA(MUTEX_ALL_ACCESS, 0, "_MSIExecute");

    if (handle == 0)
        return false;

    CloseHandle(handle);

    return true;
}
```

Listing 3.1: *Existenz des Mutex mit Microsoft Visual C# prüfen*

Nachdem die serverseitige Installation beendet wurde, wird vom Client der Rückgabewert ausgewertet und die temporären Dateien werden gelöscht. Ist ein Computerneustart zum Abschluss der Installation erforderlich, weist der Client den Server an, diesen Neustart auszuführen. Die Feststellung ob ein Neustart des Computers erforderlich ist, wird immer vom Server getroffen. Ist dies der Fall wird der Client darüber informiert, so dass dieser entsprechende Dialogfelder anzeigen kann. Der Client fordert anschließend den Server auf, den Neustart einzuleiten.

Server-Prozess

Der Server-Prozess ist primär dafür verantwortlich, die tatsächlichen Modifikationen am System vorzunehmen. Er besteht aus mehreren Komponenten, die funktionsspezifische Aufgaben im Installationsprozess verrichten. Wie schon bei der Beschreibung zum Client-Prozess werde ich zunächst die Kommunikationswege und die benötigten Objekte erläutern. Hierbei werde ich auf die Betrachtung einiger Objekte verzichten, da der Aufbau und die Funktionalität dieser Objekte identisch mit dem Client-Prozess sind.

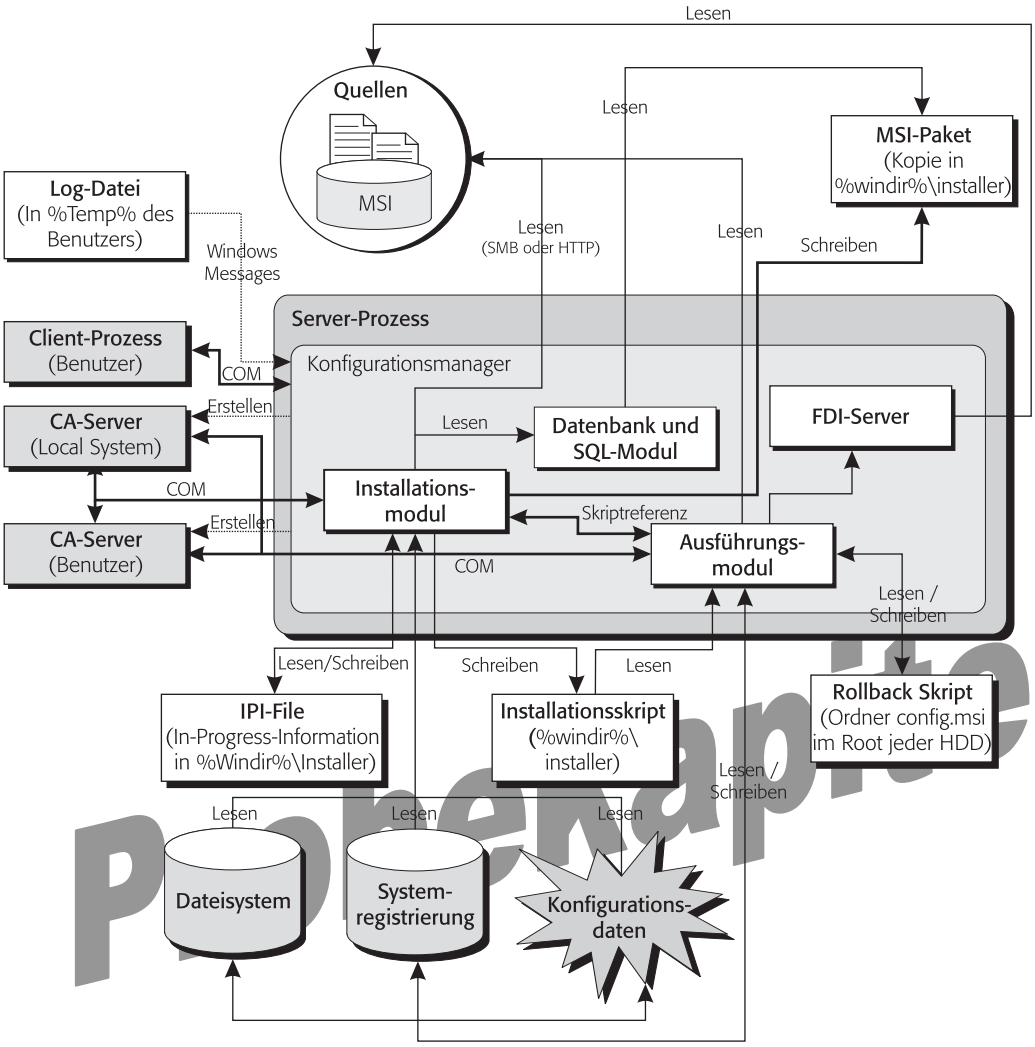

***Abbildung 3.7**: Darstellung des Server-Prozesses*

FDI-Server

Beim FDI-Server (File Decompression Interface) handelt es sich um eine Komponente, die vom Ausführungsmodul verwendet wird, um die zu installierenden Dateien aus den Kabinettdateien (`.cab`) zu extrahieren.

Custom Action-Server

Beim Custom Action-Server handelt es sich um eine Komponente, die individuellen Code ausführen kann, der im Installationspaket definiert wurde. Auf einer x86 Plattform existieren im Server-Prozess zwei Custom Action-Server und auf einer 64-Bit-Plattform vier Custom Action-Server. Bei den zwei Servern auf der x86 Plattform handelt es sich zum einen um den impersonierten Server, also dem Prozess der mit den Privilegien des Benutzers ausgeführt wird und zum anderen um den Prozess, der mit den erhöhten Rechten des lokalen Systemkontos ausgeführt wird. Bei den 64-Bit-Plattformen wurden die Anzahl der Server verdoppelt, um sowohl 32-Bit Code als auch 64-Bit Code auszuführen.

Installationsmodul

Die Initialisierung des Server-Prozesses geschieht auf eine ähnliche Weise wie die des Clients. Die primäre Komponente dieses Prozesses ist der Konfigurationsmanager. Der Konfigurationsmanager empfängt die Installationsaufforderung vom Client, wozu ihm der `ProductCode`, die Top-Level-Aktion, die Befehlszeile, Informationen zur Protokollierung und eine Referenz auf den UI-Handler übergeben werden. Zunächst wird geprüft ob der Mutex `_MSIExecute` existiert. Im Anschluss wird die Methode `RunInstall()` aufgerufen, wodurch das Installationsmodul erstellt und der Installationsprozess gestartet wird. Um sicherheitsrelevante Risiken auszuschließen, kann der Server keine diesbezüglichen Informationen vom Client empfangen. Aus diesem Grund wird der Systemstatus erneut analysiert und eine Überprüfung des Paketes hinsichtlich der *Richtlinien für Softwareeinschränkungen (SAFER)* durchgeführt. Falls das Produkt noch nicht installiert ist, wird das Installationspaket im Verzeichnis *%windir%\installer*

gespeichert. Diese Vorgehensweise hat auch wieder sicherheitsrelevante Hintergründe. Modifikationen von Dateien in diesem Verzeichnis, können nur von Administratoren oder durch das lokale Systemkonto vorgenommen werden.

Der Server-Prozess kann keine Benutzeroberfläche darstellen. Aus diesem Grund werden Meldungen zum Client gesendet, die dann vom clientseitigen UI-Handler ausgewertet und dargestellt werden. Wird eine Installation ohne Benutzeroberfläche durchgeführt, werden keine diesbezüglichen Meldungen vom Server versendet.

WICHTIG: Vermeiden Sie die Implementierung von Oberflächenelementen im Server-Prozess. In einer Terminal-Server Session und bei der Verwendung der schnellen Benutzerumschaltung (Fast-User-Switching) wird dieses Modul in einer anderen Session (immer Session »0«) ausgeführt als in der Session des aktuellen Benutzers. Würde der Server eine Benutzeroberfläche anzeigen, würde diese in einer anderen Session bzw. auf einem anderen Desktop angezeigt. Weiterhin steht in einigen Szenarien kein gültiger Desktop zur Verfügung (Winlogon), so dass die Darstellung einer Benutzeroberfläche zu einem Systemstillstand führen würde.

Der Client-Prozess übergibt dem Server neben vielen Informationen auch die aktuelle Top-Level-Aktion. In Abhängigkeit zu dieser Aktion wird die Tabelle für die Ausführungssequenz bestimmt und die darin enthaltenen Aktionen ausgeführt. Folgende Tabellen werden hierzu in Abhängigkeit zur Top-Level-Aktion verwendet (Siehe auch Abbildung 3.6):

- **Install:** InstallExecuteSequence
- **Admin:** AdminExecuteSequence
- **Advertise:** AdvtExecuteSequence

Wie im clientseitigen Prozess werden zunächst Aktionen zur Bestimmung der Installationsinformationen (Acquisition-Phase) ausgeführt. Im ersten Teil werden hier die Systemvoraussetzungen (LaunchCondition) geprüft und nach abhängigen Ressourcen gesucht (AppSearch). Es können auch benutzerdefinierte Aktionen in diesem Teil zur Anwendung kommen. Im weiteren Verlauf werden die Aktionen zur Berechung des Speicherbedarfs (CostInitialize, FileCost und CostFinalize) ausgeführt. Die Bezeichnung »Costing-Actions« für diese Aktionen ist jedoch nicht sehr glücklich gewählt, denn die Ausführung der Aktionen ermittelt nicht nur den Speicherbedarf, sondern dient auch dazu, die Installationsverzeichnisse zu bestimmen. Um diese Aktivitäten durchzuführen, werden die folgenden Objekte verwendet:

- **SelectionManager:** Verwaltet den Status der Features und Komponenten, die im Installationspaket definiert wurden. Hierbei werden der Systemstatus und die individuelle Konfiguration der Features berücksichtigt.
- **DirectoryManager:** Bestimmt die tatsächlichen Zielverzeichnisse anhand der Systemkonfiguration und der Definition im Installationspaket.

Bei der Berechnung des Speicherbedarfs wird sichergestellt, dass das Zielsystem über ausreichend Speicherplatz auf den Datenträgern verfügt. Hierbei werden Szenarien für die lokale Installation von Komponenten, die Ausführung von Komponenten vom Quellmedium und für Komponenten, die nicht installiert werden sollen, getrennt berücksichtigt. Die Werte werden jeweils für Rollback-Szenarien und für Szenarien in denen kein Rollback ausgeführt wird, berechnet. Zur Ermittlung des tatsächlichen Installationsumfangs und somit der Bestimmung des benötigten Festplattenspeichers, benötigt der SelectionManager die folgenden Informationen zu den betreffenden Komponenten:

- Installationsstatus (InstallState): Der bisherige Status der Komponente auf dem Zielsystem.
- Anforderungsstatus (RequestState): Der Status der Komponente, der nach der Installation hergestellt sein soll.
- Aktionsstatus (ActionState): Die Aktion, die ausgeführt werden muss, um die Komponente vom Installationsstatus in den Anforderungsstatus zu überführen.

Es ist zu beachten, dass der Aktionsstatus nicht immer identisch mit dem Anforderungsstatus sein muss. Bei Komponenten die bereits in einer höheren Version auf dem Zielsystem vorliegen, würde sich für eine lokale Installation dieser Komponente folgende Darstellung ergeben:

- Installationsstatus: Local
- Anforderungsstatus: Local
- Aktionsstatus: Null

Nachdem der benötigte Speicherbedarf ermittelt wurde, wird während der Aktion InstallValidate geprüft, ob ausreichend Festplattenspeicher zur Verfügung steht. Mit einem Debugger für Windows Installer-Pakete können Sie diese Informationen sehr einfach ermitteln. Setzen Sie hierzu einen Haltepunkt (Breakpoint) auf die Aktion InstallInitalize und führen Sie die Installation bis zu diesem Punkt aus. Betrachten Sie nun die Eintragungen in der Tabelle Component. Diese Tabelle wurde um einige temporäre Spalten erweitert, die die ermittelten Werte enthalten.

Spalte	Beschreibung
Installed	Enthält einen Wert der den Installationsstatus festlegt. Gültige Werte sind lediglich »1« oder »0«. Bei dem Wert »1« ist die Komponente bereits installiert, beim Wert »0« hingegen nicht.
Action	Enthält einen Wert der den Anforderungsstatus festlegt. Dieses Feld enthält lediglich den Wert »1«, wenn die Komponente lokal installiert werden soll. In allen anderen Fällen enthält dieses Feld den Wert Null.
ActionRequest	Enthält einen Wert der den Aktionsstatus festlegt. Dieses Feld enthält lediglich den Wert »1«, wenn die Komponente lokal installiert wird. In allen anderen Fällen enthält dieses Feld den Wert Null.
LocalCost	Enthält den Speicherbedarf für eine lokale Installation der Komponente.
NoRbLocalCost	Enthält den Speicherbedarf für eine lokale Installation der Komponente. Hierbei wird zusätzlicher Speicherbedarf für den Rollback nicht berücksichtigt.
SourceCost	Enthält den Speicherbedarf, falls die Komponente vom Quellmedium ausgeführt wird.
RemoveCost	Enthält den Speicherbedarf, der eingespart wird, falls die Komponente vom Zielrechner entfernt wird.
NoRbSourceCost	Enthält den Speicherbedarf, falls die Komponente vom Quellmedium ausgeführt wird. Hierbei wird der zusätzliche Speicherbedarf für den Rollback nicht berücksichtigt.
NoRbRemoveCost	Enthält den Speicherbedarf, der eingespart wird, falls die Komponente vom Zielrechner entfernt wird. Hierbei wird der zusätzliche Speicherbedarf für den Rollback nicht berücksichtigt.
ARPLocalCost	Enthält den Speicherbedarf, der für das lokale Cachen der Komponente benötigt wird.
NoRbARPLocalCost	Enthält den Speicherbedarf, der für das lokale Cachen der Komponente benötigt wird. Hierbei wird der zusätzliche Speicherbedarf für den Rollback nicht berücksichtigt.

Tabelle 3.2: *Temporäre Spalten der Tabelle »Component«*

Der berechnete Speicherbedarf wird in den entsprechenden Spalten in Einheiten à 512 Byte dargestellt. Verwenden Sie beispielsweise eine Komponente, die bei einer lokalen Installation 53.248 Byte Speicherplatz benötigen würde, würde das Feld LocalCost den Wert 104 (= 53.248 / 512) enthalten.

Während der Aktion InstallValidate wird weiterhin geprüft, ob zu ersetzende Dateien in Verwendung sind. Diese Prüfung ist nur für PE-Dateien (Portable Executable File Format) möglich, die ein *Window-Handle* zurückliefern. Die Aktion InstallInitialize schließt die Initialisierungsphase ab. Falls die Aktion RemoveExistingProducts an dieser Stelle der Sequenztabelle eingeordnet wurde, wird zunächst geprüft, ob die Deinstallation eines Produktes erforderlich und für den aktuellen Benutzer auch zulässig ist. Eine Deinstallation ist nur möglich, wenn eine der folgenden Bedingungen erfüllt ist:

- Der Anwender verfügt über Administratorenrechte.
- Die Richtlinie AlwaysInstallElevated ist für den Computer und den Anwender aktiviert.
- Installation wird für den Benutzer ausgeführt (Per-User Installation)
- Installation wird für den Computer ausgeführt. Hierbei handelt es sich um ein verwaltetes Upgrade einer existierenden Installation.

Trifft keine dieser Bedingungen zu, ist eine Deinstallation nicht erlaubt und die Installation wird abgebrochen. Wird die Installation hingegen fortgesetzt, wird durch die Methode BeginTransaction() die Transaktion gestartet, in der die Modifikationen des Zielsystems vorgenommen werden. Die Methode BeginTransaction() erstellt zunächst einen Wiederherstellungspunkt (PCHealth) aber nur wenn die Installation mit einer Benutzeroberfläche ausgeführt wird. Im Folgenden wird der Server gesperrt. Hierzu wird in der Systemregistrierung der Schlüssel HKLM\Software\Microsoft\Windows\CurrentVersion\Installer\InProgress erstellt. Existiert dieser Schlüssel bereits, kann die Installation nicht fortgesetzt werden, da bereits eine weitere Installation aktiv ist. Es wird weiterhin eine Datei zur Aufnahme von Informationen erstellt, die im Falle eines Computerneustarts benötigt werden. Hierbei handelt es sich um ein Verbunddokument, dass im Ordner *%windir%\installer* erstellt wird. Diese Datei wird auch als IPI-Datei (In-Progress-Information) bezeichnet. Der Pfad zu dieser Datei wird in dem gerade dargestellten Schlüssel der Systemregistrierung abgelegt.

Die Installation wird nun mit den Aktionen (InstallFiles, WriteRegistryValues etc.) fortgesetzt, die Operationsanweisungen (OPCodes) erstellen. Es wird zunächst eine Skriptdatei im Ordner *%windir%\installer* angelegt. Der Dateiname wird aus einer temporären Zeichenfolge mit dem Präfix »Msi« gebildet. Der Typ der Skriptdatei ist abhängig vom jeweiligen Ausführungsmodus der Installation. Als erste Eintragung wird der Header in die Skriptdatei geschrieben. Der Header besteht aus den folgenden Daten:

- **Signature:** Konstanter Wert (Immer 0x534f5849L)
- **Version:** Version des Windows Installers
- **Timestamp:** Zeitstempel

- **LangId:** Sprache des Paketes
- **ScriptType:** Typ des Skriptes
- **ScriptMajorVersion** und **ScriptMinorVersion:** Felder zur Festlegung der Kompatibilität des Skriptes.
- **ScriptAttributes:** Zusätzliche Informationen zur Ausführung des Skriptes wie Elevate oder UseTSRegistry.

Als zweite Eintragung werden die Produktinformationen geschrieben, die Identifizierungsmerkmale des Produktes und des Windows Installer-Paketes enthalten. Diese Informationen und der Skriptheader sind im folgenden Auszug aus einem Installationsskript aufgeführt.

