Softwarepaketierung mit dem Package-Launcher 2020

Robuste Softwarepaketierung und
SCCM Application Automatisierung

Real Packaging GmbH

Eichenweg 9, 3123 Belp. www.realpackaging.ch

Inhaltsverzeichnis

1 Vorwort

Bei der Installation und Verwaltung von Softwarepaketen spielen einfaches Handling, eine transparente Implementation, Effizienz und klare sowie übersichtliche Schnittstellen eine grosse Rolle. Mit dem *Real Packaging Package-Launcher* erstellen Sie robuste Softwarepakete mit automatisierbaren Schnittstellen und verwalten Ihre *Updates* und *Upgrades*. Dabei bietet er zahlreiche Erleichterungen und Automatismen, die die Paketerstellung vereinfachen, eine transparente Softwareverwaltung ermöglichen und den Support vereinfachen. Das Servicemodell verfolgt neben vielen anderen Vorteilen vor allem ein Hauptziel: Die Optimierung des Softwarebereitstellungs- und Verteilungsprozesses.

1.1 Ziel und Zweck dieses Dokumentes

Das vorliegende Dokument dient als Referenz und Grundlage im Zusammenhang mit Fragen aus dem Bereich des *Package-Launchers*. Es werden alle Einstellungen und Verfahren aufgezeigt, welche zu einer erfolgreichen Anwendung von Softwarepakettransaktionen erforderlich sind.

1.2 Abgrenzungen

Die hier skizzierten Einstellungen und Methoden auf einen reibungslosen Betrieb in der Softwarepaketierung und im -Deployment des aktuellen Unternehmens, um Softwarepakete störungsfrei zu installieren und zu deinstallieren.

Zum Lieferumfang des *Package-Launchers* gehören Scripts, um die Qualität der erstellten Softwarepakete zu prüfen. Auf die Verwendung dieser Elemente wird in diesem Dokument nicht eingegangen.

Die Informationen in diesem Dokument beziehen sich auf die jeweils verfügbare aktuelle Version des Package-Launchers.

1.3 Legende

In *kursiv* geschriebene Wörter sind Namen, Fremdsprach- und Fachausdrücke, sowie Kapitelverweise. Auf Kapitelverweisen kann man mit „Ctrl. + Mausklick, dem Link folgen.

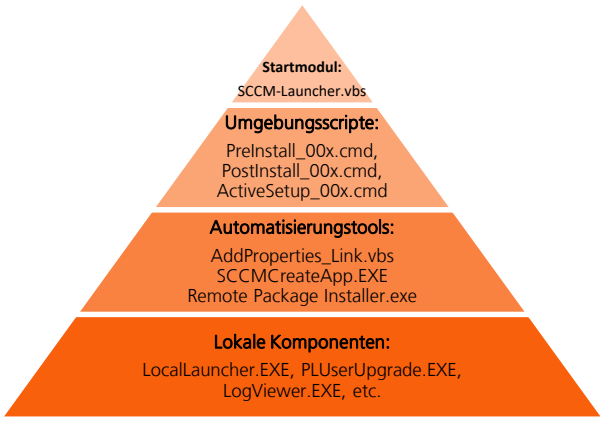
1.4 Vorarbeiten

Der *Package-Launcher* verwendet lokale Elemente. Vor Benutzung des *Package-Launchers* muss gewährleistet sein, dass das Softwarepaket *RealPackaging-PackageLauncher-2020* lokal installiert ist.

2 Methodische Betrachtung des Package-Launchers

2.1 Umfang des Package-Launchers

Der *Package-Launcher* besteht aus verschiedenen Elementen, die nur in sich als Ganzes einen logischen Zusammenhang ergeben. Viele der Elemente sind zur optionalen Verwendung gedacht und werden sich daher nicht in jedem Softwarepaket wiederfinden. Grundsätzlich übernimmt das Startmodul, das *SCCM-Launcher.vbs* das Auslesen des Paket-Wurzelverzeichnisses und das Übergeben der Kommandozeile an *LocalLauncher.EXE*. Dieser erledigt schliesslich die zentralen Aufgaben. Nach Bedarf wird zudem auf die optionalen, durch den Software-Paketierer zur Verfügung gestellten Umgebungsscripte zugegriffen.

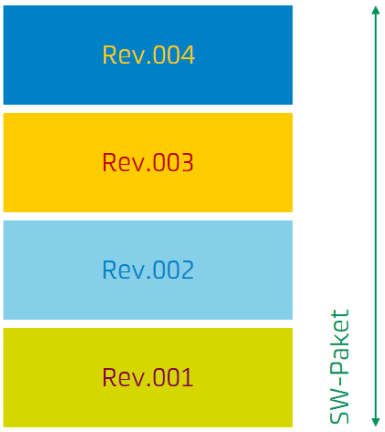


2.2 Aktualisierungen: Updates, Inplace-Updates und Upgrades

Die in diesem Dokument verwendeten Formen des Softwareaktualisierungsprozesses sind in *Updates, Inplace-Updates* und *Upgrades* unterteilt. Mit einem ***Update*** ist eine Aktualisierungsform gemeint, die eine bestehend installierte Software installiert belässt und diese erweitert oder Ressourcen daraus verändert. Demgegenüber steht der *Upgrade* von Anwendungen. Bei einem ***Upgrade*** handelt es sich um eine Aktualisierungsform, die in der Regel ein Softwareprodukt deinstalliert, um schliesslich ein neues Produkt zu installieren. Meistens werden solche Aktualisierungsformen bei einem neuen Softwarerelease angewendet. Für Spezialfälle gibt es noch *Inplace-Updates*, um veränderte Ressourcen als Update auszurollen. Siehe Kapitel *3.13.1 DMBuild (Inplace-Update)*

Der *Package-Launcher* unterstützt *Updates* in Form von *Revisionen*, sogenannte *Revisionsupdates*. Die Realisierung solcher *Updates* war eine der zentralen Anforderungen an den *Package-Launcher.* Neben dieser Form der Softwareaktualisierung werden *Upgrades* in Form einer vorgängigen Deinstallation eines Vorproduktes über zwei verschiedene Verfahren realisiert: den *Package-Upgrade*, der durch den *Package-Launcher* selbst gesteuert wird (Standard) und den *Major-Upgrade*, der auf die *Windows Installer* Technologie zurückgreift (nicht empfohlen).

2.3 Revisionsupdates

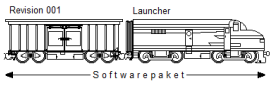


Das Hauptziel von *Revisionsupdates* ist es, vom Auftraggeber beauftragte Änderungsabsichten, Erweiterungen und Hersteller*updates*, die nach einer Produktivsetzung (*RTM*) eines Softwarepakets appliziert werden sollten, in einer zusätzlichen *Revision* abzubilden, ohne das getestete und eingeführte Basispaket abändern zu müssen. Der Umfang eines Softwarepakets wird dann neu auf die Summe aller *Teilrevisionen* (siehe Abbildung) erweitert.

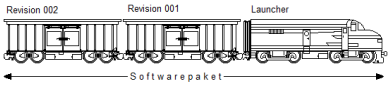
Alle *Revisionsupdates* werden chronologisch und transaktional installiert und deinstalliert. Bei der Anwendung zusätzlicher *Revisionen* erkennt der *Package-Launcher* den lokalen Installationsstatus und ergänzt die lokale Installation mit allen fehlenden *Revisionen*. Dieses Verfahren ist individuell und kann im Verlauf von Computer zu Computer unterschiedlich sein: Sollte beispielsweise ein Computer **A** den Installationsstatus einer Software der *Revision* 001 haben und ein anderer Computer **B** den Installationsstatus der *Revision* 002, so wird bei der Anwendung des gleichen *Revisionsupdates* auf die *Revision* 003 Computer **A** die *Revision* 002 und 003 erhalten, während Computer **B** automatisch nur deren Differenz 003 erhält. Die Überführung auf *Revision* 003 ist also einheitlich, die einzelnen lokalen Transaktionen, die zu diesem Ziel führen sind hingegen unterschiedlich. Der *Package-Launcher* wird automatisch die richtig installierte *Revision* an *SCCM* zurückmelden, damit dort der Status aktualisiert wird.

Durch das Verfahren der *Revisionsupdates* werden im Softwarebereitstellungsprozess bei Hersteller*updates* oder nachträglichen Erweiterungen **nicht mehr zwei Formen von Softwarepaketen erforderlich** – ein Vollpaket für alle, die das Produkt noch nicht installiert bekamen und ein *Updatepaket* für alle Clients, die bereits das Produkt ohne *Update* besitzen: Es kann lediglich das bestehende Softwarepaket mit einer neuen *Revision* ergänzt und die neue *Paketrevision* mittels automatischem Script *SCCMCreateApp_Link.vbs* in *SCCM* überführt werden. Mit dieser Realisierung sind alle nötigen „Hausaufgaben" gemacht.

Erste Produkteinführung mit *Revision* 001:



Update auf *Revision* 002 :



2.4 Transaktionaler Installationsprozess

Der *Package-Launcher* installiert die *Revisionen* des Softwarepakets transaktional. Wenn während der Installation einer *Revision* ein Fehler auftritt, so wird der Zustand vor der Installation dieser *Revision* wiederhergestellt und der Fehler wird in einer zentralen Protokolldatei ausgewiesen. Zudem erfolgt eine Rückmeldung der genauen Fehlermeldung an *SCCM*, so dass nach fehlerhaften Verteilungsaufgaben konkrete Rückschlüsse auf die lokal vorliegenden Probleme möglich werden.

Eine transaktionale Installation bedeutet, dass der *Package-Launcher* jede *Revision* immer vollständig oder gar nicht installiert und diesen effektiven Status auch unmittelbar an *SCCM* zurückmeldet. Durch dieses Modell wird verhindert, dass der Client in einen unverwaltbaren Mischzustand gerät. Beim Support oder beim Lösen von Störungen ist oft die Ausgangsbasis entscheidend. Wenn nun Anwendungsinstallationen in einem halb installierten, bzw. halbaktualisierten Zustand resultieren, können gerade solche Geisterinstallationen zu weiteren Folgefehlern beim Betrieb der Software und auch bei folgenden Installationstätigkeiten führen. Das transaktionale Modell dient letztlich zur Erhaltung eines konsistenten Systems und verhindert viele Folgefehler bereits im Keim.

Das transaktionale Verhalten wird folgendermassen realisiert:

- Der *Package-Launcher* überprüft seine Umgebung vor der Ausführung von schreibenden und löschenden Operationen nach Richtlinien, die zur erfolgreichen Änderung des Installationsstatus eingehalten werden müssen. Sollten Abhängigkeiten fehlen oder andere Richtlinien verletzt werden, wird dies an *SCCM* und im *History.LOG* rapportiert und die laufende Operation bricht ohne Änderung ab.

- *MSI*-Installationen erfolgen durch den *Windows Installer* von Grund auf transaktional. Die durch den Softwarepaketierer applizierten Implementationen in *Custom Actions* sollen im Fehlerfall ein *Rollback* auslösen, damit der Zustand vor der Installation wiederhergestellt wird. Nachfolgeprozesse werden nicht mehr ausgeführt.

- Installationen ohne *MSI*-Datei (bspl. *PreInstall_00x.cmd*) werden im Fehlerfall abgebrochen und Nachfolgeprozesse werden nicht mehr ausgeführt. Der Paketierer bemüht sich innerhalb der *PreInstall_00x.cmd* um die Einhaltung des transaktionalen Modells, eventuell auch unter Zurücknahme der im Script bereits durchgeführten Tätigkeit. Unter Umständen lassen sich dort platzierte Aktionen auch einfach erneut durchführen, wenn es zu einer Neuinstallation des Pakets kommt.

3 Technische Betrachtung des Package-Launchers

3.1 Hauptfunktionen des Package-Launchers

Folgende Tätigkeiten kommen mit der Verwendung des *Package-Launchers* in ihrem Grundsatz zur Anwendung:

- Unterstützung von verschiedenen Betriebssystemplattformen (*x86*/*x64*) und verschiedener Sprachausprägungen im selben Softwarepaket und Ermittlung des richtigen Paketverzeichnisses in Abhängigkeit des *Real Packaging/Package-Launcher/MainLanguage*-Keys und des Betriebssystems.
- *MSI*-Installationen: Alle allgemeinen *MST*-Dateien anwenden.
- Protokollieren der Transaktionen nach *%WINDIR%\Logs\History.LOG* und der Installationen/Deinstallationen nach *..\Logs\Install/Uninstall\<Paketname>_<Revision>.log*
- Rückmelden des Installationsstatus und allfälliger einzeiliger Fehlermeldung an *SCCM*. (immer in 32 Bit Registry, auch bei 64 Bit Clients!)
- Rückgabe eines qualifizierten Rückgabewertes an *SCCM*, welches die erfolgreiche Installation von einer nicht erfolgreichen Installation so unterscheiden kann.
- Lokales Zwischenspeichern des Softwarepaketes, wenn dies in der *INI*-Datei vermerkt ist und Entfernen lokal zwischengespeicherter Softwarepakete bei der vollständigen Deinstallation, sowie beim *Upgrade*.
- Möglichkeit der Prüfung nach Softwarepaketen, die vorgängig installiert sein müssen.
- Möglichkeit der Prüfung nach Softwarepaketen, die nicht lokal installiert sein dürfen.
- Anwendung von Berechtigungsanpassungen (Öffnen der Security).
- Unterstützung von *Updates* in Form von *Revisionen* und automatischen Upgrades durch den *Package-Launcher.*
- Unterstützung von Installationselementen, die vor, bzw. solche die nach der Hauptinstallation erfolgen sollen (*PreInstall_00x.cmd* und *PostInstall_00x.cmd*)
- Auslesen und Anwenden von allgemeinen Einstellungen und Vorgaben über eine *INI*-Datei, die den Namen des Softwarepaketes trägt.
- Verwalten von Neustarts mit Interaktion mit dem Benutzer.
- Trennung zwischen Entwicklungsumgebung und Produktionsumgebung beim Überführen der Softwarepakete.

3.2 Bestimmen des richtigen Paket-Unterverzeichnisses

Der *Package-Launcher* ermittelt das richtige Paket-Unterverzeichnis anhand der vorliegenden Verzeichnisstruktur, der lokalen Spracheinstellung in der Registry (*Real Packaging/Package-Launcher/MainLanguage)*, sowie der verwendeten Plattform (*x86* oder *x64*). Zuerst wird ermittelt, ob ein Paketverzeichnis mit dem selben Plattform/Sprachkürzel vorhanden ist, wie die Kombination, die der Client selbst verwendet. Auf einem *x86*-Client, mit deutscher Einstellung (*GE*) käme beispielsweise primär ein Unterverzeichnis mit dem Namen *x86-GE* zur Anwendung. Wenn kein identisches Unterverzeichnis im Softwarepaket vorgefunden wird, dann wird versucht, das Unterverzeichnis mit einem identischen Plattformtyp und der für die aktuelle Spracheinstellung am ähnlichsten vorgefundenen Sprache zu ermitteln. Die Sortierreihenfolge, die hier angewendet wird, ist folgende:

Wenn Registrykey *LanguageSort=New* verwendet wird (standardmässig, ab 2020):

Client	1.Wahl	2. Wahl	3. Wahl	4. Wahl	5. Wahl
GE	GE	ML	EN	FR	IT
FR	FR	ML	EN	GE	IT
IT	IT	ML	EN	GE	FR
EN	EN	ML	GE	FR	IT

Ist beispielsweise die lokale Plattform *x64* und die verwendete Spracheinstellung *GE*, existieren aber mit dem *x64*-Bezeichner nur zwei Unterverzeichnisse mit der Bezeichnung *x64-EN* und *x64-FR*, dann wird *x64-**EN*** angewendet. Wird für den aktuellen Plattformbezeichner kein einziges Unterverzeichnis gefunden, so wird bei *x64* (und nur bei *x64*!) die Suche nach *x86*-Paketen erweitert.

Sollte gar kein regelkonformes Verzeichnis gefunden werden, protokolliert dies der *Package-Launcher* im History.LOG mit der folgenden Fehlermeldung:

```
"Package folder for this language not found! (GE)"
```

Achtung
Diese Meldung kann auch daher kommen, wenn im Explorer das Softwarepaket per Doppelklick ausgeführt wird und im Kontext des Administrators nach Bestätigung der UAC-Meldung kein Zugriff auf den Ablagepfad des Softwarepakets möglich ist!

Beispiele:

Aktuelle Plattform	Aktuelle Sprache	Vorgefundene und gewähltes Unterverzeichnis/se (fett=gewähltes Verzeichnis)
x86	GE	x64-EN,x64-GE, **x86-GE**, x86-FR, x86-IT
x86	GE	x64-EN,x64-GE, x86-IT, **x86-FR**
x64	FR	x86-GE, **x86-FR**, x86-IT
x86	FR	x64-GE, x64-FR, x64-IT -> Fehlermeldung in *History.LOG* (kein gültiges Verzeichnis gefunden)

Achtung: Es wird immer nur **ein** Unterverzeichnis ausgewählt!

3.3 Kommandozeilenoptionen

Die Kommandozeilenoptionen, die mit dem Start des *Package-Launchers* übergeben werden können, sind kombinierbar und fehlertolerant. Beim Aufruf spielt es grösstenteils keine Rolle in welcher Reihenfolge die Optionen eingegeben werden, ob gross- oder klein geschrieben wird und ob "/" oder "-" als Optionenkennzeichner verwendet werden. Sogar in sich ausschliessende Varianten (/x und /i - Deinstallation und Installation) sind in gewissem Masse zulässig: Dann gewinnt einfach die letzte übergebene Option der ausschliessenden Optionen. Neben den uns gebräuchlichen Varianten sind auch die Optionen des *Windows Installers* in seiner Terminologie zulässig. Zusätzliche *Windows Installer Properties* für *MSI*-Installationen können im Testumfeld einfach angefügt werden (*PROPERTY=VALUE*). Generell übersteuern Kommandozeilen-Optionen die Einstellungen, die in der *INI*-Datei abgespeichert sind. Mit /? werden die Kommandozeilenoptionen angezeigt.