```
MSI (s) (F8:FC) [17:45:44:734]: Executing op: Header(Signature=1397708873,Version=310,
Timestamp=846368183,LangId=1031,Platform=0,ScriptType=1,ScriptMajorVersion=21,
ScriptMinorVersion=4,ScriptAttributes=1)
MSI (s) (F8:FC) [17:45:44:734]: Executing op: ProductInfo(ProductKey={41F35721-4039-4B0E-AD89-
1A851084AEAB},ProductName=Darwin Descriptor 1.0,PackageName=Setup.msi,Language=1031,Version=16777216,
Assignment=1,ObsoleteArg=0, ProductIcon=MyIcon.exe,0,,PackageCode={E7135BE9-64A5-468D-A5E3-
EFCBBAB9285F},,,InstanceType=0,LUASetting=0,RemoteURTInstalls=0)
```

Das Installationsmodul enthält ein Objekt zum Auswerten der Bedingungen von Standardaktionen (ConditionParser). Gibt eine Bedingung den Wert True zurück, wird für diese Aktion eine Operationsanweisung (OPCode) in das Skript übertragen. Die Operationsanweisung wird in Abhängigkeit zum Installationsstatus, zum Anforderungsstatus und zum Aktionsstatus der Features und Komponenten aus einem Satz von vordefinierten Anweisungen für diese Aktion bestimmt. Jede der Operationsanweisungen entspricht einer gespeicherten Prozedur im Datenbankmodul, bei denen die Signaturen übereinstimmen müssen.

Diese Vorgehensweise bezieht sich auf den überwiegenden Teil der Standardaktionen. Einige Aktionen werden jedoch auf eine abweichende Art verwendet:

- **InstallExecute:** Die Aktion InstallExecute muss zwischen den Aktionen InstallInitialize und InstallFinalize definiert werden. Durch den Aufruf von InstallExecute werden alle im Skript befindlichen Aktionen seit dem Start der Installation oder dem letzten Aufruf von InstallExecute oder InstallExecuteAgain ausgeführt. Das System wird hierdurch aktualisiert ohne jedoch die Transaktion zu beenden.
- **InstallExecuteAgain:** Die Aktion InstallExecuteAgain ist identisch mit der Aktion InstallExecute. Sie verfügt lediglich über eine andere Bezeichnung, da der Name der Aktion der Primärschlüssel in der jeweiligen Tabelle ist.
- **ForceReboot:** Die Aktion ForceReboot veranlasst einen umgehenden Neustart des Systems. Dem Anwender wird hierzu eine Dialogbox angezeigt, falls die Darstellungsform der Benutzeroberfläche dieses erlaubt. Beim Ausführen von ForceReboot werden alle vorherigen im Skript befindlichen Aktionen abgeschlossen. Der Installer erstellt den Schlüssel HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce in der Systemregistrierung und fügt den Productcode oder den Namen des Installationspaketes als Wert hinzu. Der Installer erstellt ebenfalls eine Befehlszeile, die diesem Schlüssel als Wert hinzugefügt wird. Diese Befehlszeile verfügt über den folgenden Aufbau: »AFTERREBOOT=1 RUNONCEENTRY=[Name des Eintrages]«. Der Eintrag RUNONCEENTRY enthält einen Verweis auf einen speziellen RunOnce-Schlüssel für den Windows Installer, der unter HKLM\Software\Microsoft\Windows\CurrentVersion\Installer\RunOnceEntries angelegt wird. Diese Informationen werden benötigt, um die Installation auf Basis der IPI-Datei nach dem Neustart fortzusetzen.
- **ScheduleReboot:** Die Action ScheduleReboot veranlasst den Neustart des Systems nach dem Abschluss der Installation.

Nachdem alle Operationsanweisungen in das Skript eingetragen wurden, wird der Abschluss der serverseitigen *Acquisition-Phase* mit der Anweisung End im Skript gekennzeichnet. Die *Execution-Phase* wird nun eingeleitet, indem das Installationsmodul die Anweisung RunScript() zum Konfigurationsmanager sendet. Der Konfigurationsmanager erstellt hierzu das Ausführungsmodul (Script Executor) dem die folgenden Argumente übergeben werden:

- **ScriptFile:** Referenz auf das Installationsskript
- **MessageHandler:** Referenz auf das Objekt zur Darstellung der Meldungen.
- **DirectoryManager:** Referenz auf das Objekt zur Verwaltung der Verzeichnisse.
- **RollbackEnabled:** Boolescher Wert der festlegt, ob der Rollback aktiviert ist oder nicht.

Die Kontrolle wird hierdurch vom Installationsmodul an das Ausführungsmodul übergeben, wobei auch ein Identitätswechsel durchgeführt wird. Das Ausführungsmodul prüft zunächst ob die Installation mit erhöhten Rechten ausgeführt werden muss, indem die im Scriptheader definierten Attribute ausgewertet

werden. Das Ausführungsmodul führt nun die Operationsanweisungen des Skriptes aus und modifiziert hierbei das Zielsystem.

Die Ausführung des Skriptes wird im Installationsprotokoll ebenfalls vermerkt. Hierbei wird der Beginn durch die Zeichenfolge »Running Script« und das Ende durch die Operationsanweisung »End« gekennzeichnet, wie auch der nachfolgende Ausschnitt zeigt:

```
Action start 17:45:44: InstallFinalize.
MSI (s) (F8:FC) [17:45:44:734]: Running Script: C:\WINDOWS\Installer\MSI3C.tmp
MSI (s) (F8:FC) [17:45:44:734]: Executing op: Header(Signature=1397708873,Version=301,
Timestamp=846368183,LangId=1031,Platform=0,ScriptType=1,ScriptMajorVersion=21,
ScriptMinorVersion=4,ScriptAttributes=1)
MSI (s) (F8:FC) [17:45:44:734]: Executing op: ProductInfo(ProductKey={41F35721-4039-4B0E-AD89-
1A851084AEAB},ProductName=Darwin Descriptor 1.0,PackageName=Setup.msi,Language=1031,Version=16777216,
Assignment=1,ObsoleteArg=0, ProductIcon=MyIcon.exe,0,,PackageCode={E7135BE9-64A5-468D-A5E3-
EFCBBAB9285F},,,InstanceType=0,LUASetting=0,RemoteURTInstalls=0)
…
MSI (s) (F8:FC) [17:45:45:109]: Executing op: End(Checksum=0,ProgressTotalHDWord=0,ProgressTotalLDWord=190821)
```

WICHTIG: Das Ausführungsmodul kann nicht auf Eigenschaften des Installationspaketes zugreifen, sondern nur Informationen verwenden, die ins Installationsskript eingetragen wurden.

Zur Realisierung des transaktionalen Verhaltens erstellt der Konfigurationsmanager ein weiteres Ausführungsmodul um das Rollback-Skript auszuführen, falls dieses nicht deaktiviert wurde (`DISABLEROLLBACK`). Wird die Installation abgebrochen oder durch einen Fehler beendet, werden die Aktionen des Rollback-Skriptes ausgeführt. Wird die Installation fehlerfrei durchgeführt werden die Commit-Aktionen ausgeführt, die sich ebenfalls im Rollback-Skript befinden.

Anschließend schließt das Installationsmodul die Erstellung des Wiederherstellungspunktes ab, oder macht diesen ungültig. Die Sperrung des Installationsmoduls wird aufgehoben, indem der Schlüssel `InProgress` der Systemregistrierung und die IPI-Datei gelöscht werden. Diese Aktionen werden im Kontext des lokalen Systemkontos ausgeführt. Im Folgenden wird das Installationsmodul terminiert. Die Terminierung wird jedoch aufgeschoben, bis alle benutzerdefinierten Aktionen abgeschlossen sind, die asynchron ausgeführt wurden. Benutzerdefinierte Aktionen, die eine ausführbare Datei verwenden (Typ 2, 18, 50, 51), sind hiervon jedoch ausgenommen.

Der Rückgabewert der Skriptausführung wird ausgewertet, um festzustellen, ob ein Computerneustart erforderlich ist. Die Anforderung für den Neustart wird an den Client übertragen und dem Benutzer in einer Dialogbox angezeigt. Wird die Installation ohne Darstellung einer Benutzeroberfläche ausgeführt, erfolgt der Computerneustart automatisch. Dieses gilt jedoch nur, falls der Neustart nicht durch die Eigenschaft `REBOOT` unterdrückt wurde.

Im Folgenden werden das Installationsskript und alle temporären Kabinettdateien gelöscht. Wurde im Rahmen der Installation ein entsprechendes Produkt deinstalliert, werden diese Daten ebenfalls entfernt. Die Aktionen zum Entfernen der Daten werden auch unter dem lokalen Systemkonto ausgeführt. Wurde festgestellt, dass ein Computerneustart erforderlich ist, empfängt der Server die Anforderung `Reboot` vom Client und führt schließlich den Computerneustart aus.

Skriptdateien

Wie bereits erläutert generiert das serverseitige Installationsmodul ein Installationsskript das im geschützten Ordner *%windir%\installer* abgelegt wird. Dieses Skript enthält Operationsanweisungen, die auf Basis des Installationspaketes, den Eingaben des Benutzers und dem aktuellen Systemstatus generiert wurden. Das Skript enthält ebenfalls den Installationskontext der durch das Installationsmodul ermittelt wurde. Mögliche Typen hierfür sind:

- Benutzerinstallation mit den Rechten des Benutzers (`Per-User-Unmanaged`)
- Benutzerinstallation mit den Rechten des lokalen Systemkontos (`Per-User-Managed`)
- Computerinstallation mit den Rechten des lokalen Systemkontos (`Per-Machine`)

Bei der Initialisierung durch das Installationsmodul wird auf Basis dieser Informationen im Skript festgelegt, ob das Ausführungsmodul im Kontext des aktuellen Benutzers oder des lokalen Systemkontos ausgeführt werden soll. Während der Initialisierung wird ebenfalls ein Rollback-Skript erstellt und im Installationsmodul registriert. Jede Operationsanweisung des Installationsskriptes wird nun ausgeführt. Es werden hierbei einige Aktionen immer mit den Rechten des lokalen Systemkontos ausgeführt, um auf geschützte Bereiche des Systems zugreifen zu können. Hierbei handelt es sich jedoch nur um Aktionen, die der Windows Installer selbst benötigt, um die Registrierung des Produktes vorzunehmen. Der Windows Installer legt beispielsweise eine modifizierte Kopie des Installationspaketes auf dem lokalen System ab. Diese Kopie wird immer im Verzeichnis *%windir%\installer* gespeichert, auf das ein Benutzer ohne

administrative Privilegien keinen Schreibzugriff hat. Wird eine Installation mit den Rechten des lokalen Systemkontos ausgeführt, wird für den Zugriff auf Netzwerkressourcen ein Identitätswechsel vollzogen, da das Systemkonto nur lokale Berechtigungen besitzt.

Das Installationsskript enthält alle Operationsanweisungen die erforderlich sind, um die Produktkonfiguration vorzunehmen. Nachfolgend finden Sie einen Auszug aus dem zur Verfügung stehenden Vorrat an Operationsanweisungen.

HINWEIS: Zur besseren Unterscheidung wird allen Operationsanweisungen der Präfix »ixo« (**I**nstaller E**x**ecution **O**peration) vorangestellt.

Metadaten für die Skriptausführung

Diese Operationsanweisungen sind lediglich für die Skriptausführung notwendig. Eine Aktualisierung des Systems wird hierdurch nicht vorgenommen.

ixoHeader

Die Operationsanweisung `ixoHeader` muss die erste Anweisung im Skript sein. Anhand dieser Anweisung wird die korrekte Version des Ausführungsmoduls geprüft. Das Argument `ScriptType`, legt den Typ des Installationsskriptes fest. Mögliche Werte hierfür sind:

- `Install` (Standardinstallation): `ScriptType = 1`
- `Rollback` (Rollback-Installation): `ScriptType = 2`
- `Advertise` (Ankündigung des Produktes): `ScriptType = 3`
- `PostAdminInstall` (Clientinstallation die von einer administrativen Installation ausgeführt wird): `ScriptType = 4`
- `AdminInstall` (Administrative Installation): `ScriptType = 5`

Das Argument `ScriptAttributes` legt fest, ob die Installation mit Privilegien des aktuellen Benutzers oder des lokalen Systemkontos (Elevated Installation) ausgeführt wird.

OPCode	Argumente
`ixoHeader`	`Signature, Version, Timestamp, LangId, Platform, ScriptType, ScriptMajorVersion, ScriptMinorVersion, ScriptAttributes`

Tabelle 3.3: *Details der Operationsanweisung »ixoHeader«*

ixoProductInfo

Bei der Operationsanweisung `ixoProductInfo` handelt es sich immer um die zweite Anweisung im Installationsskript. Die Anweisung enthält Informationen zu dem Produkt, das installiert werden soll. Die Informationen werden vom Ausführungsmodul benötigt, um den Installationstyp zu ermitteln und hierdurch den Ort festzulegen, an dem die Informationen zur Produktveröffentlichung und zur Registrierung von COM-Komponenten abgelegt werden. Der Installationstyp wird auf Basis des Argumentes `Assignment` festgelegt. Gültige Werte hierfür sind:

- Anwenderbezogene Installation: `Assignment = 0`
- Maschinenbezogene Installation: `Assignment = 1`

Alle weiteren Argumente dieser Operationsanweisung finden Sie in der Tabelle 3.4. Bei den Daten zur Produktveröffentlichung handelt es sich um den Teil der Konfigurationsdaten, die unabhängig vom aktuellen Installationsstatus des Produktes sind.

OPCode	Argumente
`ixoProductInfo`	`ProductKey, ProductName, PackageName, Language, Version, Assignment, ObsoleteArg, ProductIcon, PackageMediaPath, PackageCode, AppCompatDB, AppCompatID, InstanceType`

Tabelle 3.4: *Details der Operationsanweisung »ixoProductInfo«*

Verfügt das Argument `Assignment` über den Wert »1«, wird zunächst geprüft, ob das Produkt für diesen Installationstyp (Computer) bereits auf dem lokalen System angekündigt wurde. Ist dieses nicht zutreffend, wird geprüft, ob der aktuelle Benutzer über administrative Privilegien verfügt, oder ob der Installationsprozess mit erhöhten Rechten ausgeführt wird. Ist eine der Bedingungen zutreffend werden die Daten zur Produktveröffentlichung in der Systemregistrierung unter `HKLM\Software\Classes\Installer` abgelegt. Informationen von COM-Komponenten werden unter `HKLM\Software\Classes` gespeichert. Im Ordner *%windir%\Installer\<Productcode>* werden Symboldateien für Dateiverknüpfungen und alle Arten von Transformationen gespeichert. Ist keine der Bedingungen zutreffend wird der Benutzer informiert, dass seine Berechtigungen für diese Installationsart nicht ausreichend sind und die Installation wird vom Ausführungsmodul abgebrochen.

Wurde das Argument Assignment hingegen auf den Wert »0« festgelegt, wird ebenfalls geprüft, ob das Produkt für diesen Installationstyp (Benutzer) bereits angekündigt wurde. Ist dieses nicht zutreffend, wird geprüft, ob der Installationsprozess mit erhöhten Rechten ausgeführt wird. In diesem Fall wird die Installation Per-User-Managed ausgeführt und die Daten zur Produktveröffentlichung werden unter HKLM\Software\Microsoft\Windows\CurrentVersion\Installer\Managed\<User-SID>\Installer gespeichert. COM-Komponenten werden hierbei unter HKCU\Software\Classes abgelegt. Wird der Installationsprozess hingegen mit den Privilegien des Benutzers ausgeführt, handelt es sich um eine Per-User-Unmanaged Installation. Die Daten zur Produktveröffentlichung werden unter HKCU\Software\Microsoft\Installer und die COM-Informationen unter HKCU\Software\Classes abgelegt. In beiden Fällen werden Symboldateien für Dateiverknüpfungen und für Transformationen, die nicht als sicher erachtet werden, im Ordner *%appdata%\Microsoft\Installer\<Productcode>* gespeichert.

ixoEnd

Die Operationsanweisung ixoEnd markiert das Ende des jeweiligen Skriptes und ist somit die letzte Anweisung im Skript. Diese Operationsanweisung verfügt über keine Argumente.

Aktualisierung der Systemregistrierung

Diese Anweisungen werden verwendet, um die Systemregistrierung zu aktualisieren. Bei der Ausführung des Installationsskriptes wird für jede dieser Anweisungen eine gegensätzliche Anweisung in das Rollback-Skript geschrieben, so dass die Modifikationen zurückgenommen werden können. Die Aktionen im Installationsskript werden entweder im Kontext des Benutzers oder des Systemkontos ausgeführt. Die Zuordnung richtet sich nach dem Installationstyp, der durch ixoHeader und ixoProductInfo definiert wurde. Die erstellten Anweisungen des Rollback-Skriptes werden immer mit den erhöhten Rechten des Systemkontos ausgeführt.

HINWEIS: Zugriffssteuerungslisten (Access Control List, ACL) von vorhandenen Schlüsseln, werden während des Schreibens der Registrierungswerte gespeichert.

Bei einer Installation mit erhöhten Rechten wird kein Identitätswechsel durchgeführt, wenn auf HKEY_CURRENT_USER zugegriffen werden muss. Der Windows Installer-Service lädt den Stamm des aktuellen Benutzers als eigenen HKEY_CURRENT_USER Stamm.

Registrierung der Installationskomponenten

Hierbei handelt es sich um Operationsanweisungen die zur Veröffentlichung des Produktes, der Komponenten und weiteren Ressourcen verwendet werden. Diese Anweisungen werden durch Aktionen generiert, die sich in den Ausführungstabellen befinden. Im Falle einer angekündigten Installation wird ein Advertiseskript erzeugt. Das Advertiseskript enthält ebenfalls Anweisungen zur Veröffentlichung und zur Ankündigung des Produktes. Weiterhin befinden sich in diesem Skript die Anweisungen um Klassen, Dateierweiterungen und Dateiverknüpfungen als Aktivierungspunkte (Entry Point) für die Installation bei Bedarf zu registrieren. Die Anweisungen zur Aktivierung von Klassen und Dateierweiterungen beruhen auf Operationsanweisungen zur Aktualisierung der Systemregistrierung. Aus diesem Grund werden diese Aktionen nicht dem Rollback-Skript angefügt.

Zu den relevanten Operationsanweisungen dieser Kategorie gehören ixoProductRegister, ixoProductUnregister, ixoProductPublish, ixoProductUnpublish, ixoFeaturePublish und ixoComponentPublish.

Aktualisierung von Dateien

Die Operationen zur Aktualisierung von Dateien werden bei einer nicht verwalteten Installation mit den Rechten des aktuellen Benutzers ausgeführt. Bei einer Installation mit erhöhten Rechten werden lokale Dateien mit den Rechten des lokalen Systemkontos aktualisiert. Dateien die sich im Netzwerk befinden werden hingegen mit den Rechten des Benutzers modifiziert.

Kopieren und Entfernen von Dateien

Im Ersten Schritt wird geprüft, ob die zu kopierende Datei bereits auf dem Zielsystem existiert. Ist dies der Fall und werden dieser Datei keine besonderen Zugriffssteuerlisten (ACL) zugewiesen, werden die Zugriffsteuerlisten der existierenden Datei gesichert und später auf die neue Datei angewendet. Ist die Datei auf dem Zielsystem noch nicht vorhanden, wird lediglich die Operationsanweisung zum Entfernen der Datei in das Rollback-Skript übertragen. Sollte die Datei existieren, wird sie vor dem Entfernen für einen eventuellen Rollback gesichert. Hierzu wird zunächst der Ort bestimmt, an dem die gesicherte Datei abgelegt werden soll. Als Erstes wird geprüft ob der Ordner *config.msi* im Stammverzeichnis des Laufwerks erstellt werden kann, auf dem sich die Datei befindet und ob in diesen Ordner geschrieben

werden kann. Ist dieses nicht der Fall wird die Datei unter einem abweichenden Namen im Originalverzeichnis gesichert. Lässt sich der Ordner *config.msi* hingegen erstellen, wird die zu sichernde Datei in diesen Ordner verschoben, wobei die Datei umbenannt wird. Falls beim Verschieben Probleme auftreten, wird zunächst die Datei kopiert und anschließend die existierende gelöscht. Schlägt der Löschvorgang fehl wird das Flag »Neustart ist erforderlich« gesetzt und die Datei zum Löschen nach dem Systemneustart markiert.

Zu den relevanten Operationsanweisungen dieser Kategorie gehören `ixoSetSourceFolder`, `ixoSetTargetFolder`, `ixoChangeMedia`, `ixoFileCopy` und `ixoFolderCreate`.

Patchen von Dateien

Die Aktion zum Patchen von Dateien wird durch die Operationsanweisungen `ixoFileCopy` und `ixoPatchApply` ermöglicht. Während der Kopieraktion wird zunächst geprüft, ob die zu patchende Datei bereits auf dem Zielsystem existiert. Ist dieses der Fall wird geprüft, ob der Patch auf diese Datei angewendet werden kann. Wird festgestellt, dass die zu patchende Datei nicht existiert, ist normalerweise ein Zugriff auf die Originalinstallationsquelle erforderlich, um die Datei zunächst auf das Zielsystem zu kopieren. Allerdings enthält der Windows Installer 3.x Mechanismen um diesen Zugriff zu vermeiden. Die Operationsanweisung `ixoPatchApply` wird nun auf diese Datei angewendet, wobei diese Anweisung eine Referenz auf die Datei und auf den Patch enthält. Die Anzahl der Patches die auf diese Datei angewendet werden sollen, wurde bereits durch `ixoFileCopy` ermittelt und wird nun benötigt, um bestimmte Aktionen für diese Datei zu überspringen. Dieses ist erforderlich falls die Datei bereits in Teilen gepatcht wurde.

Im Folgenden wird der Patch zunächst in einen Sicherungsordner gespeichert und anschließend auf die Datei angewendet. Das Ergebnis, also die gepatchte Datei, wird ebenfalls in dem Sicherungsordner abgelegt. Alle weiteren Patches werden nun auf diese temporäre Datei angewendet. Nachdem alle erforderlichen Patches angewendet wurden, wird die Datei in das eigentliche Zielverzeichnis kopiert.

Ausführung von benutzerdefinierten Code

Die nachfolgend aufgeführten Operationsanweisungen ermöglichen die Ausführung von benutzerdefinierten Code während der Installation. Das Ausführungsmodul stellt hierbei sicher, dass diese Aktionen in einem separaten Prozess ausgeführt werden, der vom Installationskontext abhängig ist. Die Aktionen werden bei einer Installation vom Typ `Per-User-Managed` oder `Per-Machine` im Kontext des lokalen Systems, bei einer Installation vom Typ `Per-User-Unmanaged` im Kontext des aktuellen Benutzers ausgeführt. Der separate Prozess ist erforderlich, um bei einer potentiellen Ausnahme (Exception) im benutzerdefinierten Code, den Installationsprozess nicht zu beeinträchtigen.

OPCode	Beschreibung	Argumente
`ixoCustomActionSchedule`	Benutzerdefinierte Aktion mit verzögerter Ausführung	`Action, ActionType, Source, Target, CustomActionData`
`ixoCustomActionCommit`	Benutzerdefinierte Aktion mit verzögerter Ausführung die nur beim Commit ausgeführt wird	`Action, ActionType, Source, Target, CustomActionData`
`ixoCustomActionRollback`	Benutzerdefinierte Aktion mit verzögerter Ausführung, die nur beim Rollback ausgeführt wird	`Action, ActionType, Source, Target, CustomActionData`

Tabelle 3.5: *Operationsanweisungen zur Ausführung von benutzerdefinierten Code*

Benutzerdefinierte Aktionen mit verzögerter Ausführung werden in so genannten Custom Action-Servern (CA-Server) ausgeführt, die von der Art der benutzerdefinierten Aktion und vom Installationskontext abhängig sind. Nur benutzerdefinierte Aktionen die mit dem Attribut `msidbCustomActionTypeNoImpersonate` markiert wurden, werden bei einer Installation vom Typ `Per-User-Managed` oder `Per-Machine` von dem Custom Action-Server ausgeführt, der im Kontext des lokalen Systemkontos ausgeführt wird. In allen anderen Szenarien wird der benutzerdefinierte Code in dem impersonierten Custom Action-Server ausgeführt. Die Operationsanweisungen `ixoCustomActionCommit` und `ixoCustomActionRollback` werden immer im Rollback-Skript abgelegt. Die Anweisung `ixoCustomActionRollback` wird ausschließlich in Rollback-Szenarien ausgeführt, während `ixoCustomActionCommit` nur ausgeführt wird, wenn die Installation erfolgreich abgeschlossen wurde.

WICHTIG: Dadurch, dass die Operationsanweisung `ixoCustomActionCommit` in das Rollback-Skript eingetragen wird, kann eine Commit Custom Action nicht ausgeführt werden, wenn der Rollback deaktiviert (`DISABLEROLLBACK`) wurde.

Selbstregistrierung von Modulen

Operationen zur Selbstregistrierung von Modulen werden vom Ausführungsmodul als benutzerdefinierte Aktion (`msidbCustomActionTypeExe` + `msidbCustomActionTypeInScript`) implementiert. Die Aktion ist bei einer Installation vom Typ `Per-User-Managed` oder `Per-Machine` mit dem Attribut

msidbCustomActionTypeNoImpersonate versehen, so dass sie mit erhöhten Rechten ausgeführt werden kann. In Szenarien, in denen ein Netzwerkzugriff erforderlich ist, wird über einen speziellen Mechanismus die Registrierung im Kontext des aktuellen Benutzers ermöglicht.

Konfigurationsdaten

Während des Installationsprozesses werden vom Windows Installer vielfältige Konfigurationsinformationen verwendet und angelegt. Diese Informationen beziehen sich nicht nur auf die von Ihnen vorgenommenen Eintragungen in der Windows Installer-Datenbank, sondern auch auf die vom Windows Installer-Service benötigten Einstellungen. Diese Einstellungen ermöglichen beispielsweise das vollständige Deinstallieren der Anwendung oder die automatische Reparatur einer defekten Installation.

Jeder von Ihnen hat bestimmt schon Eintragungen in der Systemregistrierung gefunden, die eine solche »%EmAj?C%k9W7cNB_.[t[Redist_Package>Cd2HfPvUY9]I?,mmJG!9« oder eine ähnliche Darstellungsform besitzen. Die Frage nach dem Aufbau und dem Zweck dieser Darstellung bleibt jedoch vielfach unbeantwortet. An dieser Stelle möchte ich Ihnen nun einen Einblick in das Layout der Konfigurationsdaten des Windows Installers geben. Bitte beachten Sie hierbei, dass die Modifikation dieser Einstellungen ausschließlich durch den Windows Installer-Service, sowie das API oder die Automatisierungsschnittstelle durchgeführt werden sollten. Eine manuelle Modifikation kann Kompatibilitätsprobleme der Anwendungen zur Folge haben.

Darstellungsformen einer GUID

Bei einer GUID *(Global Unique Identifier)* handelt es sich um eine Zeichenfolge, die verwendet wird, um Objekte eindeutig zu bezeichnen. Bekannt geworden sind GUIDs im Rahmen der COM-Technologie (Component Object Model), um solche Objekte eindeutig zu identifizieren. Die Erzeugung einer GUID kann durch Funktionen des Betriebssystems, wie CreateGuid(), CoCreateGuid() oder durch Funktionen der *Base Class Library (BCL)* des *Microsoft .NET Frameworks* erfolgen. Bei der Generierung einer GUID werden Informationen des lokalen Computers verwendet, um die Eindeutigkeit zu gewährleisten. Ein Baustein hierbei ist das Identifizierungsmerkmal einer Netzwerkkarte, das als *MAC-Adresse (Media Access Control Adress)* bezeichnet wird. Weitere Bausteine werden aus dem aktuellen Datum und der aktuellen Uhrzeit gebildet. Zum Generieren von GUIDs existiert eine Vielzahl von Tools. Eines davon ist der *Microsoft GUID Generator (guidgen.exe)*, der mit allen Versionen von *Visual Studio* ausgeliefert wird.

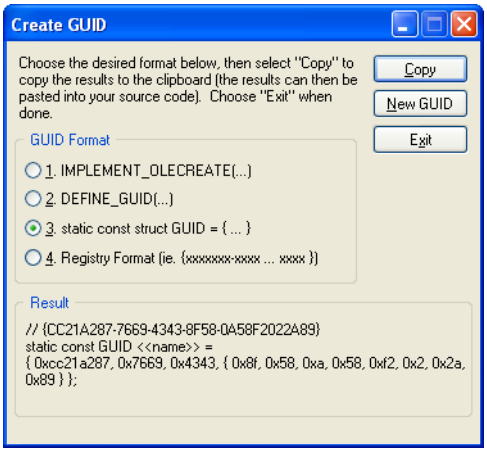

Abbildung 3.8: *Microsoft GUID Generator*

Der *Microsoft GUID Generator* ermöglicht die Erzeugung von GUIDs und die Präsentation in unterschiedlichen Darstellungsformen. Der Windows Installer verwendet unterschiedliche Repräsentationsformen einer GUID, die jedoch von der als »Registry Format« bezeichneten Form abgeleitet werden können.

Standard-GUID

Bei der Standarddarstellung einer GUID handelt es sich um eine aus 38 Zeichen bestehende Zeichenfolge, bei der jedes Zeichen einen hexadezimalen Wert darstellt. Die GUID {012F8BAC-80ea-43fc-BA96-CB6ffbE952A1} wurde mit dem *Microsoft GUID Generator* erzeugt und entspricht augenscheinlich der Standarddarstellung einer GUID für den Windows Installer. Hierbei ist zu beachten, dass der *Microsoft*

GUID Generator sowohl Groß- als auch Kleinbuchstaben zur Repräsentation der GUID verwendet. Der Windows Installer benötigt jedoch zwingend Großbuchstaben, so dass die dargestellte GUID entsprechend geändert werden muss {012F8BAC-80EA-43FC-BA96-CB6FFBE952A1}.

Eine Standard-GUID wird in der Systemregistrierung beispielsweise unter HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall verwendet, um die Produkte zu speichern, die im Dialogfeld *Software* der Systemsteuerung angezeigt werden.

HINWEIS: *Microsoft Visual Studio .NET* enthält ein Tool mit der Bezeichnung *uuidgen.exe*, mit dem Windows Installer konforme GUIDs erzeugt werden können.

Gepackte GUID

Bei einer gepackten GUID handelt es sich um eine aus 32 Zeichen bestehende Darstellung einer GUID, wobei einige Zeichen entfernt und neu geordnet wurden. Nachfolgend finden Sie die bereits oben dargestellte GUID in gepackter Darstellungsform: CAB8F210AE08CF34AB69BCF6BF9E251A

Um eine Standard-GUID {0B533DB3-A248-4E72-B47B-34F9F3342418} in eine GUID in gepackter Darstellungsform umzuwandeln, gehen Sie wie folgt vor:

- Entfernen Sie die öffnende und schließende Klammer: »0B533DB3-A248-4E72-B47B-34F9F3342418«
- Verwenden Sie den ersten Zeichenblock »0B533DB3« in umgekehrter Reihenfolge: »3BD335B0«
- Verwenden Sie den zweiten Zeichenblock »A248« in umgekehrter Reihenfolge: »842A«
- Verwenden Sie den dritten Zeichenblock »4E72« in umgekehrter Reihenfolge: »27E4«
- Verwenden Sie alle weiteren Zeichen »-B47B-34F9F3342418« paarweise in umgekehrter Reihenfolge, wobei Sie die Bindestriche (-) ignorieren: »4BB7439F3F434281«
- Fassen Sie nun alle umgewandelten Zeichenblöcke zusammen, um die gepackte GUID »3BD335B0842A27E44BB7439F3F434281« zu erhalten.

Gepackte GUIDs werden in der Systemregistrierung beispielsweise unter HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Installer\Products verwendet, um Informationen für Produkte abzulegen, die für alle Benutzer installiert wurden.

Komprimierte GUID

Bei einer komprimierten GUID handelt es sich um eine aus 20 Zeichen bestehende Darstellung einer GUID, wobei die Originalzeichen ersetzt werden, wie die Darstellung »7HIH!$RBq9`O-xKW14q[« zeigt. Eine komprimierte GUID wird auch als »Squished GUID (zerquetschte GUID)« oder »SQUID« bezeichnet. Der Algorithmus zum Umwandeln einer SQUID in eine GUID ist nicht dokumentiert. Es existiert jedoch die undokumentierte Windows Installer-Funktion MsiDecomposeDescriptor() mit der Sie diese Umwandlung vornehmen können. Die Funktion wird normalerweise dazu verwendet, eine verschlüsselte Beschreibung der Windows Installer-Komponenten in seine einzelnen Bestandteile zu zerlegen.

Darwin Descriptor

Bei der Installation von Anwendungen werden Informationen auf dem Zielsystem abgelegt, die der Windows Installer benötigt, um die Produktkonfiguration vorzunehmen. Verwenden Sie beispielsweise die Funktion MsiProvideQualifiedComponent() des Windows Installer-API können Sie den Pfad zu einer qualifizierten Windows Installer-Komponente abrufen. Die hierfür benötigten Informationen werden an unterschiedlichen Stellen in der Systemregistrierung, wie HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Installer\Components abgelegt. An dieser Stelle finden Sie für jede qualifizierte Windows Installer-Komponente einen entsprechenden Unterschlüssel, der als gepackte GUID abgebildet wird. Jeder dieser Schlüssel verfügt über einen oder mehrere Werte, die die jeweilige Komponente identifizieren. Dieser Wert wird in verschlüsselter Form dargestellt und enthält Referenzen auf das zugeordnete Produkt, das zugeordnete Feature und die Komponente selbst. Die hier gewählte Darstellungsform wird als Darwin Descriptor bezeichnet.

Eine typische Darstellungsform für den Darwin Descriptor hat folgendes Erscheinungsbild: »v[X,qfU0E?q35RF2Nru?dotNET_Framework_SDK>HDI1hlAB*Av(Q&g3&VT!«. Sie erkennen in der Mitte der Zeichenfolge, den Namen des Features (dotNET_Framework_SDK). Der zugehörige ProductCode und die ComponentId sind durch komprimierte GUIDs dargestellt. Das definitive Format des Darwin Descriptors zur Referenzierung einer Windows Installer-Komponente ist abhängig von der Anzahl der Komponenten und Features im Installationspaket.

Anzahl der Komponenten	Anzahl der Features	Formt des Darwin Descriptors
> 1	> 1	{Komprimierter ProductCode}{Name des Features}>{Komprimierte ComponentId}
> 1	1	{Komprimierter ProductCode}>{Komprimierte ComponentId}

1	> 1	{Komprimierter ProductCode}{Name des Features}<
1	1	{Komprimierter ProductCode}<

Tabelle 3.6: *Aufbau des Darwin Descriptors*

Wie bereits bei der komprimierten GUID erläutert, ist der Algorithmus zum Entschlüsseln einer SQUID und somit auch des Darwin Descriptors nicht dokumentiert. Um den Darwin Descriptor in eine lesbare Form umzuwandeln, kann die undokumentierte Funktion MsiDecomposeDescriptor() der *Msi.dll* verwendet werden, die wie folgt aufgebaut ist.