3.3.1 Folgende Kommandozeilenoptionen sind zulässig

Option	Beschreibung
/i	Installiert oder reinstalliert Produkt mit Fortschrittsanzeige. Ist die Applikation bereits auf dem System installiert, so wird die Installation ab der nächstfolgenden *Revision* im Rahmen eines *Revisionsupdates* gestartet (Differenz der installierten Version zu der zu installierenden Version) Ist das Produkt in der gleichen *Revision* wie die maximal verfügbare *Revision* aus der Paket-Source bereits korrekt auf dem System installiert, so wird eine Reparatur aller *Revisionen* ab *Revision* 001 ausgelöst.
ohne Parameter	Entspricht /i (Bspl. Doppelklick auf *SCCM-Launcher.vbs*)
/i:00x	Installiert und/oder reinstalliert Produkt **ab** der *Revision* 00x. Durch diesen Mechanismus kann genau vorgegeben werden, ab welcher *Revision* die Installation starten soll. Es ist zur Installationszeit keine Source < 00x erforderlich. Bspl. Installiert / Kommandozeile / Auswirkung 003 / /i:004 / installiert ab *Revision* 004 003 / /i:005 / Fehlermeldung in *History.LOG* 003 / /i:003 / 003 wird repariert
/i:-00x	Installiert und/oder reinstalliert Produkt **bis und mit** *Revision* 00x. Diese Option erlaubt die Verteilung des Produktes losgelöst von der Installation durchzuführen. Wird ein Job so initiiert, so hat man Gewähr, dass bis zu der mit 00x angegebenen *Revision*, unabhängig weiterer verfügbarer Source installiert wird. Bspl. Paket-Source verfügbar / Kommandozeile / Auswirkung 005 / /i:-004 / installiert bis *Revision* 004 005 / /i:-009 / installiert bis und mit *Revision* 005, dann Fehler im History.LOG 004 / /i:-005 / Fehler im History.LOG

/i:00x /i:-00x	Kombination von Beginn bis Ende *Revision.*
/s (/q)	Installiert oder deinstalliert ohne Benutzerinterface.
/e	Übersteuerung der Fehlermeldung „REMOVE action is only valid for products which are currently installed". D.h. Deinstallation wird weitergeführt
/f	Reparatur wird ausgelöst. **Achtung:** Wird gleichzeitig eine *Initialrevision* vorgegeben (Bspl: /i:003), dann gewinnt die Vorgabe der *Initialrevision*!
/x	Deinstallation des Produktes. Deinstalliert das Produkt vollständig rückwärts ab der letzten erfolgreich installierten *Revision* (003 -> 002 -> 001). Sollte bei der Deinstallation ein Fehler auftreten, so gilt das Produkt nachwievor als installiert – und zwar bis zur letzten erfolgreich deinstallierten *Revision* minus 001. **Achtung:** Die Deinstallation kann, muss aber nicht mit dem Startmodul *SCCM-Launcher.vbs* initiiert werden. Die bevorzugte Methode ist, *LocalLauncher.EXE* direkt anzusprechen. Dies ermöglicht die Deinstallation von Softwarepaketen ohne verfügbare Source. Bspl: %WINDIR%\Package-Launcher\Bin\LocalLauncher.EXE Adobe-Reader-9.3 /x %PL%\LocalLauncher.EXE Adobe-Reader-9.3 /x
/x:-00x	Deinstalliert ab installierter *Revision* rückwärts **bis und mit** *Revision* 00x. installiert Kommandozeile Auswirkung 005 /x:-003 deinstalliert *Rev.* 005, dann 004, dann 003
/G	Ermöglichte einen Upgrade von Paketen ohne darauffolgende Installation. Bspl. LocalLauncher.exe /G Upgrade=Adobe-Reader …entfernt alle Adobe-Reader Versionen auf dem Client
MainLanguage	Gibt für den *Package-Launcher* die Clientsprache (MainLanguage) vor, anstatt dass dieser den Wert aus der Registry liest. Bspl. MainLanguage=GE
ForcedMainLanguage	Setzt die Clientsprache (Registrykey) **temporär** auf den übergebenen Sprachkürzel. So ist gewährleistet, dass auch *AppSearch*-Implementationen aus MSI-Dateien und *Pre-* oder *PostInstall*-Anweisungen, sowie Scripts, die auf den Registrykey prüfen, reibungslos funktionieren. Bspl. ForcedMainLanguage=FR
INI-Einträge	*Upgrade*, *RemoveIncompatibleMsi*, *PlatformLangChange*, *UpgradeExitIfFailed*, *CopyLocal*, *AppVAutoStopProcesses*, *AllowDowngrade*, *Dependence* & *Incompatibilities*, *AppShutdown* …können statt aus der INI-Datei auch als Kommandozeilenoption angegeben werden. Die Kommandozeilenoption übersteuert die INI-Datei. Bspl: SCCM-Launcher.vbs Dependence=-
PROPERTY	*Windows Installer Properties* können der Kommandozeile übergeben werden, indem die Property, das Gleichheitszeichen „=" und der Zuweisungswert angefügt werden. Enthält der Zuweisungswert ein Leerzeichen, so ist der Zuweisungswert in Anführungszeichen einzufassen.

	Bspl.: SCCM-Launcher.vbs INSTALLDIR="C:\Program Files\MyDir"
VARIANTEN	Wird ein einzelnes Wort ohne „/“ und ohne „=“ dem Befehlszeilenaufruf angefügt, so interpretiert dies der *Package-Launcher* als Variante. Bspl: SCCM-Launcher.vbs ADMIN

3.3.2 Kommandozeilen-Beispiele

Aufruf	Auswirkung
Doppelklick auf ..\Pack\SCCM-Launcher.vbs	• Sofern das Paket noch nicht installiert wurde, wird das Paket in allen in der Paket-Source verfügbaren *Revisionen* **installiert**. • Ist das Produkt hingegen in einer kleineren *Revision*, als in der Paket-Source verfügbar, auf dem System installiert, so wird ein ***Revisionsupdate*** ab der installierten *Revision* bis zu der in der Paket-Source verfügbaren *Revision* durchgeführt. • Ist das Produkt in der gleichen *Revision*, wie die maximal in der Paket-Source verfügbaren *Revision* installiert, so wird eine Reparatur aller *Revisionen* bis zu der in der Paket-Source verfügbaren *Revision* ausgelöst.
..\Pack\SCCM-Launcher.vbs /i	Das selbe Verhalten wie oben abgebildet.
..\Pack\SCCM-Launcher.vbs /i INSTALLDIR=D:\	Paket wird in allen verfügbaren *Revisionen* installiert. Die Installation erfolgt mit der erweiterten Property *INSTALLDIR*, welche auf „D:\“ gesetzt wird.
..\Pack\SCCM-Launcher.vbs /i:-003	• Ist das Paket noch nicht auf dem Client installiert, so wird das Paket bis und mit *Revision* 003 vollständig installiert. • Ist das Paket bereits in einer *Revision* < 003 auf dem System vorhanden, so wird ein *Revisionsupdate* bis 003 ausgeführt. • Ist das Paket bereits in der *Revision* 003 auf dem System vorhanden, so wird eine Reparatur bis und mit 003 ausgeführt. • Ist das Paket sogar mit einer höheren *Revision* auf dem System installiert, so resultiert ein Abbruch mit Fehlerausweisung im *History.LOG*: *ERROR: Given Last Revision (003) is < installed Revision (004)*
..\Pack\SCCM-Launcher.vbs /i:003 /s	Das Paket wird **ab** *Revision* 003 silent ohne Benutzerinterface installiert. Ist 003 grösser als die installierte *Revision* + 1, dann wird mit Fehler abgebrochen.
..\Pack\SCCM-Launcher.vbs /i:002 /i:-003	Das Paket wird **ab** *Revision* 002 **bis** *Revision* 003 installiert. Für alle zu installierenden *Revisionen*, die bereits auf dem System installiert sind wird eine Reparatur ausgelöst, für alle anderen *Revisionen*, die zudem > 001 sind wird ein *UPDATE* (*Revisionsupdate*) ausgelöst.
..\Pack\SCCM-Launcher.vbs /x oder %WINDIR%\Package..\Bin\LocalLauncher.EXE Byron-SALZ-1.0.3 /x	Das Paket wird komplett (*alle Revisionen*) deinstalliert. **Achtung:** Deinstallationen können auch direkt über *LocalLauncher.EXE* – auch ohne verfügbare Source abgesetzt werden:
..\Pack\SCCM-Launcher.vbs /x:-003	Das Paket wird rückwärts bis **und mit** *Revision* 003 deinstalliert.

3.3.3 Kommandozeilenempfehlung für die Integration in SCCM

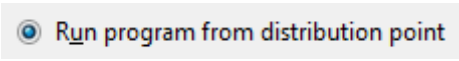
Installation:
Standard: SCCM-Launcher.vbs /i /q
Variante: SCCM-Launcher.vbs ADMIN /i /q

Revisionsupdate:
SCCM-Launcher.vbs /q /i:00x (bspl. SCCM-Launcher.vbs /q /i:002)

Deinstallation:
%PL%\LocalLauncher.EXE Byron-SALZ-1.0.3 /x /q
Instanzvariante: …\LocalLauncher.EXE Byron-SALZ-1.0.3-ADMIN /x /q

Die Parameter unterscheiden sich in den angebrachten *SCCM*-Beispielen neben der *Initialrevision* bei *Revisionsupdates* nur durch den *Silent*-Switch. Dadurch wird verhindert, dass bei *SCCM*-Jobs dem Benutzer eine Fortschrittsanzeige angezeigt wird (Beispielsweise durch *PostInstall_00x-CustomActions*).

Deinstallationen können mit dem lokalen *Launcher* ohne Verfügbarkeit der Paket-Source ausgeführt werden. Auf einen Download durch *SCCM* kann somit verzichtet werden. Das *Uninstall-Advertisement* wird durch das Script *Create_SCCM_Package_Link.vbs* (*SCCM 2007*) daher mit der Option ohne vorherigen Download erstellt:



Achtung: Die vorgängig dokumentierten Kommandozeilenoptionen werden derzeit automatisch beim Erstellen der *SCCM*-Application mit *SCCMCreateApp_Link.vbs* generiert!

3.3.4 Rückgabewerte

Der *Package-Launcher* gibt 5 verschiedene Rückgabewerte zurück:

Rückgabewert	Bedeutung
1	Fehler vor Ausführung der effektiven Installationstätigkeit
2	Fehler im Befehlszeilenaufruf
3	Standardfehler, Details in History.LOG und Registry ersichtlich
999	Paket RealPackaging-PackageLauncher-2020 ist lokal nicht installiert
3010	Bei CommitRebootExitCode=True
1641	Bei CommitRebootExitCode=True
1618	Abbruch nach Prüfung von Prozessen (CheckProcesses) → *Fast Retry* in *SCCM CB*

In der Registry legt der *Package-Launcher* unter *HKLM\Software\Real Packaging\Package-Launcher\Packages\<PackageName>\MsiExecReturn* zudem den Rückgabewert der letzten nicht erfolgreich ausgeführten *Windows Installer* Transaktion des entsprechenden Pakets ab.

3.4 Fehlermeldungen und History.LOG

Der *Package-Launcher* weist ein genaues Fehlermanagement auf und ermöglicht die schnelle Erkennung von allen Installationsfehlern an einem zentralen Ort oder über *SCCM*. Ein Fehler wird immer via *LogWriter.EXE* im *History.LOG* ausgewiesen und in die Registry geschrieben.

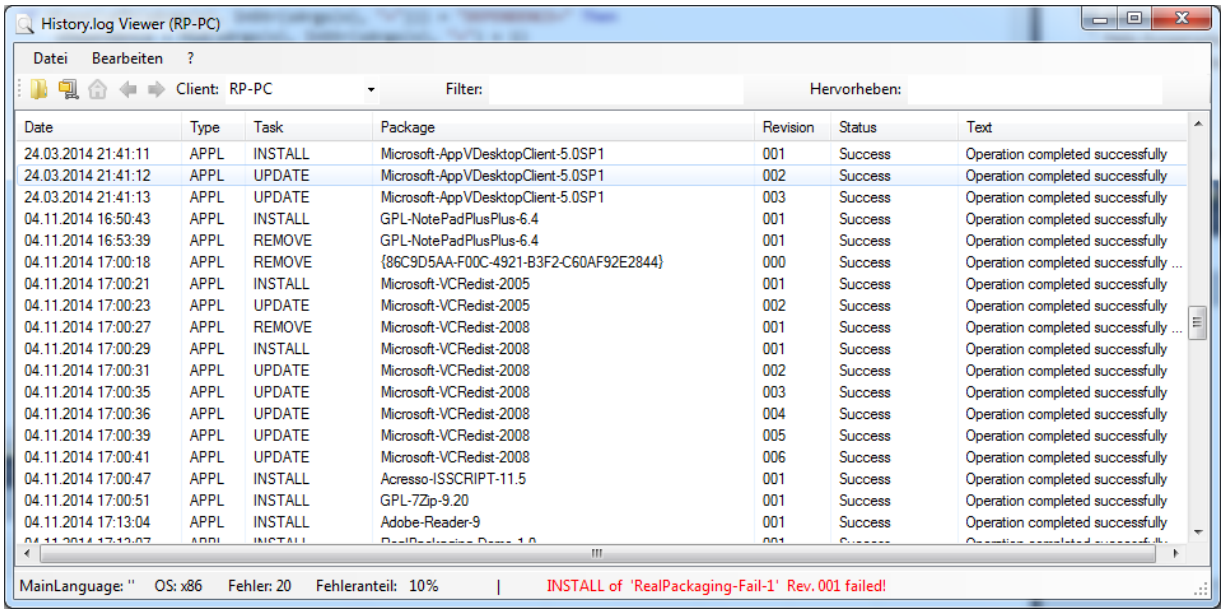
Viele Erkenntnisse aus dem Paketierungsbetrieb flossen in die Erarbeitung des *Package-Launchers*. So ist es beispielsweise technisch nicht möglich, dass der *Package-Launcher* über eine zweite Instanz versucht, gleichzeitig eine zweite Installation auf dem gleichen Client zu starten und so das transaktionale Modell unterlaufen könnte. Dieser Fehler wird vorher abgefangen. Auch Installationen von Produkten, die mit gleichem *ProductCode* bereits (oder noch?) auf dem System installiert sind (bspl. Handinstallationen), seien diese für den Computer oder für einen Benutzer installiert, werden erkannt und können wahlweise automatisch deinstalliert werden oder zum dokumentierten Abbruch der aktuellen Installation führen. Auf die durch solche Fehler entstehende mühsame Fehlersuche kann daher künftig verzichtet werden. Problematisch wären solche Situationen vor allem dort gewesen, wo das Produkt in einer „veränderten Fassung" (keine erweiterten *CustomActions*, andere Einstellungen, etc.), beispielsweise wie vom Hersteller geliefert, ausserhalb der Mechanismen des Unternehmens (*Package-Launcher*) installiert würden und dann das „gleiche" Produkt in Form eines *Package-Launcher*-Installationspakets hätte installiert werden sollen. Dadurch, dass *Windows Installer* bei der Installation des Installationspakets auf die gecachete Variante zugreifen würde, könnte nicht selten ein Zustand oder Abbruchsverhalten entstehen, welches schwer analysierbar und auch reparierbar wäre. Daher wird per Standard im Einsatz mit dem *Package-Launcher* ein solcher Zustand im *History.LOG* ausgewiesen und die aktuelle Operation abgebrochen.

Bei einem entstandenen Fehler durch den *Windows Installer* Installationsprozess erfolgt durch den *Package-Launcher* die Auswertung der *Windows Installer* Protokolldatei. Der ermittelte Reintext der Fehlermeldung wird nun an *SCCM* rapportiert und ebenfalls in der Datei *History.LOG* ausgewiesen. Dies gestattet einen einfacheren Support, kann in *SCCM* aber auch zu Statistikzwecken verwendet werden. Zudem wird erst dadurch eine robuste Verwaltung der Softwareinstallationen mittels *SCCM* möglich.

3.4.1 History.LOG

Die zentrale Protokolldatei zeigt übersichtlich den Transaktionsverlauf aller Softwareinstallationen dar. Ob, wann und wie eine Transaktion ausgeführt wurde, dies alles finden wir in der zentralen Datei *%WINDIR%\Logs\History.LOG*. Aufgrund des transaktionalen Modells finden wir denn auch in der Regel genau eine Zeile pro Transaktion. Der Aufbau sieht folgendermassen aus:

Ansicht History.Log mit dem History.LOG Viewer:



Neben dem Status *Error* und *Success* kann der Software-Paketierer mittels Vorlagen im *PreInstall_00x.wse* und *PostInstall_00x.wse* auch noch eine Informationszeile (*Information*) ausgeben. Andere Statusmeldungen sind nicht erlaubt. Das *History.LOG* wird durch das externe Programm *%WINDIR%\Package-Launcher\Bin\LogWriter.EXE* geschrieben. Dadurch wird externen Prozessen der schreibende Zugriff auf das *History.LOG* ermöglicht. Mittels *%WINDIR%\Package-Launcher\Bin\LogViewer.EXE* kann die Protokolldatei in übersichtlicher Form angezeigt werden.