```
UINT MsiDecomposeDescriptor(
    LPCTSTR szDescriptor,
    LPCSTR szProductCode,     // Ermittelter ProductCode
    LPCSTR szFeatureId,       // Ermittelter Name des Fetures
    LPCSTR szComponentCode,   // Ermittelte ComponentId
    DWORD *pcchArgsOffset     // Offset der Argumente im Descriptor
    );
```

szDescriptor: Der Darwin Descriptor, der entschlüsselt werden soll.

szProductCode: Zeiger auf eine Variable, die den ermittelten ProductCode aufnehmen soll. Der Puffer muss groß genug sein, um 39 Zeichen aufnehmen zu können. Der Parameter kann Null sein.

szFeatureId: Zeiger auf eine Variable, die den ermittelten Namen des Features aufnehmen soll. Der Puffer muss groß genug sein, um 39 Zeichen aufnehmen zu können. Der Parameter kann Null sein.

szComponentCode: Zeiger auf eine Variable, die die ermittelte ComponentId aufnehmen soll. Der Puffer muss groß genug sein, um 39 Zeichen aufnehmen zu können. Der Parameter kann Null sein.

pcchArgsOffset: Zeiger auf eine Variable zur Aufnahme des Offsets bis zum Beginn der Argumente. Einem Darwin Descriptor können zusätzliche Argumente angefügt werden. Ist der Darwin Descriptor beispielsweise 72 Zeichen lang, wird in dieser Variablen der Wert 73 zurückgegeben, da die Argumente an dieser Position beginnen. Dieser Parameter kommt u.a. bei der Verwendung von qualifizierten Komponenten zum Einsatz. Der Parameter kann Null sein.

Wird die Funktion MsiDecomposeDescriptor() erfolgreich ausgeführt, wird ERROR_SUCCESS zurückgegeben. Tritt ein Fehler auf, gibt die Funktion ERROR_INVALID_PARAMETER zurück. Die Funktion kann ebenfalls verwendet werden, um eine SQUID in eine GUID umzuwandeln. Handelt es sich bei der SQUID um den ProductCode eines installierten Produktes wird ERROR_SUCCESS zurückgegeben. Handelt es sich um keinen ProductCode, wird zwar ERROR_INVALID_PARAMETER zurückgegeben, jedoch in der Variablen szProductCode die umgewandelte GUID abgelegt.

Programmtechnische Umsetzung

Die Verwendung der Funktion MsiDecomposeDescriptor() gestaltet sich sehr einfach, um die benötigten Informationen abzurufen. Die folgenden Codefragmente zeigen die Deklaration und die Verwendung dieser Funktion in *Microsoft Visual C# .NET*.