3.4.2 Bedeutung von Fehlermeldungen

In der folgenden Tabelle werden einige *History.LOG* Fehlermeldungen beschrieben. (Nur solche, die nicht aufgrund einer *MSI*-Fehlermeldung entstehen (alphabetisch aufsteigend sortiert):

Fehlermeldung	Bedeutung
%WINDIR%\Logs not exist for package logfile	Auf Logfile-Pfad kann nicht zugegriffen werden.
abnormal Exception!	Aussergewöhnlicher Fehler mit weiteren Angaben.
A REPAIR can only be performed for products which are installed successfully!	Das gewählte, für eine Reparatur vorgesehene Paket ist auf diesem Client gar nicht installiert.
A REMOVE can only be performed for products which are installed successfully!	Das gewählte, für eine Deinstallation vorgesehene Paket ist auf diesem Client gar nicht installiert.
Cannot open package. Is it in use?	Auf die *MSI*-Datei konnte nicht zugegriffen werden. Möglicherweise ist die Datei geöffnet.
Copy source to local drive failed!	Das aufgrund der Einstellung *CopyLocal=True* beauftragte lokale Kopieren des Softwarepakets in das Verzeichnis *%WINDIR%\PackageLauncher-\Cache* konnte nicht durchgeführt werden.

Error during determine ProductCode!	Der *ProductCode* der zu installierenden Anwendung konnte nicht ermittelt werden.
Given Initial Revision (003) is > Installed Revision (001) + 1	Initial*revision* (/i:003) muss <= aktuell installierte *Revision* + 1 betragen.
Given last REMOVE Revision (003) is > Installed Revision (002)	Ein über /x:-003 angegebene Deinstallation kann nicht ausgeführt werden, da die momentan installierte *Revision* kleiner ist, als die verlangte *EndRevision*.
Given Last Revision (003) is < installed Revision (004)	*Lastrevision* (/i:-003) muss >= aktuell installierte *Revision* betragen
I/O Exception while copying HP-Soft-1.0_001.cmd	Fehler beim Kopieren des Security-Batches.
I/O Exception while copying files to ..\Package-Launcher\Scripts path! I/O Exception while sync files in ..\Package-Launcher\Scripts path!	Fehler beim Aktualisieren oder Löschen von *PreInstall*,*PostInstall Scripts*-Verzeichnis.
Incompatible Application installed (Byron-HIP-1.0)	Ein in der *INI*-Datei unter *Incompatibilities* vermerktes Softwarepaket wurde auf diesem Client installiert vorgefunden.
Installation source for Revision 003 not exist	Es konnte eine von *Initalrevision* (/i:003) oder *LastRevision* (/i:-003) verlangte *Revision* nicht in der Paket-Source vorgefunden werden.
Invalid command line arguments for MSIEXEC.EXE.	Falsche Kommandozeilenparameter wurden übergeben oder wurden über *MsiInstallProperties* in der *INI*-Datei deklariert.
Invalid Installation/Remove command line arguments: overflow exception!	Unzulässige Zeichen in der Kommandozeile verwendet.
Nothing made! No valid source found! (argLine: Byron-HIP-1.0 c:\Temp /s /qb)	Es wurde nichts ausgeführt. Gründe sind unbekannt (keine unterstützte Installationsdateien im *Revisions*verzeichnis vorhanden?))
Other Windows Installer process in progress! Try again later.	Im Hintergrund läuft bereits eine Installation oder eine *Windows Installer* Reparatur.
Package folder for this language not found! (GE)	Es konnte kein geeignetes Paket-Unterverzeichnis gefunden werden.
PreRequisite Software not installed (Byron-HIP-1.0)	Eine in der *INI*-Datei unter *Dependence* übergebene Applikationsabhängigkeit ist auf dem Client nicht installiert.
REMOVE action is only valid for products that are currently installed. This Revision isn't installed at this time!	Die Deinstallation der aktuellen *Revision* kann nicht ausgeführt werden, da sie zurzeit gar nicht mehr installiert ist.
REMOVE of incompatible MSI has returned with failure. Please check Byron-HIP-1.0_001.INCOMPATIBLE.TXT	Fehler während der automatischen Deinstallation fremdinstallierter Software (Pakete, welche nicht durch den *Package-Launcher* auf dem System installiert wurden). Dieser Mechanismus wird über

	den *INI*-Eintrag oder die Kommandozeilenoption *RemoveIncompatibleMsi=True* initiiert.
Terminalserver is in EXECUTE mode. Switch to INSTALL mode!	Beim Installationsziel handelt es sich um einen Terminalserver. Der Terminalserver muss aber vor der Paketinstallation zuerst mit „change USER /INSTALL“ in den Installation mode gebracht werden.
The ProductCode of this Revision is used in other Revision (Byron-HIP-1.0_002)!	Der Paketierer verwendet eine ungültige *MSI*-Datei in einem bestimmten *Revisions*verzeichnis.
The program cannot be started several times. LocalLauncher.EXE already running with following application: "Byron-HIP-1.0". Try again later.	Es wurde versucht eine Installation zu tätigen, obwohl parallel oder im Hintergrund bereits eine Installation läuft.
This application is already installed by other package. Please remove other package (Adobe-Reader) and try again!	Das Produkt ist unter einem anderen Paketnamen bereits auf diesem Client installiert. Bitte Paketierung melden!
This application is already installed manually or by other processes. Please remove the product first (msiexec /x {1F6435C5-429D-42E5-B0B7-CBEAEE66EA0F}) or set INI entry RemoveIncompatibleMsi to 'True'	Die Applikation wurde auf diesem System bereits in anderer Form, als durch eine reguläre Paket-variante installiert oder die zu installierende Ver-sion wurde in installiertem Zustand unter anderem Namen vorgefunden. Durch Setzen des *INI*-Eintrages oder der Komman-dozeilenoption ***RemoveIncompatibleMsi=True*** kann ein automatisches vorgängiges Deinstallieren erwirkt werden.
This package is installed in other language. Please remove old package first! (Old package: x86-GE – This package x86-ML)	Dieses Softwarepaket wurde auf diesem Client bereits installiert. Bei der letzten Installation wurde aber ein anderer Plattform/Sprachfolder verwen-det. Entfernen Sie das bestehende Paket zuerst.
This package requires a reboot after the instal-lation of following products: Byron-HIP-1.0	Gemäss Vorgabe in der *INI*-Datei muss eine Ab-hängigkeit installiert und danach ein Reboot aus-gelöst sein, damit dieses Softwarepaket installiert werden kann. Diese Bedingung trifft auf diesem Client nicht zu.
Unable to determine if this installation is a Small Update or Minor Upgrade to installed Product {B23C702D-2BB9-42F7-818A-7EFE88BA7371}. Error while open cached msi ...	Wenn ein Paket mit dem selben *ProductCode* bereits installiert ist, das lokal gecachte *MSI* aber nicht geöffnet werden kann, erscheint diese Meldung.
Unexpected Error during execution of PreInstall	Nicht lokalisierbarer Fehler bei der Ausführung von *PreInstall*. *PreInstall* hat einen Fehlercode zurück-geliefert, der Fehlermeldungseintrag unter *HKLM-\...\Packages\PKG\Error* fehlt aber.
Unexpected Error during execution of install.msi (Return value 2099)	Unbekannter Fehler während der Ausführung des *MSIs*. Die Ausführung von *MSIEXEC* lieferte nicht 0 zurück, in der Protokolldatei ist aber kein Fehlertext erkenntlich.

3.5 INI-Datei

In der *INI*-Datei des Softwarepakets werden Einstellungen definiert, die für die Installation des Softwarepakets relevant sind. Die *INI*-Datei wird im Wurzelverzeichnis des Softwarepakets abgebildet und trägt den Namen des Softwarepakets. Bspl. *Byron-HIP-1.0.INI.* Wie mit den wichtigsten Bezeichnern umgegangen wird und wie sie im Softwarepaket eingesetzt werden, finden Sie im Kapitel *6.10 INI-Datei des Softwarepakets*.

3.5.1 Zusammenfassung der wichtigsten INI-Einträge

Anbei finden sich die gängigen Keys unter der *Section Install*. Für alle nicht unter der *Section Install* zu verwendenden Keys ist die *Section* in Klammer angegeben:

Key	InstallFile
Einsatz	Optional
Beschreibung	Hier kann die Bezeichnung der *MSI*-Datei angegeben werden. Der Wert ist optional. D.h., wenn er leer ist, dann wird vom *Package-Launcher* die erstbeste *MSI*-Datei verwendet, die im *Revisions*verzeichnis vorgefunden wird. Ist der Eintrag angegeben, so wird primär nach der *MSI*-Datei gesucht, die wie angegeben benannt ist. Nur wenn keine solche gefunden wird, wird die Suche auf andere Dateinamen ausgeweitet. Eine Möglichkeit der Verwendung ist die, dass man im selben *Revisionsverzeichnis* neben einer Installations-*MSI*, noch eine *MSI* verwendet, die zur Installation von „UserSettings" vorgesehen ist und bei der durch den *Package-Launcher* initiierten Installation selbst nicht zu installieren ist. Der *Package-Launcher* wird dann angewiesen, die andere, im selben Verzeichnis befindliche *MSI*-Datei zu installieren.
Beispiel	InstallFile=install.msi

Key	CopyLocal
Einsatz	Optional (True¦False)
Beschreibung	Steht der Eintrag auf ‚True', dann wird die Source des kompletten Softwarepaketes lokal nach *%WINDIR%\Package-Launcher\Cache\<Package-Name>* kopiert. Bei einer *Active-Setup*-Implementation kann so zum Beispiel gewährleistet werden, dass die Source nach der Installation für die Benutzerkonfiguration zur Verfügung steht. Achtung: Es kann nur ein komplettes Paket zwischengespeichert werden! Die lokale Source wird bei einer kompletten Deinstallation automatisch entfernt.
Beispiel	CopyLocal=True

Key	**Reboot**
Einsatz	Optional (True¦False)
Beschreibung	Steht der Eintrag auf ‚True', dann wird nach der Installation des Softwarepaketes der Eintrag *MakeReboot* in der Registry gesetzt. Dieser Key wird vom *Package-Launcher Restart Manager* verwendet, um den Benutzer dafür aufzufordern, einen Neustart des Computers auszulösen. Im abgemeldeten Zustand wird der Neustart automatisch ausgelöst. **Achtung:** Wird keine *Revision* am Bezeichner angegeben, so erfordert das Schreiben des Registrykeys nach Beendigung der kompletten Installation. Soll der Key nur bei einer spezifischen *Revision* geschrieben werden, so kann dies mittels *Reboot_00x=True* bewerkstelligt werden.
Beispiel	Reboot_002=True

Key	**Logoff**
Einsatz	Optional (True¦False)
Beschreibung	Steht der Eintrag auf ‚True', dann wird nach der Installation des Softwarepaketes der Registry-Eintrag *MakeLogoff* in der Registry gesetzt. Dieser Key wird vom *Package-Launcher Restart Manager* verwendet, um den Benutzer dafür aufzufordern, sich neu anzumelden.
Beispiel	Logoff=True

Key	**Dependence**
Einsatz	Optional
Beschreibung	Dieser Bezeichner ist mit den Paketnamen zu ergänzen, welche als Abhängigkeit dieser Applikation fungieren. Ein einzelner Paketname kann auch nur eine Teilbezeichnung der Applikation enthalten. Die verschiedenen Abhängigkeiten sind mit einem Lehrzeichen aneinanderzufügen. Die Abhängigkeiten können mit dem ‚OR' Operator verknüpft. Der *Package-Launcher* prüft vor der Installation, ob all diese Abhängigkeiten lokal installiert sind. Beachten Sie auch die Ausführungen unter Kapitel *6.10.3 Umgang mit Abhängigkeiten*. Neben dieser Funktionalität werden über den selben Bezeichner beim Überführen des Softwarepakets zusätzlich die so deklarierten Abhängigkeiten in den *Deploymenttype* der *Application* eingetragen (*SCCM CB*).
Beispiel	Dependence=Adobe-AcrobatReader Acresso-InstallScriptMSIEngine

Key	**DependenceReboot**
Einsatz	Optional
Beschreibung	Wie Primärfunktion aus *Dependence*, nur muss gewährleistet sein, dass nach der Installation dieser Abhängigkeit bis zur Installation dieses Produktes mindestens ein Reboot des Computers ausgeführt wird. Ergibt die Prüfung durch den *Package-Launcher* False, dann wird folgende Meldung ins History.LOG geschrieben: *This package requires a reboot after the installation of following products: PackageName-Version*

Key	TaskSequence
Einsatz	Optional
Beschreibung	Über den Bezeichner *TaskSequence* ist es möglich, einen zum *Dependence*-Eintrag abweichenden Aufbau der Abhängigkeiten anzuführen, die für die automatische Erstellung der *Customized Task Sequence* beim Überführen des Softwarepakets verwendet werden (*SCCM 2007*) und für RPI-Installationen gelten soll. Die Abhängigkeiten werden der Reihe nach ausgewertet. Wird der Eintrag nicht verwendet, so wird stattdessen der *Dependence*-Eintrag interpretiert.
Beispiel	Dependence=SAP-SAPGUI TaskSequence= Microsoft-VCRedist-2008 SAP-SAPGUI

Key	Incompatibilities
Einsatz	Optional
Beschreibung	Ist mit den Paketnamen zu ergänzen, welche nicht installiert sein dürfen. Der Paketname kann auch nur ein Teilstring der Applikation enthalten. Die verschiedenen zu dieser Applikation inkompatiblen Paketnamen sind mit einem Leerzeichen aneinanderzufügen. Die *Incompatibilities* können mit dem ‚AND' Operator verknüpft werden (siehe *6.10 INI-Datei des Softwarepakets*)
Beispiel	Incompatibilities=iTunes QuickTime

Key	Upgrade
Einsatz	Optional
Beschreibung	Dieser Key bestimmt, welche Softwarepakete der *Package-Launcher* in Form eines *Package-Upgrades* vor einer Installation entfernt. Dies ist die favorisierte Variante, Softwareprodukte im Rahmen eines *Upgrades* zu entfernen. Nicht *Package-Launcher* kompatible Pakete können durch Hinzufügen des oder der {ProductCodes} entfernt werden. **Achtung:** Fehlt dieser Key oder ist er leer, so wird bei der Ausführung von *AddProperties.vbs* die *MSI/MST*-Datei modifiziert, um einen *Major-Upgrade* in Form von *Windows Installer* Erweiterungen zu applizieren! Für die korrekte Bereinigung der Inventarisierungsregistrykeys ist im Rahmen eines *Major-Upgrade* eine *Custom Actions* zuständig – wird hingegen ein *Package-Upgrade* durchgeführt, so erledigt der *Package-Launcher* diese Aufgaben direkt.
Beispiel	Upgrade=Adobe-AcrobatReader Acresso-InstallScriptMSIEngine

Key	RemoveIncompatibleMsi RemoveIncompatibleMsiUPG
Einsatz	Optional
Beschreibung	Entfernt Produkte, die mit dem selben *ProductCode* ausserhalb der Mechanismen des *Package-Launchers* installiert sind (*RemoveIncompatibleMsi*), bzw. entfernt im Rahmen eines *Package-Upgrades* alle *MSI*-Installationen, die den selben *UpgradeCode* (*RemoveIncompatibleMsiUPG*) verwenden.
Beispiel	RemoveIncompatibleMsi=True

Key	**MsiInstallProperties**
Einsatz	Optional
Beschreibung	*Public Windows Installer Properties* können so für die **Installation** übergeben werden. Zwischen zwei Properties ist ein Leerzeichen einzufügen. *Public Properties* sind in Grossbuchstaben zu verwenden. **Achtung:** Durch Anfügen des Revisionsbezeichners können für einzelne Revisionen unterschiedliche Properties übergeben werden. (Bspl. *MsiInstallProperties_002=USER=Admin*)
Beispiel	MsiInstallProperties=SERVER=SB0004

Key	**MsiRepairProperties**
Einsatz	Optional
Beschreibung	*Public Windows Installer Properties* können so für die **Reparatur** übergeben werden. Zwischen zwei Properties ist ein Leerzeichen einzufügen. *Public Properties* sind in Grossbuchstaben zu verwenden. **Achtung:** Durch Anfügen des Revisionsbezeichners können für einzelne Revisionen unterschiedliche Properties übergeben werden. (Bspl. *MsiRepairProperties_002=USER=Admin*)
Beispiel	MsiRepairProperties=REINSTALLMODE=**v**omus (Das *MSI* wird recached)

Key	**MsiRemoveProperties (Section Remove)**
Einsatz	Optional
Beschreibung	*Public Windows Installer Properties* können so für den **Remove** übergeben werden. Zwischen zwei Properties ist ein Leerzeichen einzufügen. *Public Properties* sind in Grossbuchstaben zu verwenden. **Achtung:** Durch Anfügen des Revisionsbezeichners können für einzelne Revisionen unterschiedliche Properties übergeben werden. (Bspl. *MsiRemoveProperties_002=USER=Admin*)
Beispiel	MsiRemoveProperties=SERVERDELETE=SB0004

Key	**MsiPatchProperties**
Einsatz	Optional
Beschreibung	*Public Windows Installer Properties* können so für den **Patch** übergeben werden. Zwischen zwei Properties ist ein Leerzeichen einzufügen. *Public Properties* sind in Grossbuchstaben zu verwenden. Standardmässig werden für die Anwendung von Patches folgende Properties verwendet (falls nicht angegeben): REINSTALL=ALL REINSTALLMODE=omus **Achtung:** Durch Anfügen des Revisionsbezeichners können für einzelne Revisionen unterschiedliche Properties übergeben werden. (Bspl. *MsiPatchProperties_002=USER=Admin*)
Beispiel	MsiPatchProperties=REINSTALL=ALL REINSTALLMODE=omus INSTALLDIR=D:\