```
[DllImport("msi.dll", CharSet=CharSet.Unicode)]
internal static extern uint MsiDecomposeDescriptor( string Descriptor,
                                                    StringBuilder ProductCode,
                                                    StringBuilder FeatureName,
                                                    StringBuilder ComponentCode,
                                                    ref uint Offset);
```

Listing 3.2: *Deklaration der Funktion »MsiDecomposeDescriptor()«*

Bei der Verwendung der Funktion müssen Objekte vom Typ System.Text.StringBuilder erzeugt werden, die eine ausreichende Kapazität besitzen (39 Zeichen), um die ermittelten Werte aufzunehmen.

```
private static void DecideDescriptor(string descriptor)
{
    // Stringbuilder zur Aufnahme der GUIDs erstellen
    StringBuilder sbProduct = new StringBuilder(39);
    StringBuilder sbFeature = new StringBuilder(39);
    StringBuilder sbComponent = new StringBuilder(39);

    // Offset definieren
    uint offset = 0;

    // Funktion aufrufen
    uint ret = NativeMethods.MsiDecomposeDescriptor(descriptor,
                                                    sbProduct,
                                                    sbFeature,
                                                    sbComponent,
```

```
                                                ref offset);

    // Daten anzeigen
    Console.WriteLine("Darwin Descriptor: {0}", descriptor);
    Console.WriteLine("ProductCode: {0}", sbProduct.ToString());
    Console.WriteLine("Feature: {0}", sbFeature.ToString());
    Console.WriteLine("ComponentId: {0}", sbComponent.ToString());
    Console.WriteLine("Offset: {0}", offset.ToString());
}
```

Listing 3.3: *Darwin Descriptor auswerten*

Undokumentierte Verwendungsmöglichkeiten

Der Darwin Descriptor kann bei vielen API Funktionen und Methoden des Windows Installer-Objektmodells anstelle des Produktcodes übergeben werden. Die Beispielanwendung *DD.exe* ermittelt u.a. auch Informationen zu installierten Produkten, mit Hilfe der Funktion `MsiGetProductInfo()`.

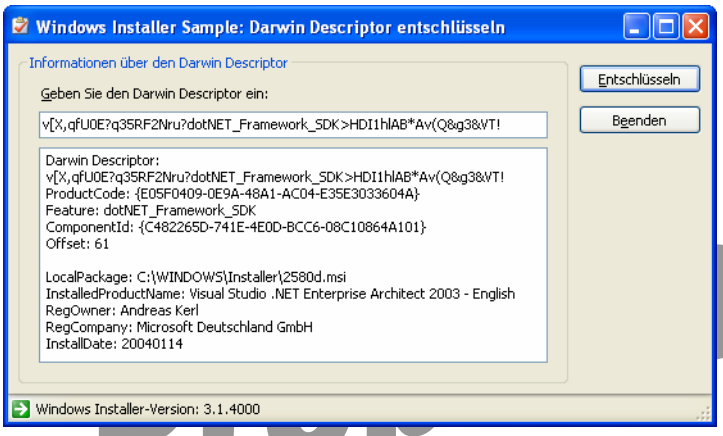

Abbildung 3.9: *Beispielanwendung zum Entschlüsseln des Darwin Descriptors*

Laut Dokumentation sind dieser Funktion der `ProductCode` und der Name der zu ermittelnden Eigenschaft zu übergeben. Als Ergebnis erhält man den tatsächlichen Wert der Eigenschaft. Die Funktion muss zunächst wie folgt deklariert werden:

```
[DllImport("msi.dll", CharSet=CharSet.Unicode)]
internal static extern uint MsiGetProductInfo(string szProduct,
                                               string szProperty,
                                               StringBuilder lpValueBuf,
                                               ref uint pcchValueBuf);
```

Listing 3.4: *Deklaration der Funktion »MsiGetProductInfo()«*

Die Verwendung dieser Funktion ist im folgenden Listing dargestellt. Die Funktion `DecideProductName()` erwartet als Parameter den Namen der zu ermittelnden Eigenschaft und eine Zeichenfolge mit dem Identifizierungsmerkmal des Produktes. In dem Beispiel wird der Darwin Descriptor anstelle des Produktcodes verwendet.

```
private static void DecideProductName(string descriptor, string propertyName)
{
    // Stringbuilder initialisieren
    StringBuilder buf = new StringBuilder(80);
    uint bufSize = (uint) buf.Capacity;

    // Produktinfo abrufen
    uint ret = NativeMethods.MsiGetProductInfo(descriptor,
                                               propertyName,
                                               buf,
                                               ref bufSize);
    if(ret != 0)
    {
        if(ret == NativeMethods.ERROR_MORE_DATA)
        {
            // Falls Buffer zu klein -> Erhöhen
            bufSize++;
            buf = new StringBuilder((int) bufSize);
```

```
            ret = NativeMethods.MsiGetProductInfo(descriptor,
                                                  propertyName,
                                                  buf,
                                                  ref bufSize);
        }
    }

    // Ausgabe
    Console.WriteLine("{0}: {1}", propertyName, buf.ToString() );
}
```

Listing 3.5: *Weitere Produkteigenschaften auswerten*

Die Verwendung des Darwin Descriptors anstelle des Produktcodes funktioniert u.a. auch bei der Methode `Installer.ProductInfo` des Windows Installer-Objektmodells.

Analyse

Das Erstellen eines Installationspaketes und die Installation der Anwendung sind komplexe Vorgänge, die in vielen Fällen Probleme verursachen können. Die Analyse des Installationsprozesses stellt eine effektive Möglichkeit dar, auftretende Probleme zu erkennen und geeignete Lösungsansätze zu entwerfen. Bei der detaillierten Analyse des Installationsprozesses können unterschiedliche Strategien zum Einsatz kommen. Eine Möglichkeit besteht darin, den Ablauf in einem Debugger für Windows Installer-Pakete zu betrachten, um hierbei die kritischen Punkte aufzuspüren. Eine weitere Möglichkeit besteht in der Auswertung des Installationsprotokolls oder des *Windows Ereignisprotokolls*. Diese Möglichkeiten sind ideal geeignet, wenn der Fehlerfall auf einem anderen Computer auftritt, da die erstellten Protokolle einfach zu erstellen und zu übertragen sind. Eine andere Möglichkeit liegt in der Auswertung des tatsächlichen Installationsskriptes. Dieses Verfahren ist für die meisten Einsatzzwecke nur bedingt geeignet, allerdings verschafft es einen tiefen Einblick über die Aktionen des tatsächlichen Installationsprozesses.

HINWEIS: Informationen zum *Windows-Ereignisprotokoll* und den hierauf basierenden Neuerungen im Windows Installer 3.1 finden Sie im Kapitel »Entfernen von Patches«.

Windows Installer-Protokollierung

Die Eintragungen im *Windows-Ereignisprotokoll* sind sehr allgemein gehalten und beinhalten keine Windows Installer spezifischen Daten, die notwendig wären, um eine exakte Fehlerdiagnose oder Problembeschreibung zu erstellen. Der Windows Installer kann aus diesem Grund angewiesen werden, detaillierte Informationen in eine eigene Protokolldatei zu schreiben. Der Umfang dieser Informationen kann durch den Protokollierungsmodus (Logging Mode) individuell festgelegt werden. Nachdem die Protokolldatei erstellt wurde, kann sie mit relativ einfachen Mitteln ausgewertet werden.

Erstellen eines Installationsprotokolls

Sie können die Protokollierung und den Umfang der Informationen auf mehrere Arten bestimmen:

- Durch Verwenden der Befehlszeilenoptionen von *Msiexec.exe*.
- Durch Eintragungen in der Systemregistrierung.
- Durch die Funktion `MsiEnableLog()` oder die Methode `Installer.EnableLog`.

Protokollierung über die Befehlszeilenoption

Bei der Verwendung der Option `/L` wird eine Protokolldatei erstellt. Der Name und der Zielort dieser Datei müssen ebenfalls in der Befehlszeile definiert werden. Die Option `/L` sollten Sie mit weiteren Argumenten erweitern, um den Umfang der zu protokollierenden Aktionen festzulegen, wie in der folgenden Syntax dargestellt wird:

- `msiexec /i {Pfad zum Installationspaket} /l{Protokollargumente} {Protokolldatei}`

Bei den Protokollargumenten handelt es sich um eine Verkettung von einzelnen Zeichen, die letztlich den Umfang des Installationsprotokolls definieren. Tabelle 3.7 stellt die einzelnen Elemente dar, die zur Definition der Protokollargumente verwendet werden können.

Argument	Beschreibung
`i`	Statusmeldungen (***I****nformational status messages*)
`w`	Warnungen (*Nonfatal* ***w****arnings*)
`e`	Alle Fehlermeldungen (*All* ***e****rror messages*)
`a`	Beginn von Aktionen (***A****ction activity*)

r	Aktionsspezifische Daten (***R**ecord data for active action*)
u	Requests des Benutzers (***U**ser requests*)
c	Initialisierungswerte der UI-Argumente (*Initial **c**lient UI parameters*)
m	Out-Of-Memory und fatale Fehlermeldungen (*Out-of-**m**emory or fatal exit information*)
o	Out-Of-Disk-Space Meldungen (***O**ut-of-disk-space messages*)
p	Eigenschaften (***P**roperties*)
v	Detaillierte Informationen (*Additional **v**erbose output*)
x	Zusätzliche Debugginginformationen (*E**x**tra debugging information*)
+	Fügt Daten an existierende Datei an
!	Direktes Schreiben jeder Protokollzeile
*	Protokolliert alle Informationen, mit Ausnahme der detaillierten Informationen und zusätzlichen Debugginginformationen.

Tabelle 3.7: *Befehlszeilenargumente zum Festlagen der Protokollierung*

Mit dem Windows Installer 3.x kann auch die Option `/log` zur Erstellung eines Installationsprotokolls nach dem folgenden Schema verwendet werden.

- `msiexec /i {Pfad zum Installationspaket} /log {Protokolldatei}`

Die Verwendung des Argumentes `/log` ist gleichzusetzen mit dem Protokollargument `/l*`. Der problematische Aspekt bei dieser Syntax ergibt sich daraus, dass ggf. nützliche Informationen für die Fehlersuche nicht ausgegeben werden, da hierbei die Argumente »v« und »x« nicht einbezogen werden.

WICHTIG: Für detaillierte Analysen des Installationsprotokolls sollten Sie dieses unter Verwendung des Argumentes `/l*vx` erstellen lassen. Alle in diesem Buch abgedruckten Installationsprotokolle und Protokollauszüge wurden unter Verwendung dieses Argumentes erstellt.

Verwenden der Systemrichtlinie

Der Mechanismus zum Aktivieren der Protokollfunktion über die Befehlszeile ist sehr flexibel und für jede Installation individuell konfigurierbar. Bei Installationsszenarien, in denen kein Aufruf über die Befehlszeile möglich ist, wie z.B. *On-Demand-Installation* oder *Self-Repair-Installation*, kann dieser Mechanismus jedoch nicht verwendet werden. Um auch in solchen Anwendungsfällen auf die Protokollierung nicht verzichten zu müssen, steht die Systemrichtlinie `Logging` zur Verfügung.

- ***Lage:*** `HKLM\Software\Policies\Microsoft\Windows\Installer\Logging`
- ***Typ:*** `REG_SZ`
- ***Werte:*** Zeichenfolge zur Darstellung der Protokollargumente
- ***Beschreibung:*** Der hier eingetragene Wert kann aus den in der vorherigen Tabelle genannten Argumenten konstruiert werden. Zur Protokollierung aller Aktionen und zusätzlicher Ausgabe der detaillierten- und Debugging-Informationen, ist die Zeichenfolge »iwearucmpvox« zu verwenden, da das Zeichen »*« hierbei nicht unterstützt wird. Sie haben in diesem Szenario keine Möglichkeit eine Protokolldatei festzulegen. Eine Protokolldatei mit der Bezeichnung *msixxxxx.log* wird automatisch im Verzeichnis für temporäre Dateien erstellt, wobei »xxxxx« durch eine zufällige Zeichenfolge ersetzt wird.

HINWEIS: Die Erstellung eines Protokolls über die Befehlszeilenoptionen wird vorrangig behandelt. Werden keine Argumente dort angegeben, wird ein Protokoll im Verzeichnis für temporäre Dateien erstellt. Werden hingegen Argumente übergeben, wird das in der Befehlszeile angegebene Protokoll erstellt.

Verwenden der Funktion »MsiEnableLog()«

Der Microsoft Windows Installer verfügt über eine Schnittstelle, um programmtechnisch auf diverse Funktionen zuzugreifen. Zur Erstellung und zur Festlegung des Umfangs eines Installationsprotokolls steht hierbei die Funktion `MsiEnableLog()` zur Verfügung. In dem in Listing 3.6 dargestellten Beispiel wird die Installation gestartet, die Darstellung der Benutzeroberfläche konfiguriert, die Erstellung des Installationsprotokolls aktiviert und der Abschluss der Installation ausgewertet.

```
public static void InstallProduct(string packagePath, string commandLine, string logFile, IntPtr Handle)
{
    // Vollständige Benutzeroberfläche
    const uint uiLevel = 5;

    // Alles protokollieren incl. Verbose und Debugging
    const uint logMode = 16351;

    // Darstellung der Benutzeroberfläche konfigurieren
    MsiSetInternalUI(uiLevel, ref Handle );
```

```
    // Umfang der Protokollierung festlegen
    MsiEnableLog(logMode, logFile, 0);

    // Produkt installieren
    uint ret = MsiInstallProduct(packagePath, commandLine );

    if(ret != 0)
    {
        // Fehler aufgetreten. Fehlermeldung aus Return-Code generieren
        throw new InstallerException(ret);
    }
}
```

Listing 3.6: *Starten und Protokollieren einer Installation*

Ausgabe von Debugginginformationen

Der Windows Installer kann angewiesen werden, die Informationen des Installationsprozesses über die Funktion `OutputDebugString()` auszugeben. Diese Informationen können dadurch in Echtzeit mit einem Debugger betrachtet werden. Hierzu ist es erforderlich die Systemrichtlinie `Debug` zu setzen.

- ***Lage:*** `HKLM\Software\Policies\Microsoft\Windows\Installer\Debug`
- ***Typ:*** `REG_DWORD`
- ***Werte:*** »1«, »2« oder »4«, sowie eine Kombination dieser Werte
- ***Beschreibung:*** Wird der Wert auf »1« gesetzt, werden allgemeine Debugginginformationen ausgegeben. Bei einem Wert von »2« werden, werden alle verfügbaren Informationen angezeigt. Der Wert »4« steht erst seit dem Windows Installer 2.0 zur Verfügung. Hierdurch werden die Parameter der Befehlszeile dem Installationsprotokoll angefügt. Um sämtliche Informationen im Debugger auszugeben, und die Befehlszeile dem Protokoll anzufügen, ist diese Richtlinie auf den Wert »7« zu setzen.

Analyse des Installationsprotokolls

Windows Installer-Protokolle stellen die effektivste Methode dar, alle Informationen des Installationsprozesses auszuwerten. Folgende Informationen sind in einem solchen Protokoll vorhanden:

- Fehler die während der Installation auftreten, einschließlich aller Windows Installer-Meldungen die einen Benutzerdialog erzeugen.
- Die benutzerdefinierten Aktionen, die ausgeführt werden.
- Ist ein Neustart des Systems erforderlich oder wurde er bereits durchgeführt.
- Abschließende Werte der Windows Installer-Eigenschaften.
- Die Position der Installationsquellen.
- Ob die Installation durch den Benutzer abgebrochen wurde.
- Wo eine Installation fehlgeschlagen ist.
- Ob eine fehlerhafte Installation zurückgesetzt (`Rollback`) wurde.

Wie bereits in diesem Kapitel verdeutlicht, wird unter Betriebssystemen die auf der Windows NT Technologie basieren, die Installation durch zwei Prozesse realisiert. Das Installationsprotokoll stellt die Informationen beider Prozesse dar. Eine Installation, die unter dem Betriebssystem *Microsoft Windows 9x* ausgeführt wird, verwendet lediglich den Client-Prozess. Eine Aktion, die innerhalb des Client-Prozesses ausgeführt wird, ist im Installationsprotokoll durch »(c)« gekennzeichnet.

```
MSI (c) (3C:58) [10:56:21:828]: Machine policy value 'DisableUserInstalls' is 0
```

Eine Aktion des Server-Prozesses ist durch »(s)« gekennzeichnet.

```
MSI (s) (84:64) [10:56:25:437]: User policy value 'TransformsAtSource' is 0
```

Der generelle Aufbau einer Eintragung im Installationsprotokoll beginnt immer mit dem Präfix »MSI«, der gefolgt wird von der Kennzeichnung des Prozesses »(c)« oder »(s)«. Im Anschluss werden die IDs des Prozesses und des Threads angefügt, von denen diese Eintragung erzeugt wurde. Bei der Eintragung des Client-Prozesses aus dem Beispiel, repräsentiert »3C« den Prozess und »58« den Thread. Hierbei ist allerdings zu beachten, dass nur die letzten beiden Stellen der IDs (`Hex`) dargestellt werden. Im Folgenden wird für jede Aktion ein Zeitstempel (`TimeStamp`) angefügt, der aus den Elementen »`[Stunde:Minute:Sekunde:Ticks]`« zusammengesetzt ist.

WICHTIG: Ein Zeitstempel wird nur mit dem Windows Installer der Version 3.0 und höher erzeugt.

Der Doppelpunkt (:) trennt die Metadaten von der eigentlichen Beschreibung der Aktion. Alle Werte die dem Doppelpunkt folgen, beschreiben die durchgeführte Aktion. Der Aufbau ist immer von der jeweiligen Aktion abhängig.

Die Protokollierung beginnt zunächst mit allgemeinen Informationen über das Datum der Installation, der Version des Windows Installers, eine Referenz auf das zu installierende Produkt und zusätzliche Argumente, die der Befehlszeile übergeben wurden.

```
=== Verbose logging started: 18.03.2005  10:56:21  Build type: SHIP UNICODE 3.01.4000.1823  Calling process:
C:\WINDOWS\system32\msiexec.exe ===
MSI (c) (5C:E4) [10:56:21:604]: Resetting cached policy values
MSI (c) (5C:E4) [10:56:21:604]: Machine policy value 'Debug' is 7
MSI (c) (5C:E4) [10:56:21:604]: ******* RunEngine:
******* Product: Setup.msi
******* Action:
******* CommandLine:  TRANSFORMS="Inst.mst" MSINEWINSTANCE=1
```

Dem Installationsprotokoll werden anschließend die Aktionen des Client-Prozesses angefügt. Hierbei handelt es sich hauptsächlich um Informationen, die die Interaktion des Benutzers mit dem Windows Installer beschreiben.

```
MSI (c) (3C:58) [10:56:21:859]: PROPERTY CHANGE: Adding USERNAME property. Its value is 'Andreas Kerl'.
```

Nachdem die Benutzerinformationen gesammelt wurden, wird die Ausführung an den Server-Prozess übergeben. Dieses wird durch folgende Zeile dargestellt:

```
MSI (c) (3C:58) [10:56:25:421]: Switching to server: COMPANYNAME="Microsoft Deutschland GmbH" TARGETDIR="F:\"
USERNAME="Andreas Kerl" INSTALLDIR="C:\Program Files\Windows Installer Spy\" GAC="F:\" WWWROOT="F:\"
SOURCEDIR="E:\" CS="C:\Program Files\Windows Installer Spy\Sourcen\CS\" VB="C:\Program Files\Windows Installer
Spy\Sourcen\VB\" CLIENTPROCESSID="1340" CLIENTUILEVEL="0" CURRENTDIRECTORY="E:\" ACTION="INSTALL"
EXECUTEACTION="INSTALL" ROOTDRIVE="F:\" SECONDSEQUENCE="1"  ADDLOCAL=Source,Complete,SourceVB,SourceCS
```


Der Wechsel zum Server-Prozess wird durch die Zeichenfolge »Switching to server« markiert. Dieser Zeichenfolge werden alle öffentlichen Eigenschaften angefügt, die durch die Aktionen des Client-Prozesses verändert wurden. Wird eine Installation mit erhöhten Rechten durchgeführt, werden nur die als sicher erachteten öffentlichen Eigenschaften (`SecureCustomProperties`) übergeben. In dem Server-Prozess werden die eigentlichen Installationsprozeduren, wie das Kopieren von Dateien und das Schreiben von Einträgen in die Systemregistrierung durchgeführt. Als nächstes werden die serverseitigen Initialisierungen angezeigt, bevor die eigentlichen Installationsaktionen ausgeführt werden. Der Aktion `InstallValidate` folgt ein Dump, in dem die benötigten Statusinformationen der Features und Komponenten zusammengefasst werden.