Key	UpgradeExitIfFailed
Einsatz	Optional (True¦False)
Beschreibung	Wenn im Rahmen eines *Package-Upgrades* eine vorher zu installierende Anwendung bei der Deinstallation einen Fehler auslöst, wird ohne Angabe dieses Bezeichners, die Installation des fokussierten Pakets weiter fortgesetzt. Sollte der Softwarepaketierer hingegen einen Abbruch der Installation mit Meldung im *History.LOG* wünschen, dann setzt er den Eintrag auf 'True'
Beispiel	UpgradeExitIfFailed =True

Key	PlatformLangChange
Einsatz	Optional
Beschreibung	Wird dieser Bezeichner in Form von *PlatformLangChange=True* implementiert, kann folgende Fehlermeldung bei einer Erweiterung eines Platform- oder Sprachverzeichnisses unterdrückt werden: "This package is installed in other language. Please remove old package first!" Das macht dann Sinn, wenn nach Einführung und Verteilung des RTM-Paketes durch den Softwarepaketierer nachträglich neue Plattform- oder Sprachordner hinzugefügt werden, die kompatibel zur Vorgängerrevision der anderen Plattform, bzw. Sprache sind. (Achtung: Deinstallation prüfen, nachdem auf einem Gerät die alte Ausprägung des Pakets mit der neuen Ausprägung geupdatet wurde).
Beispiel	PlatformLangChange=True

Key	UpgradeProperty
Einsatz	Optional
Beschreibung	Bei einem *Package-Upgrade* können für die DeInstallation des alten Paketes zusätzliche Properties übergeben werden. Diese sind diesem Bezeichner anzufügen.
Beispiel	UpgradeProperty=REBOOT=ReallySuppress

Key	UpgradeInRevision
Einsatz	Optional (True¦False)
Beschreibung	Soll eine *Revision* einen *Major-Upgrade* der *Revision* 001 durchführen und wurde in der *Revision* 001 ein *Pre-* oder *PostInstall* verwendet, so verhindert dieser Bezeichner, wenn er auf *True* gesetzt ist, dass beim Installieren der neuen *Revision* die Inhalte aus dem *Scripts*, *Security* und *Cache* Verzeichnisses im Rahmen des Upgrades gelöscht werden. INSTALL 001 INSTALL 002 *REMOVE 001 (Inst. der Rev.003, die ein Major-Upgrade ausführt)* *INSTALL 003*
Beispiel	UpgradeInRevision=True

Key	**ScriptAdditionalUpgrades**
Einsatz	Optional
Beschreibung	**Hierbei handelt es sich um eine Anweisung für das Script *AddProperties.vbs*!** Unter dieser Property kann der Paketierer Namen von Softwarepaketen anfügen, die nach den Richtlinien für *Package-Launcher v2.2* Pakete erstellt wurden. Die *UpgradeCodes* dieser Namen werden durch *AddProperties.vbs* in der *MSI/MST* der *Revision* 001 unter der *Upgrade*-Tabelle angefügt, um zusätzlich beim *Major-Upgrade* fremde Produkte zu deinstallieren. Auf die Versionsangabe kann optional verzichtet werden – Dann werden alle Versions-Verzeichnisse berücksichtigt. Achtung: Beim standardmässig verwendeten *Package-Upgrade* kann auf den Einsatz dieses Bezeichners verzichtet werden!
Beispiel	ScriptAdditionalUpgrades=Byron-HIP

Key	**ScriptUpgradeCode**
Einsatz	Erforderlich
Beschreibung	**Hierbei handelt es sich um eine Anweisung für das Script *AddProperties.vbs*!** Der Eintrag bestimmt den *UpgradeCode* des Produktes. Alle Produkt*revisionen* und alle Produktversionen, also die komplette Produktfamilie verwenden den selben *UpgradeCode*. Dieser Eintrag wird durch das Script *AddProperties.vbs* erstellt und sollte (ausser mit ‚False' nicht verändert oder gelöscht werden!! **Wird der Bezeichner auf 'False' gesetzt, wird der UpgradeCode im Original belassen!**
Beispiel	ScriptUpgradeCode={1F6435C5-429D-42E5-B0B7-CBEAEE66EA0F}

Key	**MultiVariants**
Einsatz	Optional (True¦False)
Beschreibung	Wird ein *Instanzvarianten*paket erstellt, welches die Installation von verschiedenen Ausprägungen von ein und demselben Paket unterstützt (siehe Kapitel *7.1.2 Instanzvarianten*), dann ist dieser Bezeichner auf ‚True' zu setzen.
Beispiel	MultiVariants=True

Key	**ReadMsiSectionPropertiesByREMOVE**
Einsatz	Optional (True¦False)
Beschreibung	Standardmässig werden *Sectionproperties* bei der DeInstallation von *Windows Installer MSI*-Dateien nicht eingelesen. Wird dieser Bezeichner auf ‚True' gesetzt, werden die *Windows Installer Properties* auch bei der Deinstallation eingelesen und an *msiexec.exe* weitergegeben. Der Bezeichner existiert nur aus Gründen der Abwärtskompatibilität.
Beispiel	ReadMsiSectionPropertiesByREMOVE=True

Key	ExecuteFile
Einsatz	Optional
Beschreibung	Mit diesem Bezeichner ist es möglich, Scripts im Revisionsverzeichnis des Softwarepakets oder ein lokales Programm auszuführen. Als Scripts können *.cmd, *.bat, *.ps1, *.vbs oder andere Varianten verwendet werden. Erforderlich ist, dass dem System die Dateierweiterung bekannt ist. Jeder im Script ausgewiesene Rückgabewert <> (0 oder 3010 oder 1641) wird vom *Package-Launcher* als fehlerhafte Ausführung interpretiert und mit einer Fehlermeldung im *History.LOG* quittiert. Im Zweifelsfall ist der Rückgabewert als letzte Aktion im Script durch den Paketierer festzulegen (bspl. *Exit 0* bei einem CMD). Sollen mehrere einzelne Steps mit einer Rückgabewertprüfung installiert werden, kann das vorgesehene Script in mehrere Einzelscripts aufgeteilt werden: Bspl: ExecuteFile_**001**=install.cmd ExecuteFile_**002**=install.cmd Oder auch nur… ExecuteFile=install.cmd, wenn ein einziges install.cmd in verschiedenen Revisionsverzeichnissen abgelegt wird, welches chronologisch zum Einsatz gelangen soll In der INI-Datei werden neben Standard-Umgebungsvariablen folgende Variablen unterstützt (werden *hier* ausführlich beschrieben): %SOURCE%, %CACHE%, %PACKAGE%, %REVISION%, %LANGUAGE%, %LOGFILE%, %VARIANTSECTION%, %VARIANT%, %SCRIPTS, %DEPAGENCY%, %TASK%, %INI% **%SOURCE%** enthält den Sourcepfad der aktuellen Installation, bzw. deren Installation. Bspl. *C:\Windows\ccmcache\4\ x86-ML\002*. Wird *CopyLocal=True* verwendet, dann wird der zwischengespeicherte und für die Installation massgebliche Pfad zurückgegeben: Bspl. *C:\Windows\Package-Launcher\Cache\TWI-FBMS-1.0\x86-ML\002.* **%CACHE%** enthält bei Verwendung von *CopyLocal=True* das *Revisions*-verzeichnis des zwischengespeicherten Pfades. Wird *CopyLocal=True* nicht verwendet, ist die Variable leer. Der Bezeichner *ExecuteFile* kann in der *[Install]*- und *[Uninstall]*-Section verwendet werden. **Achtung:** Bei Verwendung eines im Paketverzeichnis abgelegten Scripts oder verwiesenen Daten, die im Paketverzeichnis abgelegt sind, auf welche im Rahmen einer **Deinstallation** zugegriffen werden soll, ist zusätzlich die Anweisung *CopyLocal=True* notwendig. **Achtung:** Wird keine *Revision* am Bezeichner angegeben, so wird in allen vorgefundenen *Revisions*verzeichnissen nach dem Namen der in der INI angegebenen Datei gesucht. Werden in *Revisions*verzeichnissen unterschiedliche Namen von Scripts verwendet, so kann dies mit dem Bezeichner *ExecuteFile_**00x**=<Name.ext>* bewerkstelligt werden. Es wird generell der Einsatz des Revisionspostfixes empfohlen.

Beispiel	ExecuteFile=install.cmd "%CACHE%\Files" ExecuteFile_003=installx.cmd "%ProgramFiles%\Datango" ExecuteFile_004=regedit /s "%SOURCE%\test.reg"

Key	PreScript/PostScript
Einsatz	Optional
Beschreibung	Mit diesem Bezeichner ist es ebenfalls möglich, Scripts im Revisionsverzeichnis des Softwarepakets oder ein lokales Programm auszuführen. Es gelten die selben Regeln wie beim Bezeichner *ExecuteFile*. (Der Bezeichner *PreScript* darf **nicht zusammen mit *ExecuteFile*** verwendet werden (entweder oder). ***Pre**Script* führt ein Script aus, welches vom *Package-Launcher* **vor** einer Installation von unterstützten *MSI/MSP/MSU/APPV*-Ressourcen im selben Revisionsverzeichnis zur Ausführung gelangt. Während das über ***Post**Script* deklarierte Script **im Anschluss** einer unterstützten Ressource, die im selben Revisionsverzeichnis abgelegt ist, ausgeführt wird. In Scripts und der INI-Datei können alle Umgebungsvariablen verwendet werden. Zusätzlich stehen zum Ausführungszeitpunkt weitere Variablen zur Verfügung *(%CACHE%, %REVISION%, %PACKAGE%, %LOGFILE%,* etc.). Es kann auch auf lokal installierte Programme zugegriffen werden, indem diese als Einzeiler integriert werden.
Beispiele	[Install] PostScript_001=config.cmd PreScript_002=REG.exe ADD HKLM\Software\Lithium /v WarpSpeed /d "1" /f PostScript_003=%WINDIR%\sysnative\cmd.exe /c "C:\Program Files\XF\hf.exe" PostScript_004=powershell.exe "%CACHE%\MyScript.ps1" [UnInstall] ExecuteFile=cmd /c exit

Key	ActiveSetupScript_00x (Section Install)
Einsatz	Optional
Beschreibung	Hier kann ein Script oder lokales Programm angegeben werden, welches ausgeführt werden soll. Dieses Script wird in der Registry in *HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components* im Active-Setup eingetragen. So sind Benutzerressourcen applizierbar. Bei Verwendung eines im Paketverzeichnis abgelegten Scripts, ist zusätzlich die Anweisung *CopyLocal=True* notwendig. **In der INI-Datei** werden neben Standard-Umgebungsvariablen folgende Variablen unterstützt (werden *hier* ausführlich beschrieben): %SOURCE%, %CACHE%, %PACKAGE%, %REVISION%, %LANGUAGE%, %LOGFILE%, %VARIANTSECTION%, VARIANT%, %SCRIPTS, %DEPAGENCY%, %TASK%, %INI% **%SOURCE%** enthält den Sourcepfad der aktuellen Installation, bzw. deren Installation. Bspl. *C:\Windows\ccmcache\4\ x86-ML\002*. Wird *CopyLocal=True* verwendet, dann wird der zwischengespeicherte und für die Installation massgebliche Pfad zurückgegeben: Bspl. *C:\Windows\Package-Launcher\Cache\TWI-FBMS-1.0\x86-ML\002.*

	%CACHE% enthält bei Verwendung von *CopyLocal=True* das *Revisions*-verzeichnis des zwischengespeicherten Pfades. Wird *CopyLocal=True* nicht verwendet, ist die Variable leer. **In einem Script** werden diese Umgebungsvariablen nicht direkt unterstützt, können aber via Kommandozeilenparameter dem Script bekannt gemacht werden. Der Revisionstoken _00x ist für diesen Bezeichner zwingend!
Beispiel	ActiveSetupScript_001="%CACHE%\myscript.cmd" "%PACKAGE%"

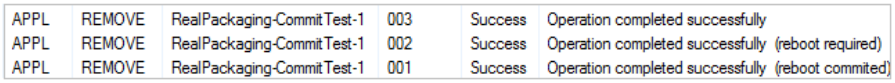
Key	**EnableMSU (Section Install & UnInstall)**
Einsatz	Optional (True¦False)
Beschreibung	Durch diesen Bezeichner werden Hotfixe in Form von *MSU*-Dateien direkt unterstützt. Ab August 2012 wird der Bezeichner standardmässig implementiert. Für alte Pakete, wo .*MSU*-Dateien durch ein *PreInstall* ausgeführt werden, ist der Bezeichner auf *False* zu stellen. Möchte man die Deinstallation des Hotfixes im Rahmen der Paketdeinstallation nicht ausführen, so kann der Bezeichner in der *UnInstall*-Sektion entfernt oder mit False gleichgesetzt werden. Sollen MSU-Pakete später wieder deinstalliert werden, ist die Angabe von *CopyLocal=True* erforderlich.
Beispiel	EnableMSU=True

Key	**REPAIR**
Einsatz	Optional (Enable¦Disable¦DisableWithRetValue0)
Beschreibung	Soll eine Reparatur in einem Paket nicht oder nicht mehr unterstützt werden, so kann dies mittels dem neuen Bezeichner *REPAIR* in der INI-Datei des Softwarepakets realisiert werden. So sind auch nachträglich in einer künftigen *Revision* entstehende Anforderungen umsetzbar, indem einfach der Bezeichner der *Install*-Sektion hinzugefügt wird. Ist *REPAIR=Disable*, dann wird der Text *„REPAIR is not supported!"* in die Datei *History.LOG* geschrieben. Der Zuweisungswert *DisableWithRetValue0* veranlasst hierbei, dass dem übergeordneten Prozess (SW-Verteilungswerkzeug) der Exit-Code 0 zurückgegeben wird.
Beispiel	REPAIR=Disable

Key	**CheckProcesses (Section Install & UnInstall)**
Einsatz	Optional
Beschreibung	Prüft vor der Transaktion, ob der oder die angegebenen Prozesse gestartet sind. Ist bei der Installation einer der angegebenen Prozesse gestartet, so wird dies in der Datei *History.LOG* quittiert und der *Exit-Code* 1618 zurückgegeben. Dieser definiert in *SCCM CB* den *Fast Retry exit code*, was *SCCM CB* dazu veranlasst, das Paket vor einem *policy refresh* nochmals zu installieren. Der Bezeichner soll nur Prozessnamen ohne Pfadangaben enthalten. Mehrere Prozesse können wie gewohnt durch ein Leerzeichen voneinander getrennt werden. Prozessnamen mit Leerzeichen sind in Anführungszeichen einzufassen. Der Bezeichner ist taskabhängig, kann also in der *Install* und der *UnInstall*-Sektion angewendet werden.
Beispiel	Checkprocesses=vmware.exe vmware-tray.exe "vmware special.exe"

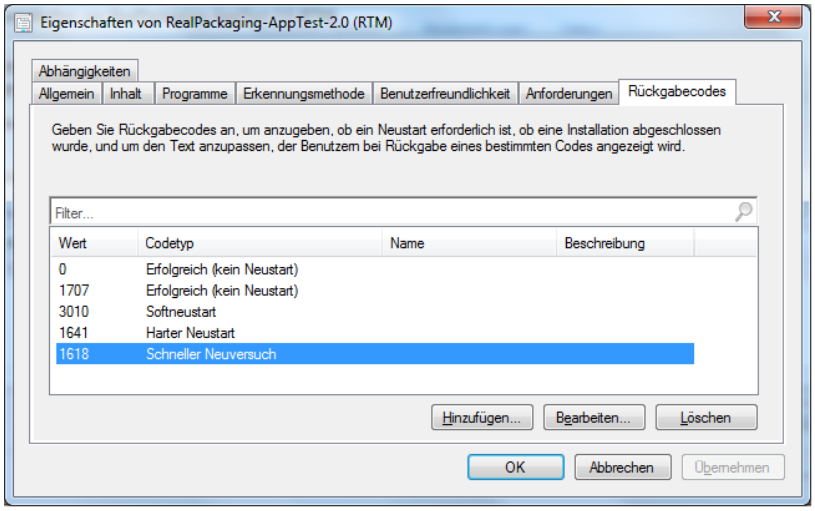
Key	CommitRebootExitCode (Section Install & UnInstall)
Einsatz	Optional (False¦True¦Enhanced)

Beschreibung	Wird *CommitRebootExitCode=True* gesetzt, so werden alle 3010er (unterdrückter Neustart) und 1641er-Rückgabewerte (initiierter Neustart) von *msiexec* dem übergeordneten Prozess (SW-Verteilungswerkzeug) zum selbständigen Handling zurückgegeben. Auch Rückgabewerte in Scripts (3010 und 1641) werden so durchgängig an das Softwareverteilungswerkzeug übermittelt.
	
	Mit *CommitRebootExitCode=Enhanced* wird auch bei angeforderten Neustarts via dem Bezeichner *Reboot=True* der Rückgabewert 3010 zurückgegeben, unabhängig davon, ob der ausgeführte Prozess (bspl. MSI-Installation) effektiv mit 3010 beendet hat oder nicht.
	
	Der Bezeichner kann **in der INI** unterschiedliche Ausprägung **je Revision** beinhalten:
	
	`CommitRebootExitCode=True`
	`CommitRebootExitCode_002=False`
	
	In dem obigen Beispiel wird für alle *Revisionen CommitRebootExitCode* auf *True* gestellt, ausser für *Revision* 002.
	