```
Action start 10:56:25: InstallValidate.
MSI (s) (84:64) [10:56:25:515]: Feature: Source; Installed: Absent; Request: Local; Action: Local
MSI (s) (84:64) [10:56:25:515]: Feature: Complete; Installed: Absent; Request: Local; Action: Local
MSI (s) (84:64) [10:56:25:515]: Feature: SourceVB; Installed: Absent; Request: Local; Action: Local
MSI (s) (84:64) [10:56:25:515]: Feature: SourceCS; Installed: Absent; Request: Local; Action: Local
MSI (s) (84:64) [10:56:25:515]: Component: WINDOW1.DLL; Installed: Absent; Request: Local; Action: Local
MSI (s) (84:64) [10:56:25:515]: Component: VB.ico; Installed: Absent;   Request: Local;   Action: Local
```

Die eigentlichen Installationsaktionen, wie das Kopieren von Dateien, werden immer durch mehrere Einträge dargestellt. Wie bereits an vorheriger Stelle in diesem Kapitel erläutert, werden Aktionen die den Zustand des Systems verändern, in das Installationsskript eingetragen. Der Beginn der Skripterstellung ist durch die folgende Zeile dargestellt.

```
Action 10:56:30: GenerateScript. Generating script operations for action:
```

Im Anschluss werden für jede Aktion, die im Rahmen der Skriptausführung verwendet werden soll, entsprechende Eintragungen angefügt.

```
MSI (s) (84:64) [10:56:30:453]: Doing action: InstallFiles
Action start 10:56:30: InstallFiles.
GenerateScript: Copying new files
Action ended 10:56:30: InstallFiles. Return value 1.
```

Wenn Aktionen ordnungsgemäß beendet wurden, wird für diese Aktion der Wert »1« zurückgegeben. Die folgende Tabelle zeigt Ihnen eine Auflistung aller möglichen Rückgabewerte von Aktionen.

Wert	Error Code	Beschreibung
0	ERROR_FUNCTION_NOT_CALLED	Aktion kann nicht ausgeführt werden.
1	ERROR_SUCCESS	Aktion ordnungsgemäß ausgeführt.
2	ERROR_INSTALL_USEREXIT	Benutzer hat die Installation abgebrochen.

3	ERROR_INSTALL_FAILURE	Nicht reproduzierbarer Fehler.
4	ERROR_INSTALL_SUSPEND	Installation vorübergehend angehalten. Installation wird später fortgesetzt.

Tabelle 3.8: *Rückgabewerte von Aktionen*

Nachdem die Erstellung des Installationsskriptes abgeschlossen ist, wird das Ausführungsmodul erzeugt und die Skriptausführung gestartet.

```
MSI (s) (84:64) [10:56:30:546]: Running Script: C:\WINDOWS\Installer\MSI14.tmp
```

Es werden nun die Aktionen des Installationsskriptes ausgeführt. Wie bereits erwähnt, wurden diese als so genannte Operationsanweisungen in das Skript eingetragen. Das Ausführen der Operationsanweisungen ist im Installationsprotokoll durch die Zeichenfolge »Executing op:« markiert.

```
MSI (s) (84:64) [10:56:30:546]: Executing op: Header(Signature=1397708873,Version=300,
Timestamp=814372624,LangId=1033,Platform=0,ScriptType=1,ScriptMajorVersion=21,
ScriptMinorVersion=4,ScriptAttributes=0)
MSI (s) (84:64) [10:56:30:546]: Executing op: ProductInfo(ProductKey={FA52CE07-1CD8-4F99-A357-
17F6447D7736},ProductName=Windows Installer Spy,PackageName=Setup.MSI,Language=1033,Version=16777216,
Assignment=0,ObsoleteArg=0,,,PackageCode={5FCC6A65-FCA6-4F4F-81DD-38DB9162CAF8},,,InstanceType=0, LUASetting=0)
```

Sie erkennen dass diese Eintragungen identisch sind, mit der Signatur der Operationsanweisungen `ixoHeader` und `ixoProductInfo`. Nachdem zunächst diese Informationen ausgewertet wurden, werden die Modifikationen am Zielsystem vorgenommen.

```
MSI (s) (84:64) [10:56:30:890]: Executing op: SetTargetFolder(Folder=C:\Program Files\Windows Installer Spy\)
MSI (s) (84:64) [10:56:30:890]: Executing op: SetSourceFolder(Folder=1\PROGRA~1\WINDOW~1\|Program Files\Windows
Installer Spy\)
MSI (s) (84:64) [10:56:30:953]: Executing op:
FileCopy(SourceName=MSISpy.exe,SourceCabKey=MSISpy.exe,DestName=MSISpy.exe,Attributes=0,FileSize=28672,PerTick=
32768,,VerifyMedia=1,,,,,CheckCRC=0,Version=1.0.1168.24222,Language=0,InstallMode=58982400,,,,,,,)
```

Die Skriptausführung wird mit der Operationsanweisung `ixoEnd` abgeschlossen, die im Installationsprotokoll wie folgt dargestellt wird:

```
MSI (s) (84:64) [10:56:31:171]: Executing op: End(Checksum=0,ProgressTotal=675236)
```

Dem Installationsprotokoll wird ein Dump aller Eigenschaften des Installationsprozesses angefügt. Hierbei sind zunächst die Eigenschaften des Server-Prozesses und im Anschluss die Eigenschaften des Client-Prozesses aufgeführt.

HINWEIS: Windows Installer 3.x protokolliert darüber hinaus jede Änderung an den Eigenschaftswerten im Installationsprozess. Diese Eintragungen sind durch den Präfix »PROPERTY CHANGE« gekennzeichnet und enthalten sowohl den alten als auch den neuen Wert der Eigenschaft.

In Abbildung 3.10 sind die Aktionsgruppen des Installationsprozesses aufgeführt, wie sie im Installationsprotokoll dargestellt werden. Die grauen Elemente in diesem Diagramm stellen die Aktionen des Client-Prozesses; die weißen Elemente stellen die Aktionen des Server-Prozesses dar.

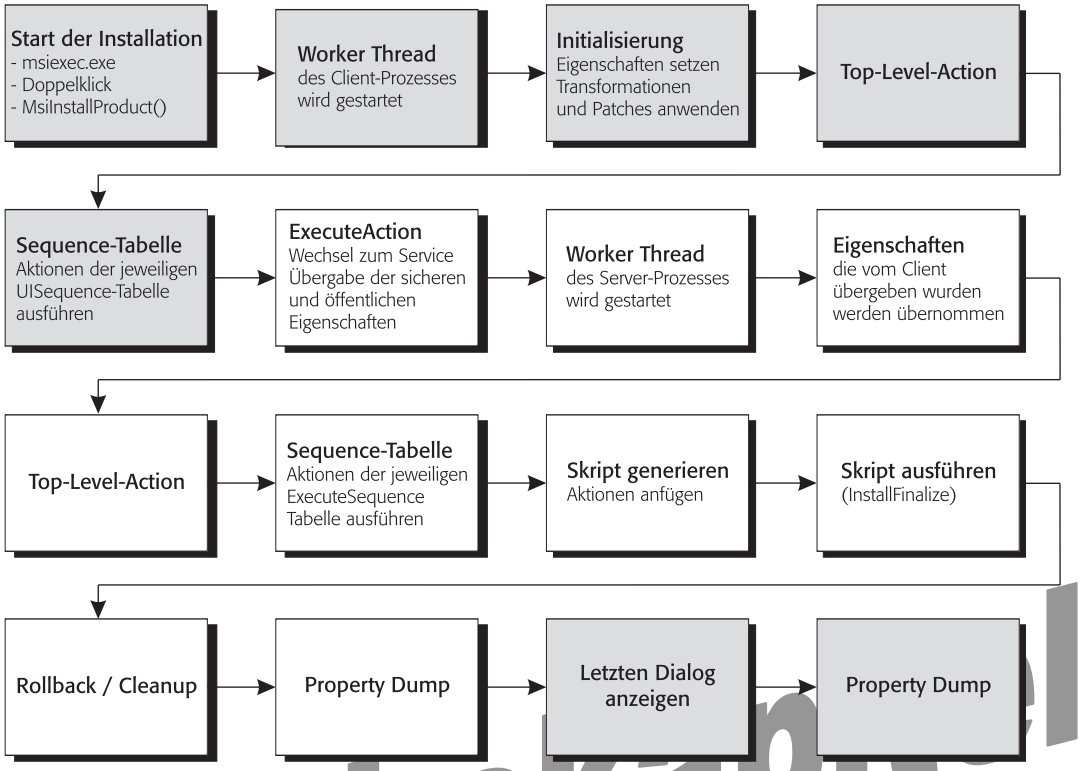

Abbildung 3.10: *Schematische Darstellung im Installationsprotokoll*

HINWEIS: Die beschriebenen Darstellungsformen im Installationsprotokoll werden in dieser Vollständigkeit nur angezeigt, wenn die detaillierte Ausgabeform (Verbose) gewählt wurde.

Strategien für die Fehlersuche

Die Fehlersuche ist ein äußerst komplexer Prozess, der eine große Erfahrung und die hierfür notwendigen Detailkenntnisse voraussetzt. Der Windows Installer informiert Sie durch geeignete Maßnahmen über das Auftreten eines Fehlers. Wird eine Installation mit einer Benutzeroberfläche ausgeführt und tritt während des Installationsprozesses ein Fehler auf, wird Ihnen eine Fehlermeldung angezeigt. Haben Sie die Protokollierung aktiviert und lassen Sie sich hierbei eine detaillierte Aufzeichnung anfertigen, stehen Ihnen zusätzliche Informationen zur Verfügung. Das Windows Installer-Protokoll ist das wichtigste Instrument, um auftretende Probleme wirkungsvoll zu beseitigen.

Handelt es sich bei dem auftretenden Fehler um einen »Internal Error« finden Sie im *Windows Installer-SDK* Informationen zu dem jeweiligen Fehlercode. Interne Fehler resultieren häufig von Problemen im Windows Installer-Paket oder im Windows Installer-Dienst.

Fehlermeldungen

Öffnen Sie das entsprechende Installationsprotokoll und suchen Sie nach der Zeichenfolge »Error«. Folgt auf diese gefundene Zeichenfolge eine Nummer, können Sie eine Erläuterung zu dem aufgetretenen Fehler in der Dokumentation des *Windows Installer-SDK* unter dem Abschnitt »Microsoft Windows Installer Error Messages« finden.

```
MSI (c) (3C:58) [10:56:22:937]: Doing action: Welcome_Dialog
Action start 10:56:22: Welcome_Dialog. Dialog created.
Error 2803: Dialog View did not find a record for the dialog User_Information_Dialog
Error 2835: The control ErrorIcon was not found on dialog ErrorDialog
Internal Error 2835. ErrorIcon, ErrorDialog
```

Berücksichtigen Sie auch, dass der Windows Installer mitunter nicht alle Fehlermeldungen protokolliert, so dass Sie unter Umständen eine andere Taktik verwenden müssen. Die Suche nach ungewöhnlichen Unterbrechungen in der zeitlichen Abfolge der Aktionen kann Ihnen hierbei helfen. Mit Verwendung des Windows Installers 3.x wird zu diesem Zweck ein Zeitstempel jeder Eintragung vorangestellt. Haben Sie einen Fehler identifiziert, sollten Sie die Vorgänge vor dem Fehler betrachten und analysieren.

Verwenden Sie, wenn möglich immer ein detailliertes Installationsprotokoll, da hier viele ausführliche Informationen zu finden sind.

```
MSI (s) (84:64) [10:56:30:953]: File: C:\Windows\inf\agtinst.inf; Overwrite; existing file is unversioned and
unmodified
```

Installationsquellen

In einigen Anwendungsfällen ist es erforderlich, dass der Windows Installer auf die Originalinstallationsquellen zugreifen muss, um bestimmte Ressourcen zu extrahieren. Wird die Installation ohne Anzeige einer Benutzeroberfläche durchgeführt und kann auf die Installationsquelle nicht zugegriffen werden, wird die Installation beendet. Das Installationsprotokoll enthält jedoch Hinweise für den Zugriff auf die Installationsquellen nach dem folgenden Schema.

```
MSI (s) (84:64) [10:56:32:171]: SOURCEMGMT: Attempting to use LastUsedSource from source list.
MSI (s) (84:64) [10:56:32:171]: SOURCEMGMT: Trying source \\eagle\setup\.
MSI (s) (84:64) [10:56:32:187]: SOURCEMGMT: Source is invalid due to missing/inaccessible package.
MSI (s) (84:64) [10:56:32:187]: SOURCEMGMT: Processing net source list.
MSI (s) (84:64) [10:56:32:187]: SOURCEMGMT: Trying source \\eagle\setup\1.0.0.
MSI (s) (84:64) [10:56:32:203]: SOURCEMGMT: Source is invalid due to missing/inaccessible package.
```

Alle Informationen, die auf die Installationsquellen abzielen, sind durch die Zeichenfolge »SOURCEMGMT« im Installationsprotokoll gekennzeichnet.

Ressourcen werden nicht installiert

Vielfach stellt sich die Frage warum eine bestimmte Ressource nicht installiert wurde. Auch hierfür ist die Analyse nicht sonderlich schwierig, da das Installationsprotokoll die erforderlichen Informationen bereitstellt. Der erste Schritt besteht darin, den Installationserfolg der Komponente zu prüfen, die die jeweilige Ressource beinhaltet.

```
MSI (s) (84:64) [10:56:34:546]: Disallowing installation of component: {10048711-2C96-11D2-9A97-006097C4E452}
since the same component with higher versioned keyfile exists
```

Ist ein solcher Eintrag im Installationsprotokoll vorhanden, kann die Begründung für das Verhalten sehr einfach abgelesen werden. Findet sich im Installationsprotokoll kein Hinweis auf die jeweilige Komponente muss zunächst geprüft werden, ob die Komponente im Installationspaket mit einer Bedingung versehen ist, die gegen False ausgewertet wurde. Ist dieses nicht der Fall, sollte eine Analyse auf Ebene der Datei erfolgen, da entsprechende Dateiaktionen dem Protokoll ebenfalls angefügt werden.


```
MSI (s) (84:64) [10:56:34:546]: Executing op:
FileCopy(SourceName=Common.dll,SourceCabKey=Common.dll,DestName=Common.dll,Attributes=0,FileSize=16384,PerTick=
32768,,VerifyMedia=1,,,,,CheckCRC=0,Version=1.1.1.0,Language=0,InstallMode=58982400,,,,,,,)
MSI (s) (84:64) [10:56:34:546]: File: C:\Program Files\Darwin Descriptor 1.0\Common.dll;       Won't
Overwrite; Won't patch; Existing file is of an equal version
```

Rollbacks

Wie bereits in vorherigen Kapiteln beschrieben, werden Windows Installer basierte Installationen transaktional ausgeführt, wobei ein Rollback durchgeführt wird, falls die Installation nicht ordnungsgemäß abgeschlossen wurde. Suchen Sie nach der Zeichenfolge »rollback« in dem Installationsprotokoll. Ignorieren Sie jedoch die Zeilen mit den Inhalten »removing rollback files« und »rollback files are removed at the end of a successfully setup«.

Das folgende Beispiel ist aus einem Protokoll einer unvollständigen Installation:

```
Internal Error 2355. C:\Program Files\Common Files\system\Mapi\1033\essec32.dll, -2147024865
Action ended 10:56:30: InstallFinalize. Return value 3.
Aktion 10:56:34: Rollback. Rolling back action: Rollback:
```

Offenbar resultiert der Rollback aus einem Fehler mit der Datei *exsec32.dll*. Sie finden entsprechende Hinweise auf die Fehlernummer (Internal Error 2355) im *Windows Installer-SDK*.

Erfolg der Installation

Wird die Installation nicht fehlerfrei beendet, so stellt sich immer die Frage nach dem Grund für dieses Verhalten. Zur Analyse des Problems kann das Installationsprotokoll nach unterschiedlichen Informationen durchsucht werden.

- *InstallFinalize: Return:* Kennzeichnet das Ende des Installation. Der letzte Fehler ist der Grund für den Abbruch (Return Code »3« bedeutet einen Fehler).
- *Executing op: Header (Letzte Instanz):* Dieser Eintrag wird zu Beginn der Skriptausführung erstellt. Die letzte Instanz dieser Eintragung kennzeichnet das Rollbackskript. Die Aktion die zum Rollback führte, befindet sich kurz vor diesem Eintrag.

Sie haben sicherlich erkannt, dass die Analyse der Fehlerprotokolle möglich aber auch sehr aufwendig ist und ein hohes Maß an Sachverständnis erfordert. Bei komplexen Installationsszenarien kommt ein

beträchtlicher Umfang des Installationsprotokolls hinzu. Eine Analyse mit *Notepad* oder einem anderen Texteditor erscheint ausgesprochen schwierig und resultiert in vielen Fällen in zusätzlichen Fehlerquellen.

Das *Windows Installer-SDK* enthält für solche Fälle ein hervorragendes Tool mit der Bezeichnung *Windows Installer Verbose Log Analyzer (wilogutl.exe)*. Diese Anwendung ist unentbehrlich bei der Analyse von Protokolldateien einer Windows Installer-Installation. Es werden ausschließlich kritische Fehler angezeigt und eine Lösung zur Abhilfe des Problems angeboten.

Analyse der Skriptdateien

Eine interessante Alternative zur Auswertung des Installationsprotokolls bietet die Analyse der Skriptdateien. Der Informationsgehalt dieser Möglichkeit ist nicht so umfangreich wie beim Installationsprotokoll, vielmehr werden hierbei die wesentlichen Faktoren des Installationsprozesses, nämlich die Modifikation des Zielsystems, berücksichtigt. Die Analyse einer Skriptdatei ist hingegen wesentlich aufwendiger, da folgende Problemfaktoren zu berücksichtigen sind:

- Die Skriptdatei wird dynamisch erzeugt, mit Daten gefüllt und nach Fertigstellung des Installationsprozesses automatisch entfernt.
- Die Skriptdatei liegt in einem Binärformat vor, so dass diese mit Standardanwendungen nicht betrachtet werden kann.

Um eine Skriptdatei trotzdem analysieren zu können, müssen einige Voraussetzungen geschaffen werden.

Sichern der Skriptdatei

Die Skriptdatei wird automatisch erstellt, wenn in der jeweiligen Ausführungstabelle (`AdminExecuteSequence`, `InstallExecuteSequence`, `AdvtExecuteSequence`) die Aktion `InstallInitalize` aufgerufen wird. Alle Aktionen zwischen `InstallInitalize` und `InstallFinalize` tragen ihre Operationsanweisungen in das Skript ein. Bei `InstallFinalize` wird das Ausführungsmodul gestartet und die Anweisungen der Skriptdatei ausgeführt. Die Skriptdatei wird im Anschluss automatisch gelöscht. Um die Skriptdatei vor dem Löschvorgang zu sichern sind zwei Möglichkeiten anwendbar.

Modifikation des Windows Installer-Paketes

Die Skriptdatei der Installation wird immer im Ordner *%windir%\installer* abgelegt. Der Name setzt sich aus dem Präfix »MSI«, einer generierten Zeichenfolge und der Dateierweiterung `.tmp` zusammen. Um eine solche Datei zu sichern und später eine Analyse durchführen zu können, muss der Installationsprozess angehalten werden, nachdem die Skriptdatei erstellt wurde und mit allen Anweisungen gefüllt ist.

Eine Möglichkeit besteht darin eine benutzerdefinierte Aktion der Ausführungstabelle hinzuzufügen, die zur Unterbrechung des Installationsprozesses führt. Hierbei ist es unerheblich, ob der Installationsprozess im Weiteren fortgesetzt wird, denn zur Analyse muss lediglich die Skriptdatei vollständig erstellt worden sein. Es empfiehlt sich eine benutzerdefinierte Aktion zu erstellen, die einen modalen Dialog aufruft.

HINWEIS: Diese Vorgehensweise entspricht nicht den Richtlinien zur Erstellung eines Windows Installer-Paketes. Wie bereits erläutert dürfen keine Oberflächenelemente vom Server-Prozess aufgerufen werden, da es hierbei zu extremen Problemen kommen kann. Da dieses Szenario jedoch nur zu Testzwecken verwendet wird, sollte einer solchen Implementierung nichts im Wege stehen.

Erstellen Sie einen Datensatz in der Tabelle `CustomAction`, der die nachfolgenden Werte enthält:

- **`Name`**: `StopProcess`
- **`Type`**: 1062 (`msidbCustomActionTypeVBScript` + `msidbCustomActionTypeInScript`)
- **`Source`**: `null`
- **`Target`**: `Msgbox „Installation wird angehalten"`

Der Typ »1062« legt hierbei fest, dass ein VBScript-Befehl aufgerufen wird, der in der Spalte `Target` definiert wurde. Da diese Aktion im Rahmen der Skriptausführung verwendet werden soll, muss das Attribut `msidbCustomActionTypeInScript` verwendet werden. Detaillierte Informationen zu benutzerdefinierten Aktionen finden Sie in einem späteren Kapitel.

Im Folgenden muss die benutzerdefinierte Aktion noch in den Installationsprozess eingebunden werden. Je nachdem welches Skript sie betrachten möchten, müssen Sie die Zuordnung in den Tabelle `InstallExecuteSequence`, `AdminExecuteSequence` oder `AdvtExecuteSequence` vornehmen. Fügen Sie die benutzerdefinierte Aktion unmittelbar vor `InstallFinalize` ein.

Im Laufe des Installationsprozesses wird nun ein Dialog angezeigt. Wechseln Sie in das Verzeichnis *%windir%\installer* und sichern Sie die entsprechende Skriptdatei. Das Rollback-Skript befindet sich im Ordner *config.msi* des Stammlaufwerkes. Sichern Sie die Datei mit der Dateierweiterung `.rbs`.

Überwachen von Verzeichnissen

Eine andere Möglichkeit zur Sicherung der Skriptdateien bietet die Überwachung der entsprechenden Verzeichnisse. Hierzu stellt das Microsoft .NET Framework die Komponente `System.IO.FileSystemWatcher` zur Verfügung. Überwachen Sie die notwendigen Verzeichnisse und kopieren Sie die jeweiligen Dateien bei jeder Änderung in ein entsprechendes Sicherungsverzeichnis.

Spezialfall »AdvertiseSkript«

Ein AdvertiseSkript wird benötigt um ein Produkt auf einem System anzukündigen. Die Sicherung eines solchen Skriptes lässt sich auch mit den vorgestellten Möglichkeiten erreichen. Allerdings gibt es für solche Skripte spezielle Implementierungen in der Programmierschnittstelle des Windows Installers. Im nachfolgenden Listing wird die Verwendung der Funktion `MsiAdvertiseProductEx()` zu Generierung eines Skripts zur Produktankündigung aufgezeigt.

```
public static void GenerateAdvertiseScript(string packagePath, string scriptFilePath, string transforms,
                                            ushort locale, uint platform, uint instance)
{
    // Funktion zum Erstellen des Skriptes aufrufen
    uint ret = MsiAdvertiseProductEx(packagePath, scriptFilePath, transforms, locale, platform, instance);
    if(ret != ERROR_SUCCESS)
    {
        // Exception
        throw new InstallerException(ret);
    }
}
```

Listing 3.7: *Erstellen eines Skriptes zur Produktankündigung*

Analysieren des Skriptes

Bei der Skriptdatei handelt es sich um ein Verbunddokument (Compound Document), welches gewisse Ähnlichkeiten mit einem Windows Installer-Paket aufweist und daher auch mit Funktionen des Windows Installer-API betrachtet werden kann. Im *Windows Installer-SDK* finden Sie zusätzlich ein Visual Basic-Skript mit der Bezeichnung *WiLstScr.vbs*. Dieses Skript kann zur Auswertung und Analyse der Skriptdateien verwendet werden.

Fazit

Der Windows Installer verwendet unter den Betriebssystemen *Microsoft Windows NT 4.0*, *Microsoft Windows 2000*, *Microsoft Windows XP* und *Microsoft Windows Server 2003* zwei Prozesse, um eine Produktinstallation auszuführen. Der Client-Prozess wird hierbei immer mit den Privilegien des aktuellen Benutzers ausgeführt. Durch diesen Prozess werden die Interaktion und die Kommunikation mit dem Benutzer sichergestellt. Die tatsächlichen Installationsaufgaben, also das Kopieren von Dateien oder das Eintragen von Werten in die Systemregistrierung werden vom Server-Prozess ausgeführt. Dieser Prozess generiert zunächst ein Installationsskript, in das die auszuführenden Aktionen eingetragen werden. Im Anschluss wird dieses Skript vom Ausführungsmodul verarbeitet, wobei gegensätzliche Aktionen in ein Rollback-Skript übertragen werden. Dieses Rollback-Skript wird ausgeführt, falls die Installation nicht fehlerfrei beendet wurde, um die durchgeführten Änderungen rückgängig zu machen. Die einzelnen Tätigkeiten des Installationsprozesses gewinnen an Bedeutung, falls es darum geht, Fehler während der Installation zu erkennen und zu beseitigen. Das Installationsprotokoll ist hierbei die erste Wahl, auch wenn andere Analysefunktionen einen gewissen Charme besitzen.