	Implementation:
	
	Der Bezeichner *CommitRebootExitCode* kann neben der INI auch über die Kommandozeile oder global in der Registry für alle Softwarepakete implementiert werden (Registry mit Vorsicht zu geniessen!). Die Gewinnhierarchie ist folgendermassen:
	
	1. INI
	2. Registry (gewinnt über INI)
	3. Kommandozeile (gewinnt über alles)
	
	In der Registry wäre der Wert unter *HKLM\Software\Real Packaging\Package-Launcher\CommitRebootExitCode* abzulegen.
	
	Verhalten in Revisionen bei der Installation:
	
	Wird bei der Installation *CommitRebootExitCode* auf *True* gestellt und beendet ein Prozess in einer *Revision* **zwingend vor dem Ende** (letzte Revision muss 0 zurückgeben) aller *Revisionen* (Script oder MSI) mit 3010 oder 1641, so wird noch der Eintrag in der *History.LOG-Datei* erstellt,…
	
	`Success  Operation completed successfully (reboot commited, installation not completed)`
	
	… in Erwartung, dass nach einem Neustart, ausgeführt durch SCCM, und nach Feststellung, dass die lokale Ermittlung der *Detection-Method* noch nicht den erwarteten Wert ergibt (bspl. Rev >=005), den Installationsprozess neu startet. Bei diesem zweiten Installationsprozess werden nur noch die fehlenden Revisionen nachgefahren. Dieses Verhalten zeigt sich nur in Installationsprozessen, nicht aber bei der Deinstallation. Eine Deinstallation wird immer vollständig ausgeführt (über alle Revisionen), bevor 3010 oder 1641 an das Softwareverteilungswerkzeug zurückgegeben wird.

	Upgrade alter Paketversionen mit CommitRebootExitCode=True: Wird ein Upgrade (Entfernen alter Version) aus einem Paket ausgelöst, welches den paketweiten Bezeichner *CommitRevbootExitCode=True* in der **Install**-Section enthält, so wird für den Upgrade dieser Bezeichner vererbt. Der Upgrade deinstalliert dann **vollständig**, auch wenn dort 3010 oder 1641 in einer Zwischenrevision zurückgegeben wird.  In diesem Beispiel wird von Revision 002 3010 zurückgegeben (reboot required). Der *Package-Launcher* deinstalliert aber vollständig bis inklusive Rev. 001, bevor er selbst 3010 an SCCM zurückmeldet (reboot commited) **Platzierung in der INI-Datei:** Für Installationen (inkl. Upgrades) ist der Bezeichner in der Install-Section zu platzieren. Nur für Deinstallationen notwendig zu übergebende Rückgabewerte platziert der Softwarepaketierer den Bezeichner in der UnInstall-Section.
Beispiel	[Install] CommitRebootExitCode=True

Key	**UpgradeAfterPreInst**
Einsatz	Optional (True¦False)
Beschreibung	Sollen bei einer Paketinstallation im Rahmen eines Upgrades noch **VOR** einer allfälligen Deinstallation eines Vorproduktes, gewisse programmatische Aufgaben ausgeführt werden, so kann dies im neuen Softwarepaket mittels einer herkömmlichen *PreInstall_**001**.EXE* oder *PreScript_001* realisiert werden, wenn zusätzlich der Bezeichner *UpgradeAfterPreInst=True* in der INI-Datei angegeben wurde. Dadurch wird sichergestellt, dass chronologisch zuerst das *PreInstall_001.exe/PreScript_001* (aus Rev. 001) ausgeführt und erst danach die Deinstallation des Vorgängerproduktes über den Upgrade ausgeführt wird.
Beispiel	UpgradeAfterPreInst=True

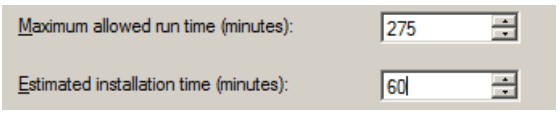
Key	**AllowDowngrade (Section Install)**
Einsatz	Optional (False¦True¦Enhanced)
Beschreibung	Standardmässig steht diese Einstellung auf False – auch wenn der Bezeichner fehlt. So werden Downgrades von Paketen verhindert, indem der Downgrade mit der folgenden History-Meldung quittiert wird: *This package doesn't support to downgrade! (Upgrade targets 'PackageName') (exit with return code 0)* Wird eine nicht konforme Versionierung verwendet, so kann die obige Fehlermeldung verhindert werden, indem in der Install-Section des neuen Pakets *AllowDowngrade=True* angegeben wird.
Beispiel	AllowDowngrade=True

Key	**FastRetry**
Einsatz	Optional (number)
Beschreibung	Unter *SCCM CB* kann mittels des *Fast Retry Return value* im *Deployment Type* automatisch eine Neuinstallation der Application vorgenommen werden, ohne auf den nächsten *Deployment Evaluation Cycle* warten zu müssen.  Der *Package-Launcher* unterstützt ein solches Verhalten bei einigen Fehlermeldungen. Unter anderem bei folgenden History.LOG-Meldungen: • *The program cannot be started several times.* • *Other Windows Installer process in progress! Try again later.* • *Running processes not allowed! (%processname%)* • *Another installation is already in progress! (Exitcode: xx)* Standardmässig wird in solchen Fällen '1618' an *SCCM* zurückgegeben. Soll der Rückgabecode ändern, kann dieser durch den besprochenen Bezeichner abgeändert werden.
Beispiel	FastRetry=1620

Key	**enforceREMOVE (Section Install)**
Einsatz	Optional (True¦False)
Beschreibung	Mit diesem Bezeichner kann ein aus der Kommandozeile bekanntes /e im Zusammenhang mit einer Deinstallation oder einem Upgrade direkt im Paket vorgesehen werden. Übersteuerung der Fehlermeldung „*REMOVE action is only valid for products which are currently installed*“. D.h. Deinstallation wird weitergeführt auch wenn die Source zur Deinstallation fehlt. Mit *EnforceREMOVE=True* wird sichergestellt, dass ein allfälliges Paket mit Revisionslücken (MSI basierend) dennoch über einen REMOVE entfernt werden kann. Revisionslücken können beispielsweise durch einen unkontrollierten Upgrade (bslpl. WSUS) entstehen.

Key	AppVAutoStopProcesses (Section Install)
Einsatz	Optional (True¦False), ohne Angabe ist die Einstellung ‚True'
Beschreibung	Standardmässig wird eine virtuelle App-V 5 Zielanwendung, die in Benutzung steht, vor einem REMOVE geschlossen, damit die Transaktion erfolgreich abgeschlossen werden kann. Soll dies verhindert werden, kann mit *AppVAutoStopProcesses=False* bewerkstelligt werden.
Beispiel	AppVAutoStopProcesses=False

Key	AppShutdown (Section Install & UnInstall)
Einsatz	Optional, Pattern: Per Default ausgeschaltet, ausser *f:Files*
Beschreibung	Der Bezeichner *AppShutdown* kann in der *Install* und *UnInstall*-Sektion angewendet werden. Fehlt dieser bei einem REMOVE in der UnInstall-Sektion, wird auch dort die *Install*-Eigenschaft angewendet. Die Eigenschaft hat vier Objektausprägungen. Diese steuern das Verhalten im Zusammenhang mit Dateien die zu aktualisieren sind, die aber während der Installation vom Benutzer noch in Benutzung stehen: [DontAskForSave|**AskForSave**|Force];[DisableUserDisableShutdown];ALL|**MSI**;[[f:]Files]|False Objektoptionen Fett geschriebene Texte sind Standardeinstellungen, die verwendet werden, wenn die Oder-Ausprägung in der Deklaration fehlt. ***ALL*** behandelt alle geöffneten Benutzerprogramme, fragt den Benutzer zum Schliessen und setzt die Installation nach Bestätigung fort. Während der Installation bleiben Startmenü, Desktop und Taskleiste verborgen. ***MSI*** behandelt alle in der beabsichtigten Transaktion zu installierenden oder deinstallierenden Exe-Dateien, die in den MSI-Dateien referenziert sind. Das weitere Vorgehen entspricht dem von *ALL*. Der Desktop wird aber angezeigt. ***[f:]Files:*** Hier können Dateien angefügt werden (datei1.exe;datei2.exe; datei3.exe; ...), die geschlossen werden sollen. Einzelne Dateiangaben können mit dem Pattern *MSI* kombiniert werden AppShutdown=**MSI**;file1.exe;file2.exe;…. Der Parameter "f:" für 'force' vor der Datei bedeutet, dass die Datei.exe forciert ohne Dialog gekillt wird. In diesem Fall kann es sich auch um einen Systemprozess oder einen Prozess, deren Schliessen erhöhte Rechte erfordert, handeln! ***False***: Ohne jegliche Angabe in der INI, der Registry oder in der Kommandozeile, verwendet der *Package-Launcher AppShutdown=MSI*. Soll grundsätzlich kein Dialog erscheinen, lässt sich das mit *AppShutdown=False* bewerkstelligen.
Beispiel	[Install] AppShutdown= f:myfile.exe;f:AWStream.exe

Key	**MaxExecuteTimeRTM und ExecuteTimeRTM MaxExecuteTime_00x und ExecuteTime_00x**
Einsatz	Optional, Pattern, Section Install
Beschreibung	Mit den Bezeichner *MaxExecuteTimeRTM* und *ExecuteTimeRTM*, deklariert in der *Install*-Section, können die folgenden Optionen im *Deployment Type* (*User Experiance*) beim Überführen des Paketes in SCCM für *(RTM)-Applications* vorgegeben, bzw. die Defaulteinstellungen (275/60) übersteuert werden.  Eine Anpassung des Wertes *MaxExecuteTimeRTM* macht nur beim Einsatz von *SCCM Wartungsfenster* Sinn. Hingegen dient der Eintrag *ExecuteTimeRTM* zur Anzeige im Softwarecenter und kann individuell pro Paket abgefüllt werden. Die Bezeichner *MaxExecuteTime_00x* und *ExecuteTime_00x* dienen demselben Ziel, mit dem Unterschied, dass diese für die *Applications* der *Revisionsupdates* vorgesehen sind. Die Spannbreite bewegt sich zwischen 15 – 720 Minuten.
Beispiel	[Install] MaxExecuteTime_004=60 ExecuteTime_004=40

Key	**Build, AutoBuild**
Einsatz	Optional, (Build=Number, AutoBuild=True¦False), Section Install, Default=nicht enthalten
Beschreibung	Die *Build*nummer wird normalerweise über die .BLD-Datei geführt. Will man den *Build* automatisch bei jeder produktiven Überführung mit *SCCMCreateApp* erhöhen, kann dies mittels der Integration der *Build*nummer in der INI bewerkstelligt werden, wenn zusätzlich *AutoBuild=True* festgelegt ist. Seien Sie vorsichtig und verwenden Sie *Build*-Informationen der Transparenz wegen nicht an beiden Orten (INI und .BLD-Datei).
Beispiel	[Install] AutoBuild=True Build=004

Key	**DMBuild**
Einsatz	Optional, (True¦False), Section Install, Default=False
Beschreibung	Mittels dem Bezeichner *DMBuild=True* in der INI-Datei und der Applizierung einer *Build*-Datei in einer RTM-*Revision* (eine *Revision*, die bei der produktiven Erstüberführung im Paket enthalten war), lässt sich ein ***Inplace-Update*** durch ein REPAIR der *RTM-App* auslösen. Zu diesem Zweck wird die SCCM-Detection der *RTM-Application* mit der Prüfung der *Build*-Nummer erweitert.

	Es ist vorgängig zu prüfen, ob die Aktualisierung die Anforderungen erfüllt und ob alle notwendigen Ressourcen durch den Aktualisierungsprozess aktualisiert werden (manueller Test der einszelnen Schritte). Das Paket *RealPackaging-PackageLauncher-2020* ist ein solches Paket, welches diese Mechanismen verwendet. Es enthält nur eine *Revision* 001, welche bei Bedarf mit neuen Ressourcen aktualisiert wird. Nur für solche Vorhaben, wo die Anforderungen einer Aktualisierung über ein REPAIR des Pakets vollständig umgesetzt werden können, eignet sich dieser Bezeichner!
Beispiel	[Install] DMBuild=True

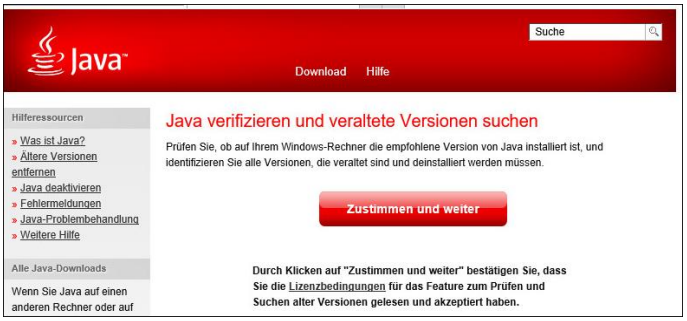
Key	CopyOnlyScriptsOnBuild
Einsatz	Optional, (Revisionsangabe), Section Install
Beschreibung	Mittels dem Bezeichner *CopyOnlyScriptsOnBuild=00x* in der *Install*-Section der INI-Datei, lassen sich folgende Bedürfnisse abdecken: • die INI-Datei kopieren • den lokalen Cache aller RTM-*Revisionen* (nur diese) aktualisieren • alle Scripts der RTM-*Revisionen* unter %PL%\..\Scripts zu aktualisieren… Die Software wird dabei nicht installiert oder repariert. Es ist erforderlich, dass auch DMBuild=True gesetzt ist.
Beispiel	[Install] DMBuild=True CopyOnlyScriptsOnBuild=004

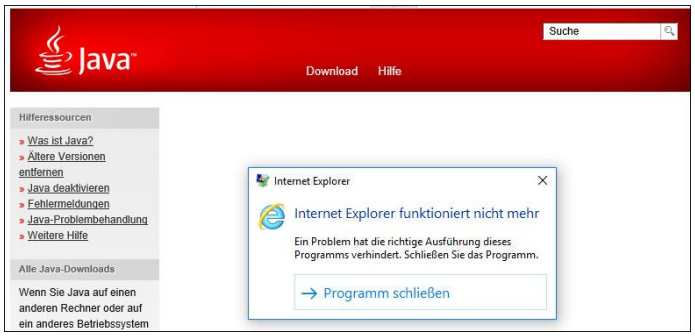
Key	AvailableForUnversionedApps
Einsatz	Optional, (True¦False), Section Install, Default=True
Beschreibung	Dieser Bezeichner steuert das Verhalten für SCCMCreateApp, ob unversionierte Applications ein Available oder Required Deployment erhalten sollen. Ohne Angabe wird standardmässig ein Available Deployment erstellt.
Beispiel	[Install] AvailableForUnversionedApps=False

Key	DisableAppvDynamicVirtualization
Einsatz	Optional, (True¦False), Section Install, Default=False
Beschreibung	Nach der Installation und anschliessendem Start einer AppV-Anwendung mit Java-Inhalten, kann im Anschluss im IE oder Browser kein Java-Applet mehr gestartet werden, bzw. es wird dann auf die AppV-Instanz mit gegebenenfalls veralteter Java verwiesen. Dies kann Auswirkungen auf Java Applikationen haben, wenn diese über den Internetbrowser geöffnet werden sollen. Solche zeigen dann ein unerwartetes

Verhalten – angefangen von Nichtanzeige der Seite/Applikation, zu unspezifischen Fehlermeldungen, bis hin zu einem Absturz der Seite / Anwendung.

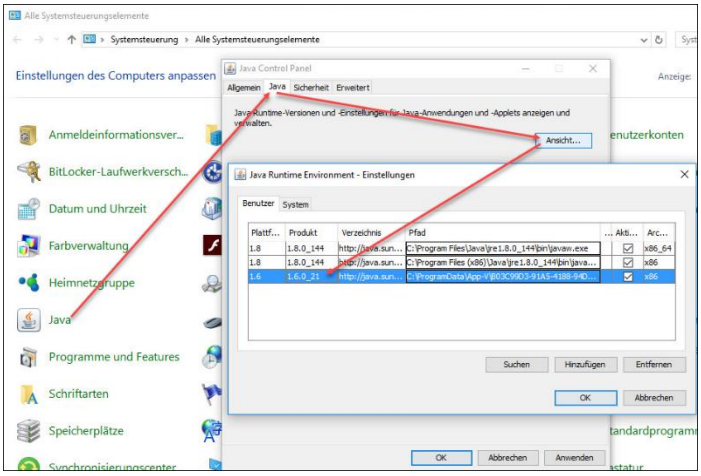
Prüfen lässt sich das Fehlverhalten, wenn **nach einmaligem Start der AppV-Anwendung** mit veraltetem Java im **physischen IE** über www.java.com/verify/ die installierte Javaversion angezeigt werden soll:





Hier verabschiedet sich der IE und stürzt gleich ab.

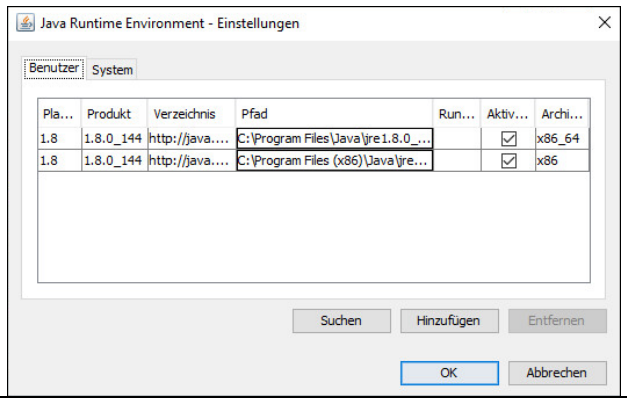
Zudem findet sich danach **nach Neustart** im Javacpl.cpl ein zusätzlicher Eintrag über die veraltete App-V Instanz:



Wird in der INI-Datei DisableAppvDynamicVirtualization=True gesetzt, bewirkt dies das zwischenzeitliche Setzen des folgenden Registrykeys:

```
HKLM\SOFTWARE\Microsoft\AppV\Client\Virtualization
EnableDynamicVirtualization="0"
```

Danach zeigt sich nach dem Testablauf wie oben im IE das folgende Bild…

	 …und nach einem Neustart im Javacpl.cpl die folgenden lokalen Versionen: 
Beispiel	[Install] DisableAppvDynamicVirtualization=True

Key	**UpgradePreRevision**
Einsatz	Optional
Beschreibung	Diese Eigenschaft kann gleich verwendet werden wie der *Upgrade* Bezeichner. Hinter *UpgradePreRevision* angegebene Pakete werden auch im Rahmen eines Revisionsupdate vor dem Update (bei Fehler und *UpgradeExitIfFailed=False* einmalig) entfernt, während der *Upgrade* das Entfernen von Paketen nur vor der Erstinstallation anbietet. Die Funktion kann verwendet werden, wenn im Rahmen eines Revisionsupdates Pakete aus einer anderen Paketfamilie vorgängig entfernt werden sollen.
Beispiel	UpgradePreRevision= dVelop-d3.Control-

Key	**ARPSYSTEMCOMPONENT**
Einsatz	Optional, (0¦1¦False), Section Install, Default=False
Beschreibung	Standardmässig erscheinen alle *Revisionen* > 001 nicht im *Add/Remove Program*. Sollen einzelne MSI-Pakete aus Revisionen > 001 dennoch angezeigt werden, kann dies mit ARPSYSTEMCOMPONENT=0 bewerkstelligt werden. Wird ARPSYSTEMCOMPONENT=False angegeben, so modifiziert *AddProperties.vbs* den Eintrag nicht und belässt den in der MSI vorgesehene Einstellung.
Beispiel	ARPSYSTEMCOMPONENT=0

Key	PushActiveSetup
Einsatz	Optional, (True¦False), Section Install
Beschreibung	Standardmässig werden die Inhalte aus der Datei *ActiveSetup_00x.cmd* im Rahmen eines Microsoft Active Setups sofort für alle angemeldeten Benutzer ausgeführt. Mit *PushActiveSetup=False* kann die sofortige Ausführung verhindert werden und die Einstellungen werden dann erst beim Neuanmelden appliziert.
Beispiel	PushActiveSetup=False

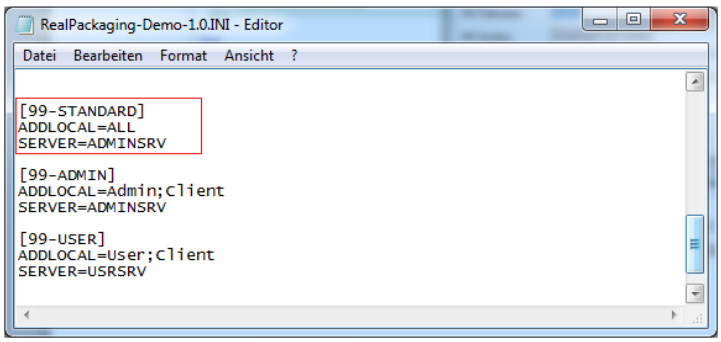
Key	SuccessErrorString (Section Install & UnInstall)
Einsatz	Optional, (True¦False), Section Install & UnInstall
Beschreibung	Der Bezeichner 'SuccessErrorString' ermöglicht, gewisse Fehlermeldungen als Successcode zu deklarieren, damit diese nicht zu einem Fehler führen. In diesen Fällen wird der Rückgabewert '0' an SCCM zurückgegeben.
Beispiel	SuccessErrorString=Install is not needed because no updates are applicable!

3.5.2 Sectionproperties

Neben den Implementationen in *MsiInstallProperties* (siehe letztes Kapitel) ist es auch möglich, *Property*ausführungen über sogenannte *Sectionproperties* zu gestalten. Diese erlauben insbesondere mit *Paketvarianten* (siehe Kapitel *7.1 Paketvarianten*) eine differenziertere Steuerung von *Windows Installer Properties*. Die Sektion *99-STANDARD* wird für alle Standardpakete ohne Variantenbezeichner eingelesen und die dort abgebildeten *Properties* zum Installationszeitpunkt allen im Paket platzierten MSI's übergeben. Auf die massgebliche Sektion kann aber auch in *WiseScript* und in allen anderen Scripts über die Variable *%VARIANTSECTION%* zugegriffen werden. Dadurch ist es möglich, die dort platzierten Variablen einzulesen und auszuwerten. Es können zudem bedingte Anweisungen in Scripts durch die Ausprägung der Variablen *%VARIANTSECTION%* realisiert werden (bspl. *IF "%VARIANTSECTION%"=="99-ADMIN" GOTO ADMIN*)

Sind *Properties* erforderlich, die ein oder mehrere Leerzeichen enthalten, so ist der Propertyinhalt in Anführungszeichen einzufassen.

Bspl. CONFIG= "User 2"



3.6 Realisierung eines Upgrades

Es gibt zwei Verfahren, um einen *Upgrade* auszulösen. Den *Major-Upgrade*, der sich vornehmlich der Techniken der *Windows Installer* Technologie bedient und der *Package-Upgrade*, welcher die Deinstallation von Softwarepaketen selber verwaltet. Das **dringend empfohlene** und für den Softwarepaketierer einfacher zu realisierende und transparentere Verfahren stellt **der *Package-Upgrade*** dar. Die Steuerung der Art des *Upgrades* wird über die *INI*-Datei des Softwarepakets gesteuert. Die massgeblichen Items sind folgende:

```
Upgrade=<Package-Family>
ScriptAdditionalUpgrades=<Package-Family>
ScriptUpgradeCode={GUID}
```

AddProperties.vbs liest die Zeilen ein und appliziert der *MSI/MST*-Datei in der *Revision* 001 die für den *Major-Upgrade* erforderlichen Erweiterungen, sofern der Eintrag unter *Upgrade* leer ist (und nur dann!). Unter *ScriptUpgradeCode* wird durch *AddProperties.vbs* der zentrale *UpgradeCode* für alle *Revisionen* dieses Softwarepakets und für die gesamte Produktefamilie abgelegt. Jede *Revision* aus der kompletten Produktefamilie verwendet sprach- und Plattformübergreifend den selben *UpgradeCode*!

Achtung: Der Eintrag *ScriptUpgradeCode* sollte durch den Softwarepaketierer nicht manipuliert werden! Das Servicemodell des Scriptes *AddProperties.vbs* verwaltet diesen Eintrag selbständig!

Beim *Package-Upgrade*, dem Standardverfahren, werden in der *INI*-Datei über den Bezeichner *Upgrade* einfach die Namen der Softwarepakete oder Softwarefamilien angefügt, die man im Rahmen des *Upgrades* vorgängig deinstallieren möchte. Hier müssen nicht komplette Bezeichnungen der Softwarepakete ausgeschrieben werden. Es können auch **Teil**strings appliziert werden. Zwischen zwei Anwendungen, bzw. Anwendungsfamilien ist ein Leerzeichen einzufügen:

Bspl:
Upgrade=Byron-HIP Adobe-Reader
(deinstalliert Byron-HIP und Adobe-Reader unabhängig der lokal installierten Version im Rahmen eines *Package-Upgrades*)

Achtung: Das Verfahren des ***Package-Upgrades* ist das bevorzugte Modell** zur Anwendung von *Upgrades*! Als Bezeichner können für nicht *Package-Launcher*-kompatible Softwarepakete auch **{ProductCodes}** angefügt werden!

3.7 Namensrichtlinien

Das Softwarepaket wird in der Regel mit dem Programm *CreatePackage.EXE* erstellt. Dadurch wird gewährleistet, dass die richtige Struktur eingehalten wird. Viele Fehler werden hier bereits abgefangen.

Folgende Regeln müssen generell eingehalten werden:

1. Der komplette Paketnamen darf die Länge von 39 Zeichen nicht überschreiten (*nur SCCM 2007*)
2. Es dürfen keine Leerzeichen im Paketnamen verwendet werden.
3. Das Trennzeichen „-„ darf im Paketnamen nicht verwendet werden.
4. Optioneneinleitungs- und abschlusszeichen "(" und ")" dürfen nicht verwendet werden. Diese werden vom Package-Launcher selbst verwendet. Bspl. *(S)*.

3.7.1 Bezeichnung von MSI und MST-Dateien

Die Namensbezeichnung der *MSI*-Dateien im *Revisions*verzeichnis ist frei. Die Bezeichnung der allgemeinen *Transformationen* (*MST*-Datei) ist ebenfalls frei. Wird durch den *Conflict Explorer 2009* eine Transformation erstellt, so sollte diese *ResolveConflicts.MST* bezeichnet verwenden. Eine so benannte Datei wird dann durch den *Package-Launcher* automatisch als letzte *Transformation* angewendet.

3.7.2 PreInstall und PostInstall

Die *Wise-Scripts* sind nur mit den folgenden Namen im *Revisions*verzeichnis gültig:

*PreInstall_**00x**.EXE* und *PostInstall_**00x**.EXE*, wobei das *00x* mit der tatsächlichen *Revision* anzupassen ist, welches der *Revisions*bezeichnung, bzw. dem Verzeichnisnamen entspricht.

Das selbe gilt für die Dateien *PreInstall_00x.cmd* und *PostInstall_00x.cmd.,* sowie *PreInstall_00x.ps1* und *PostInstall_00x.ps1.*

3.7.3 Bezeichnung des Security-Batch

Eine Vorlage des Securitybatches wird durch *CreatePackage.EXE* in das Verzeichnis *Work* kopiert. Bei dieser Vorlage muss nur noch die *Revision* von *00x* auf die tatsächliche *Revision* angepasst und im Bedarfsfall in das *Revisions*verzeichnis kopiert werden. Folgende Namensbezeichnung ist einzuhalten:

<Package>_00x.CMD, bspl. *Byron-HIP-1.0_**001**.cmd*

3.7.4 Bezeichnung der Build-Datei

Die *Build*-Datei trägt den Namen des Softwarepakets, gefolgt vom *Revisions*-Bezeichner und der Erweiterung *.BLD.*

3.8 Einsatz von eigenen Scripts und Batchprogrammen

Für Installationsanweisungen ausserhalb von MSI-Dateien stehen *WiseScript* oder Standardscripts zur Verfügung. Auf die *WiseScript*-Implementation wird in diesem Dokument nicht eingegangen. Für Standardscriptimplementationen sind zwei verschiedene Umsetzungsvarianten möglich:

1. Scripts, die im *Revisions*verzeichnis abgelegt sind und spezifische Namensrichtlinien einhalten: *PreInstall_00x.**cmd*** oder ***.ps1***, sein Gegenspieler *PostInstall_00x.**cmd*** oder ***.ps1*** und *ActiveSetup_00x.**cmd***. Wobei das "00x" mit der Revision zu ersetzen ist, worin sich das Script befindet. Bspl. *PreInstall_001.cmd.*
2. Scripts, die über die INI-Bezeichner *ExecuteFile, PreScript, PostScript und ActiveSetupScript* ausgelöst werden – siehe Kapitel *3.5.1 Zusammenfassung der wichtigsten INI-Einträge*.

In beiden Fällen ist sorgfältig vorzugehen und es sind zwingend die nachgeführten Richtlinien einzuhalten!

Scripts können zur Ausführung von vor- und nachgeschalteten Anweisungen eingesetzt werden oder zum Ausführen von *Legacy-Setups*. *Legacy-Setups* mit auspackbaren *MSI*-Ressourcen sollten hingegen nicht als EXE-Datei ausgeführt werden. Hier wären stattdessen die ausgepackten Ressourcen zu verwenden (siehe *Kapitel 4.10 Paketierungsarten*).

Auch auf lokale Programme kann über Scripts direkt zugegriffen werden. Zudem werden verschiedene, aus der Installation stammende Umgebungsvariablen bei der Integration mit den Scriptbezeichnern unterstützt, womit bedingte Anweisungen in der Implementation möglich werden.

3.8.1 Richtlinien beim Einsatz von Scripts

1. Verwenden Sie keine Verweise und Anweisungen auf durch den *Package-Launcher* unterstützte Ressourcen (*MSI*, *MSP*, *MST*, *App-V, MSU*, etc.) – siehe Kapitel *3.9 Formen von Ressourcen*. Diese Ressourcen sollen direkt vom *Package-Launcher* angesprochen werden, da nur dadurch das Fehlerhandling und die Transaktionskonsistenz gewährleistet sind.

2. Verwalten Sie den Status der Transaktion über den Rückgabewert! Nur ein Rückgabewert von 0 oder 3010/1641, wenn nicht *CommitRebootExitCode* auf True steht, interpretiert der *Package-Launcher* als erfolgreichen Transaktionsabschluss. Prüfen Sie nach der Ausführung ihrer Installationsanweisungen im Script den Erfolg Ihrer Implementation und setzen Sie wenn nötig den Rückgabewert selber. (Bspl. fügen Sie bei einem Kommandozeilenbatch eventuell „Exit 0“ als letzte Anweisung hinzu, wenn die Installationsaufgaben erfolgreich durchgeführt wurden.)

   Gewährleistet ein von Ihnen in ein Kommandozeilenbatch integriertes *Legacy-Setups* selber, dass dieses bei Fehlern einen Rückgabewert <> 0 ausweist und bei einer erfolgreichen Installation 0 zurückgibt, können Sie auch auf die Implementation eines Rückgabewertes am Schluss des Scripts verzichten. Der Kommandozeilenprozessor übergibt in diesem Fall den Rückgabewert der letzten ausgeführten Exe-Datei an den *Package-Launcher* zurück.

3. Vielleicht macht es Sinn, das Vorhaben auf mehrere *Revisionen* zu splitten, um einzelne *Legacy-Setup*-Aufrufe isoliert zu installieren und eine genaue Kontrolle der Einzeltransaktionen zu erhalten. So ist es möglich, dass nach einem Fehler innerhalb der chronologischen Abfolge, bei einer späteren ReInstallation nur noch die fehlenden *Revisionen* nachgefahren werden.

4. **Wichtig:**

   Wird ein ***Post****Install_00x.cmd* verwendet und befinden sich *MSI/MSP/MSU*-Ressourcen im selben *Revisions*verzeichnis, dann fügen Sie die Scriptinhalte besser in ein ***Pre****Install_00x**+1**.cmd* in das Verzeichnis der nächsten *Revision*! Ausser, die Aufrufe in Ihrem *PostInstall_00x.cmd* können in keiner Konstellation zu einem Fehler führen.

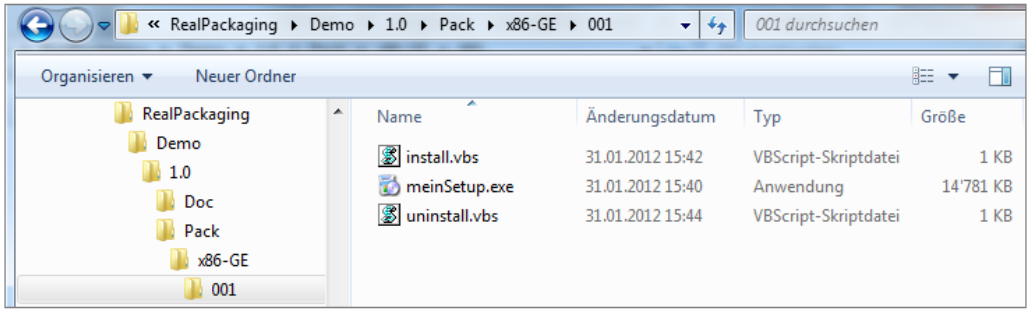
   Durch das Verschieben der Anweisungen in die nächste *Revision* wird sichergestellt, dass nach einem Abbruch infolge eines Fehlers bei Ausführung von *PostInstall_00x.cmd* und darauffolgender Reinstallation die *MSI/MSP/MSU*-Ressourcen nicht repariert werden (diese wären ja schon erfolgreich installiert). Zudem entfernt eine nach einem solchen Fehler initiierte Deinstallation alle *MSI/MSP/MSU*-Ressourcen (auch die der letzten, bzw. zweitletzten Revision)!

5. Testen Sie Ihre Scripte in jedem Kontext! INSTALL / **REPAIR** / REMOVE.

6. Beachten Sie den Ausführungskontext: Scripte (**ausser *ActiveSetup* und Powershell-Scripte!**) werden immer als 32 Bit Prozess ausgeführt. Die Variable *%PROGRAMFILES%* wird hierbei auf x64 Clients immer auf *C:\Program Files (x86)* stehen! Auch bei Subprozessen. Sollen Datenaufrufe in einer x64 Umgebung nach *C:\Program Files* zeigen, verwenden Sie in solchen Scripts die hartcodierte Variante, also nicht über die Environmentvariable, oder *%ProgramW6432%*.

   Möchten Sie die 64-Ausprägung von Exedateien verwenden, setzen Sie den alias *%WINDIR%\sysnative* ein!

7. Das Ausführungsverzeichnis wird vom *Package-Launcher* vor der Ausführung Ihres Scripts bei der Installation auf das *Revisions*verzeichnis gesetzt (ausser bei *ActiveSetup*).



8. Verwenden Sie *CopyLocal=True* in der INI-Datei des Softwarepakets, wenn im Rahmen einer **De**installation auf ein Script im Paket <> *PreInstall_00x.cmd*/*PostInstall_00x.cmd* zugegriffen werden soll oder wenn die **Inhalte** von *PreInstall_00x.cmd*/*PostInstall_00x.cmd* den Zugriff auf Paketressourcen bei der DeInstallation oder beim *REPAIR* benötigen! Ohne diesen Bezeichner kann das Paket bei Vorhandensein eines Deinstallationsscripts möglicherweise nicht deinstalliert werden, wenn die Deinstallation direkt über *LocalLauncher.EXE* gesteuert wird (Deinstallation über *SCCM* oder *Package-Upgrade*).

```
%PL%\LocalLauncher.EXE RealPackaging-Demo-1.0 /x
```

3.8.2 PreInstall_00x.cmd/.ps1 und PostInstall_00x.cmd/.ps1 (empfohlen)

Für einfachere Scriptimplemenationen ist der Einsatz von *PreInstall_00x.cmd* / *PostInstall_00x.cmd* und *ActiveSetup_00x.cmd* prädestiniert (.ps1 werden auch unterstützt). Der Bezeichner "00x" ist entsprechend der *Revision* anzupassen. Werden solche Dateien im *Revisions*verzeichnis vorgefunden, kopiert der *Package-Launcher* diese nach *%WINDIR%\Package-Launcher\Scripts\%PACKAGE%,* um auch im Rahmen einer DeInstallation Zugriff auf das DeInstallationsscript zu ermöglichen. In vielen Fällen ist bei so konzipierten Paketen ein *CopyLocal=True* nicht zwingend erforderlich, ausser die Script**inhalte** benötigen zum DeInstallationszeitpunkt die Verfügbarkeit der Ressourcen. Falls nötig, können diese alternativ auch in einem *PreInstall_00x.cmd* vorgängig nach %SCRIPTS% kopiert und bei der Deinstallation daraus angesprochen werden.

Werden Scripts über die Standardnamen *PreInstall_00x.cmd* / *PostInstall_00x.cmd* und *ActiveSetup_00x.cmd* implementiert, so werden allfällige Scriptimplementationen über die INI-Datei (*ExecuteFile, PreScript, PostScript, ActiveSetupScript_00x*) ausser Kraft gesetzt, bzw. damit ersetzt. Eine Kombination von beiden Verfahren ist innerhalb der selben *Revision* nicht möglich!

Beachten Sie, dass bei der Implementation in einer der Dateien *PreInstall_00x.cmd* / *PostInstall_00x.cmd*, der jeweilige Installationsstatus im Script abgefragt werden muss, damit bei der Installation und DeInstallation nicht die selben Scriptinhalte zur Ausführung gelangen. (Sie finden fertige Vorlagen für diese Scripte unter Work\00x):

Beispiel eines *PreInstall_**001**.cmd* oder *PostInstall_**001**.cmd*:

```
IF NOT    "%TASK%"=="REMOVE"  GOTO INSTALL_UPDATE_REPAIR
IF        "%TASK%"=="REMOVE"  GOTO REMOVE
```

Bei einer *Revision* 001 wird der *%TASK%* bei der Installation auf "*INSTALL*" oder "*REPAIR*" stehen - je nachdem, ob installiert wird oder ob es sich um eine Reparatur handelt.

In einer *Revision* >=001 wird die *%TASK%*-Variable, sofern die *Revision* noch nicht auf dem Client installiert ist, auf "*UPDATE*" stehen. Wird eine Reparatur vollzogen, ist auch eine *%TASK%*-Variable mit "*REPAIR*" zu erwarten:

Merke
Wird ein Scripts mit der Bezeichnung *PreInstall_00x.cmd* verwendet, ignoriert der *Package-Launcher* eine allfällige Implementation über *ExecuteFile* und *PreScript* in der INI-Datei. Das Gleiche gilt für den Bezeichner *PostScript* aus der INI-Datei: Dieser wird beim Einsatz einer Datei *PostInstall_001.cmd* ignoriert! Zudem ignoriert der *Package-Launcher* INI-Einträge über *ActiveSetupScript_00x*, wenn im Revisionsverzeichnis eine Datei *ActiveSetup_00x.cmd* vorliegt.

3.8.3 Pfadverweise

Der Startpfad wird vom *Package-Launcher* bei der Installation bei der Ausführung von *PreInstall_00x.cmd* / *PostInstall_00x.cmd* auf das *Revisions*verzeichnis des Installationspaket gesetzt (*CopyLocal=False*). Beispiel:

C:\Windows\ccmcache\4\x86-ML\002.

Wird *CopyLocal=True* verwendet, dann wird der Startpfad auf den entsprechenden Cache-Folder eingestellt:

Beispiel:

```
C:\Windows\Package-Launcher\Cache\TWI-FBMS-1.0\x86-ML\002.
```

Die Verwendung von **relativen** Pfaden in den Inhalten der Scripts funktioniert nur, wenn das Paket von lokaler Stelle (Cache-Folder bei CopyLocal=True, SCCM-Cache oder RPI-Cache), bzw. nicht von einem UNC-Share ausgeführt wird. Wenn das Paket mit relativen Pfaden hingegen per Doppelklick auf *SCCM-Launcher.vbs* direkt ab UNC-Share installiert wird und ein *CopyLocal=True* fehlt, funktionieren solche Aufrufe nicht (siehe https://support.microsoft.com/de-ch/kb/156276). Im Zweifelsfall verwenden Sie den Bezeichner *~%dp0* oder %SOURCE% und %CACHE%, anstatt relative Pfade:

```
"%SOURCE%\Source\Activate.EXE" "%SOURCE%\Source\License.lic"
```

3.8.4 ActiveSetup_00x.cmd (empfohlen)

Mit den Scripts *ActiveSetup_**INSTALL**_00x.cmd* (vereinfacht auch als *ActiveSetup_00x.cmd* integrierbar) *und ActiveSetup_**REMOVE**_00x.cmd* oder können sehr einfach Anforderungen zur Aktualisierung von Benutzerressourcen (HKCU-Regkeys, Dateien im Benutzerprofil) umgesetzt werden.

Werden Scriptteile in den besagten Dateien appliziert, so wertet der *Package-Launcher* **bei der Installation** Zeile um Zeile aus und löst allfällige Umgebungsvariablen auf. Dadurch ist gewährleistet, dass auch **bei der späteren Ausführung** des Scripts während der Anmeldephase über Mechanismen des *Active Setups*, die erweiterten Variablen wie %CACHE%, %PRODUCTCODE%, etc. indirekt Anwendung finden. Soll eine Variable – meist handelt es sich hierbei um Benutzervariablen wie %APPDATA% oder dergleichen – ausnahmsweise bei der Paketinstallation nicht aufgelöst werden, kann man die Variable mit einem doppelten Prozentzeichen maskieren:

Beispiel für ein *ActiveSetup_001.cmd*:

```
xcopy "%CACHE%\AppData" "%%APPDATA%%\" /s /y
msiexec /fup %PRODUCTCODE% /qb-!
```

Dieses Beispiel wird beim Kopieren durch den *Package-Launcher* nach *%WINDIR%\Package-Launcher\Scripts\%PACKAGE%\ActiveSetup_001.cmd* zu…

```
xcopy "C:\Windows\Package-Launcher\Cache\RealPackaging-Test-1\x86-ML\001\AppData" "%APPDATA%\" /s /y
msiexec /fup {100072AD-5B0F-4DAB-AF4F-A353C776234F} /qb-!
```

In der Registry wird das Script durch den *Package-Launcher* unter *HKLM\Software\Microsoft\Active Setup\Installed Components\%PACKAGE%\StubPath* folgendermassen hinterlegt:

```
"C:\Windows\Package-Launcher\Bin\ActiveSetup.vbs"
"C:\Windows\Package-Launcher\Scripts\RealPackaging-Test-1\ActiveSetup_001.CMD"
```

Merke
Während der späteren Ausführung des Scripts im Rahmen der Anmeldephase steht hier der Ausführungspfad auf *C:\Windows\System32*! Verwenden Sie daher absolute Pfade innerhalb des Scripts, bzw. mit Pfadvariablen initiierte Verweise (siehe Beispiel oben), ausser die angegebenen Befehlsdateien befinden sich in einem in der PATH-Variable, deklarierten Verzeichnis.

3.8.4.1 Wie geht der Package-Launcher mit diesen Dateien um

Während der Installation werden alle Scripts mit den folgenden Namen in das Verzeichnis *%SCRIPTS%\%PACKAGE%* kopiert:

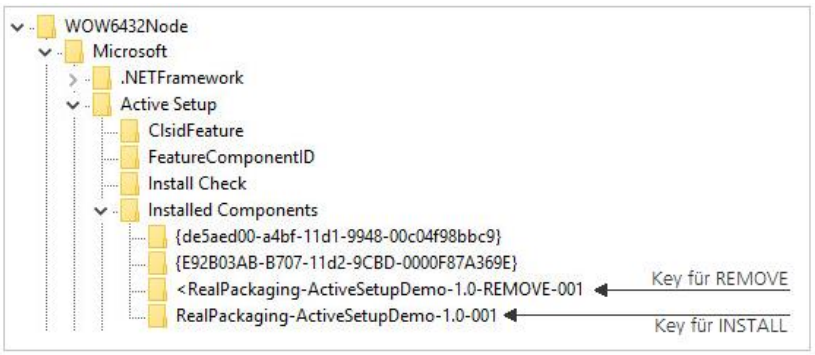
- ActiveSetup_INSTALL_00x.cmd
- ActiveSetup_REMOVE_00x.cmd
- ActiveSetup _00x.cmd

Zudem werden die Environmentvariablen in den Scripts aufgelöst und in den Zieldateien abgespeichert - mit Ausnahme von doppelt maskierten Variablen (%%, siehe letztes Kapitel).

Befinden sich Inhalte zu REG-Dateien, die direkt im *Revisions*verzeichnis abgelegt sind, kopiert der *Package-Launcher* diese in das *%SCRIPTS%*-Verzeichnis, damit solche auch aus dem *SCRIPTS*-Verzeichnis ansprechbar sind.

Die Datei *ActiveSetup_**REMOVE**_00x.cmd* und allfällig darin verwiesene REG-Dateien werden zudem im *%SCRIPTS%\%PACKAGE%*-Verzeichnis mit dem *ReadOnly* Dateiattribut versehen, um die Verfügbarkeit dieser Dateien nach einer Deinstallation des Paketes sicherzustellen. Damit also ein *Active Setup* auch nach der Deinstallation und nach dem damit verbundenen Löschen der Dateien (diese, die kein *ReadOnly*-Attribut aufweisen) aus *%SCRIPTS%\%PACKAGE%* funktioniert.

Ein *ActiveSetup_**REMOVE**_00x.cmd* wird in der Registry so registriert, dass dieses immer vor allen Installations-ActiveSetups ausgeführt wird. Ferner werden bei einer Installation die DeInstallations-Einträge aus einer früheren Installation entfernt und umgekehrt die Installations-Einträge bei der DeInstallation gelöscht.



Beispielinhalt für ein *ActiveSetup_**REMOVE**_00x.cmd*:

```
REG IMPORT "%SCRIPTS%\regkeys_REMOVE.reg"
```

3.8.4.2 Sofortige Ausführung von Active Setup

Standardmässig werden die Inhalte aus der Datei *ActiveSetup_00x.cmd* im Rahmen eines *Microsoft Active Setups* sofort für alle angemeldeten Benutzer ausgeführt (ausser *PushActiveSetup=False* wird über die Registry, Kommandozeile oder INI-Datei definiert), welche sich gerade am System angemeldet haben.

Sollte im Script ein *msiexec*-Aufruf enthalten sein, wird das *Active Setup* erst dann effektiv initiiert, wenn der Prozess *msiexec.exe* nicht mehr aktiv ist. Der Prozess *msiexec.exe* vernichtet sich selbst 5 Minuten nach einer Installation. Danach wird in einem Abstand von 15 Sekunden drei mal auf das Nichtvorhandensein von *msiexec* geprüft, bevor das *Microsoft Active Setup* zur Ausführung gelangt.

Durch diese Sicherheitsvorkehrung soll verhindert werden, dass eine über das *Active Setup* initiierte Ausführung von *msiexec* zu einem Zeitpunkt ausgelöst wird, wo *msiexec* bereits läuft. Wie wir wissen, kann dieser Prozess nicht gleichzeitig zweimal gestartet werden.

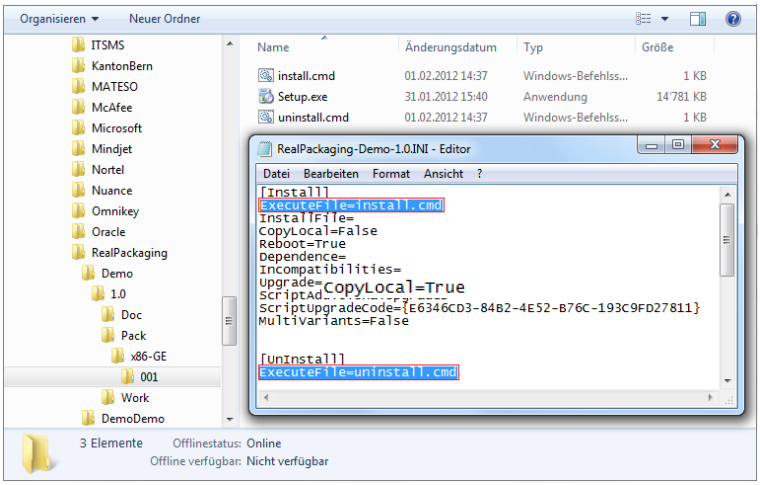
In einer Multibenutzerumgebung, wo mehrere Benutzer gleichzeitig angemeldet sind und zu diesem Zeitpunkt Software *per Package-Launcher* installiert wird, wird ein im Paket befindliches *Active Setup* mit *msiexec*-Inhalten nur in einer Benutzersitzung sofort ausgeführt. Die anderen Benutzersitzungen erhalten das *Active Setup* erst bei der Neuanmeldung.

3.8.5 Scripts über die INI-Datei (nicht empfohlen)

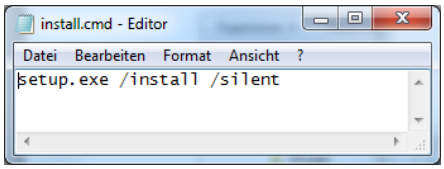
Über die INI-Datei ist es möglich, einzelne Kommandozeilen zu applizieren, deren Verweise entweder im *Revisions*verzeichnis abgelegt sind oder lokal verfügbar sind. Zudem sind mit dieser Variante **alle dem System bekannten Scriptarten einsetzbar**. Diese Scriptumsetzung gilt als Altenative und sollte nur verwendet werden, wenn das im Kapitel 3.8.2 und folgenden Kapiteln angegebene Vorgehen nicht zielführend ist.

3.8.5.1 Ein Beispiel

In der folgenden Abbildung sehen wir die Einträge aus der INI-Datei:



In der hier verwendeten Ausführung regelt das *Legacy-Setup* den Rückgabewert selbständig:



3.8.5.2 Scriptbezeichner ExecuteFile, ActiveSetupScript, PreScript & PostScript

Insgesamt stehen vier verschiedene Scriptbezeichner zur Verfügung, die in der *[Install]*- und *[UnInstall]*-Section verwendet werden können (nicht empfohlen).

1. ExecuteFile

   Soll ein lokales Programm oder ein einfaches Script ausgeführt werden, welches in einem *Revisions*verzeichnis abgelegt ist und welches ohne Rücksicht auf den Installationsablauf anderer Paketressourcen (bspl. MSI im selben *Revisions*verzeichnis), installiert werden kann, ist dies der richtige Bezeichner.

2. PreScript

   Müssen **zusammen mit nachgeschalteten Scripts** Scriptanweisungen **vor** der Installation von Installationselementen im selben *Revisions*verzeichnis durchgeführt werden, die durch den *Package-Launcher* direkt unterstützt werden (*MSI, MSP, MSU, APPV*), können diese mittels des Bezeichners *PreScript* zur Ausführung gelangen.

3. PostScript

   Müssen Scriptanweisungen **nach** der Installation von Installationselementen im selben *Revisions*verzeichnis durchgeführt werden, die durch den *Package-Launcher* direkt unterstützt werden (*MSI, MSP, MSU, APPV*), können diese mittels des Bezeichners *PostScript* zur Ausführung gelangen.

4. ActiveSetupScript

   Mittels dieses Bezeichners können Scripts, die im *Revisions*verzeichnis abgelegt sind, **im Kontext des Benutzers** ausgeführt werden. So werden Benutzerressourcen appliziert.

   Hierbei ist es bei Scriptverweisen, die auf ein Script im Paketverzeichnis verweisen (bspl. *ActiveSetupScript_001=%CACHE%\MyActiveSetupScript.cmd)* zwingend notwendig, dass auch *CopyLocal=True* in der INI-Datei, in der *Install*-Section gesetzt ist, damit das Script zum Ausführungszeitpunkt lokal verfügbar ist. Bei Verweisen auf **lokal** existierende Komponenten kann auf *CopyLocal=True* verzichtet werden (bspl. *ActiveSetupScript_001=REG.exe ADD HKCU\Software…).* Beachten Sie, dass Sie in das Script keine Anweisungen integrieren können, die erhöhte Rechte benötigen. So ist ein Schreiben der HKCU-Keys ohne Probleme möglich, nicht aber ein Schreiben der HKLM-Keys, sofern diese nicht per Security-Batch geöffnet wurden.

   Das mittels dieses Bezeichners angegebene Script wird bei der nächsten Benutzeranmeldung einmalig ausgeführt. Wird das Paket deinstalliert und neu installiert, "zieht" die *Active Setup* Implementation auch nach der zweiten Installation, da der *Package-Launcher* über Versionierung arbeitet. Bei der DeInstallation wird der *Active Setup* Key durch den *Package-*

Launcher automatisch bereinigt, so dass bei neuen Benutzern danach das Script nicht mehr ausgeführt wird.

Beispiele:

```
[Install]
ActiveSetupScript_001=regedit.exe /s "%CACHE%\HKCUEntries.reg"
ActiveSetupScript_002=xcopy.exe "%CACHE%\AppData" "%%APPDATA%%\" /y /s
ActiveSetupScript_003=msiexec /fup %PRODUCTCODE% /qb-!
ActiveSetupScript_004=%CACHE%\ActiveSetupScript_004.vbs
```

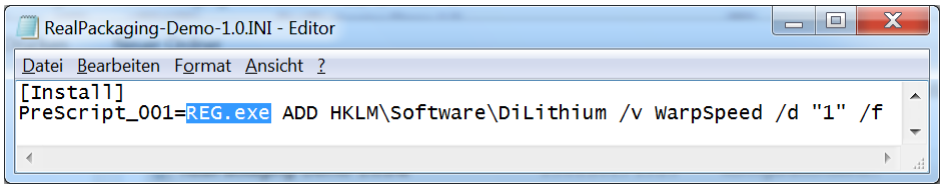
Man beachte auch die Maskierung mittels doppeltem Prozentzeichen in %%APPDATA%%. Diese bewirkt, dass zum Ausführungszeitpunkt %APPDATA% nicht aufgelöst wird und stattdessen als Variable "%APPDATA%\" in die Registry geschrieben wird. Diese Variable soll erst bei einer Benutzeranmeldung aufgelöst werden!

Alle Scriptbezeichner, ausser *ActiveSetupScript_00x* können mit oder ohne *Revisions*zusatz in der INI-Datei angegeben werden (beispielsweise *ExecuteFile=install.cmd* oder *ExecuteFile_**001**=install.cmd*). Es wird empfohlen, immer den *Revisions*zusatz zu verwenden, ausser bei der Planung des Paketes ist klar, dass alle künftigen *Revisionen* in ihrem *Revisions*verzeichnis immer das selbe Script mit dem selben Namen verwenden werden.

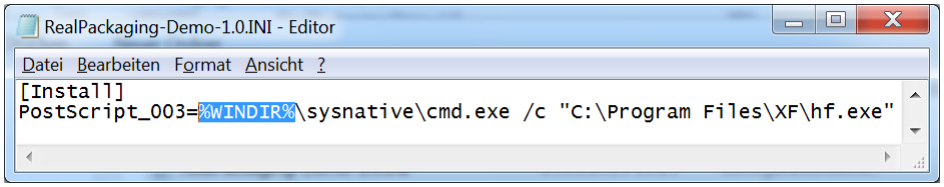
Achtung

Wird in einer Section eine *PreScript*-Zuweisung integriert, darf nicht gleichzeitig der Bezeichner *ExecuteFile* verwendet werden! In diesem Fall wären auch die für *ExecuteFile* vorgesehenen Anweisungen in das *Pre*- oder *PostScript*-Element aufzunehmen.

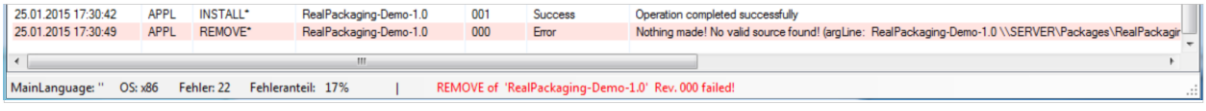
Alle Scriptbezeichner unterstützen auch die direkte Ausführung von Programmen, die lokal auf dem Computer abgespeichert sind.



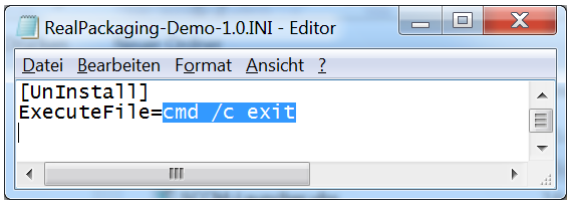
Zudem werden in der INI-Datei platzierte Umgebungsvariablen aufgelöst:



Wird in einer *Revision* nur ein Script bei der **Installation** angewendet, ohne dass sich weitere Ressourcen im Revisionsverzeichnis befinden, schlägt die **DeInstallation** fehl:



Diese Meldung kann umgangen werden, indem in der *UnInstall*-Section ebenfalls ein Script angegeben wird. Mit dem folgenden Befehl, der keinen funktionellen Charakter hat, lässt sich dies bewerkstelligen:



Merke
Wird für die **De**Installation ein im Paketrevisionsverzeichnis abgelegtes Script benötigt, muss im Paket in der **Install**-Section die Anweisung *CopyLocal=True* stehen, damit das Script bei der DeInstallation lokal zur Verfügung steht.

3.8.6 Zusätzliche Umgebungsvariablen während des Installationsprozesses

Während des Installationsprozess kann in der INI-Datei und in Scripts auf zusätzliche Umgebungsvariablen zugegriffen werden. Folgende Liste gibt Aufschluss über die vom Package-Launcher zusätzlich zur Verfügung gestellten Variablen:

Environmentvariable	Beschreibung
CACHE	Entspricht dem *Revisions*verzeichnis des lokal in *%WINDIR%\Package-Launcher\Cache* zwischengespeicherten Pakets (bspl. *C:\Windows\Package-Launcher\Cache\Acresso-ISSCRIPT-11.5\x86-EN\001*). Wird *CopyLocal=True* in der INI-Datei verwendet, ist dies das primäre Ausführungsverzeichnis der Installationsressourcen. Steht *CopyLocal* auf *False* ist der Bezeichner leer " ".
SOURCE	Entspricht dem *Revisions*- und Ausführungsverzeichnis. Wird für dieses Paket der Bezeichner *CopyLocal=True* eingesetzt, ist diese Variable der *CACHE*-Variable äquivalent. Wird hingegen *CopyLocal=False* eingesetzt, entspricht SOURCE **beim Installieren** mit *SCCM* dem *Revisions*verzeichnis des Pakets im lokalen *SCCM*-Cache, hingegen bei einer Installation ab Netzwerkshare dem *Revisions*- und Ausführungsverzeichnis des dort abgespeicherten Pakets. Bei einer **DeInstallation** mit *SCCM* und *CopyLocal=False* ist diese Variable leer " ".
PACKAGE	Diese Variable trägt den Namen des Pakets. Bspl. *RealPackaging-Demo-1.0* Bei einer Instanzvariante (siehe Kapitel *7.1.2 Instanzvarianten*) enthält die Variable den vollen Namen inkl. Variante Bspl. *RealPackaging-Demo-1.0-**ADMIN***
REVISION	Diese Variable entspricht der aktuellen, in diesem Moment zu installierenden *Revision*.
LOGFILE & LOGDIR	Diese Variable ist vorgesehen, um in einem Script eine Protokolldatei anzulegen. Der Wert entspricht der Datei… %WINDIR%\Logs**Install**\%PACKAGE%_%REVISION%_CUSTOM.LOG Bei einer DeInstallation trägt die Variable einen Wert mit Pfad im **UnInstall** Verzeichnis.
VARIANTSECTION	Diese Variable kann bei Variantenpaketen verwendet werden, um auf die eingelesene Sektion in der INI-Datei zuzugreifen. Entspricht beispielsweise der Section "99-USER" oder "99-ADMIN" aus der INI-Datei. Dadurch lassen sich per Script verschiedene Installationsausprägungen desselben Pakets realisieren.

VARIANT	Diese Variable enthält bei Variantenpaketen die verwendete Variante. Folgendes Beispiel installiert die Variante ADMIN:
	`SCCM-Launcher.vbs /i ADMIN`
	"%VARIANT%" wird in diesem Fall auf "ADMIN" stehen. So sind Scriptsniplets der folgenden Art möglich:
	`IF "%VARIANT%"=="ADMIN" GOTO ADMIN`

SCRIPTS	Diese Variable enthält den lokalen Pfad, wo der Package-Launcher die INI-Datei, sowie *PreInstall_00x.exe*, *PreInstall_00x.cmd*, *PostInstall_00x.exe* und *PostInstall_00x.cmd* ablegt.
DEPAGENCY	Enthält die lokal ermittelte *DepAgency*-Variable aus der Registry. Dies entspricht dem Inhalt *HKLM\Software\Real Packaging\Package-Launcher\DepAgency*
LANGUAGE	Enthält die Systemsprache (MainLanguage): "GE"|"FR"|"IT"|"EN"

PL_xxxx	Alle aus der INI in der *Variantsection* deklarierten Properties stehen über die Variablen PL_xxx für die Verwendung in Scripts zur Verfügung.
	Beispiel:
	`[99-STANDARD]`
	`SERIALNUMBER=1111-2222-3333-4444`
	ergibt: *PL_SERIALNUMBER=1111-2222-3333-4444*

TASK	Der Wert dieser Variable enthält "INSTALL", "UPDATE", "REPAIR" oder "REMOVE".
	Werden die Dateien *PreInstall_00x.cmd*, bzw. *PostInstall_00x.cmd* eingesetzt, so sind bedingte Anweisungen mit Berücksichtigung der TASK-Variablen unbedingt vorzusehen, da diese Scripts sowohl bei der Installation, als auch beim REMOVE ausgeführt werden:
	`IF "TASK%"=="INSTALL" GOTO INSTALL`
	`IF "TASK%"=="REPAIR" GOTO INSTALL`
	`IF "TASK%"=="REMOVE" GOTO REMOVE`

INI	Dateinamen und Pfad zu lokal gecachter INI-Datei.
PRODUCTCODE	Mittels der Variablen *PRODUCTCODE* kann bei einer MSI-Installation auf den *ProductCode* der MSI-Datei zurückgegriffen werden. Wird er in der INI-Datei zusammen mit dem Bezeichner *ActiveSetupScript_00x* verwendet, entspricht der Wert dem *ProductCode* der in diesem *Revisions*verzeichnis installierten MSI-Datei. (Siehe auch letztes Beispiel aus Kapitel *3.8.8 Beispiele*)
VIRTUALMACHINE	True|False: True, wenn Maschine virtuell ist.

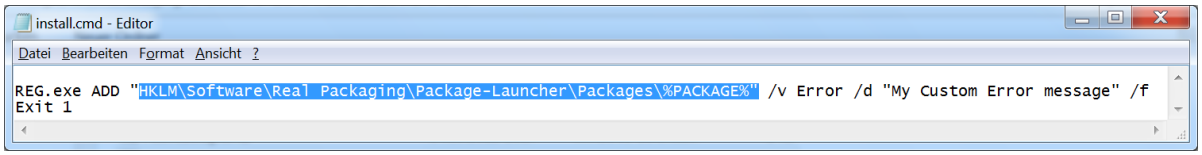
Achtung

Auf diese Variablen kann man auch in *Windows Installer Properties* über Variantensektionen der INI-Datei oder über die Bezeichner *MsiInstallProperties*, *MsiRepairProperties*, etc. (Bspl. *MsiInstallProperties=LOCALSOURCE=%CACHE%*) platziert in der INI-Datei, zugreifen. Daneben sind sie auch während des *MSI*-Installationsprozess als Umgebungsvariablen verfügbar.

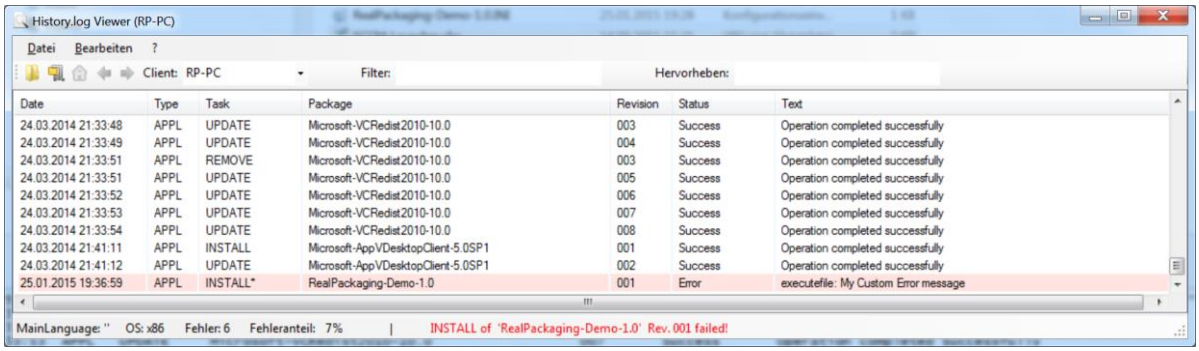
3.8.7 History.LOG kompatible Fehlermeldungen erstellen

Soll ein Script abgebrochen werden und möchte man eine spezifische Fehlermeldung in der Datei *History.LOG* erstellen, genügt es, im Script folgenden Registrykey zu schreiben:

```
HKLM\Software\Real Packaging\Package-Launcher\Packages\%PACKAGE%\Error
```



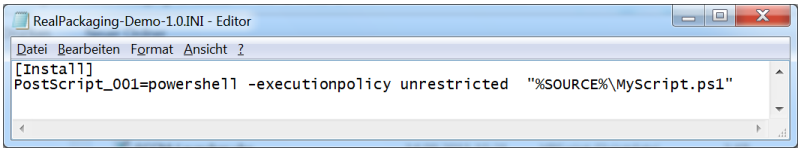
Wird das Script danach mit einem Exitcode beendet, zeigt der *Package-Launcher* dies in der Datei History.LOG an:



3.8.8 Beispiele

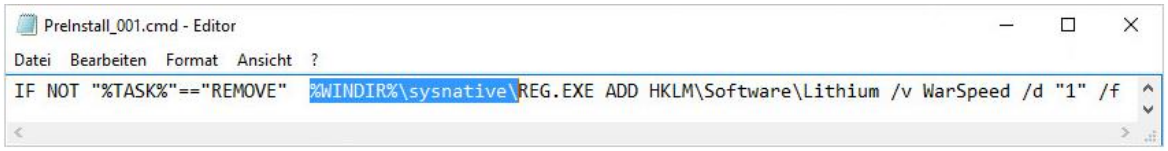
1. Nicht empfohlene Variante über INI: In diesem Beispiel wird vor der MSI-Installation ein cmd-Batch (install.cmd) ausgeführt. Bei der Deinstallation erledigt ein vb-Script (uninstall.vbs) abschliessende Bereinigungsaufgaben. Beide Scripts befinden sich im *Revisions*verzeichnis 001. Durch den Bezeichner *CopyLocal=True* wird sichergestellt, dass bei der Ausführung von *uninstall.vbs* die Source und das Script lokal vorliegend und zwischengespeichert sind.

2. Hier wird ein PowerShell-Script ausgeführt, welches im *Revisions*verzeichnis 001 abgespeichert ist (nicht empfohlene Variante über INI):

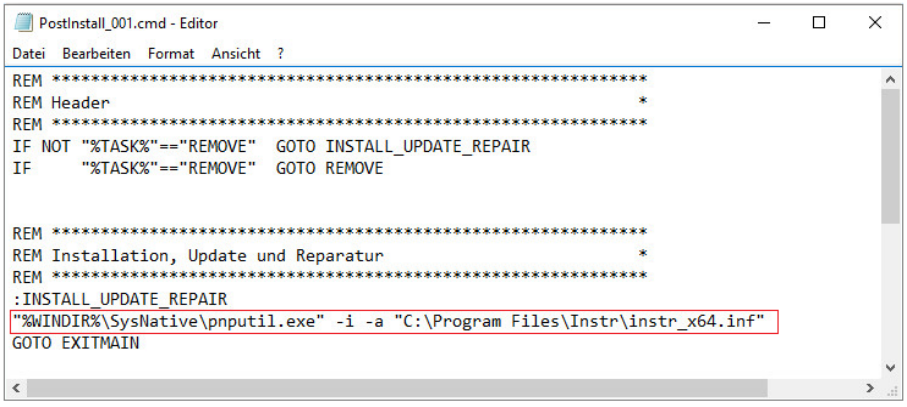


3. Erstellung eines Registrykeys vor Ausführung der MSI-Installation in *Revision* 002 durch Ausführung einer lokalen exe-Datei (REG.exe) über ein abgelegtes *PreInstall_001.cmd*: Achtung: das Script registriert so für x64 Clients in HKLM\Software**Wow6432Node**\Lithium.

4. Das selbe Script registriert hier auf x64 Clients in HKLM\Software\Lithium

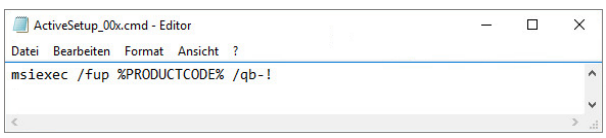


5. Ausführung einer für 64 Bit vorgesehenen Systemdatei unter x64. Durch Angabe des Alias "sysnative" kann auf x64 Clients aus 32 BIT Prozessen auf das Verzeichnis *C:\Windows\System32* (anstatt auf *C:\Windows\SysWOW64*) zugegriffen werden. Zwingend ist auch die Angabe des Pfades auf *C:\Program Files*, anstatt über die Variable *%PROGRAMFILES%*, wenn die Datei unter x64 in *C:\Program Files* liegt. Denn Environmentvariablen werden vom Parent-Prozess geerbt, welcher als 32 BIT ausgeführt wird.



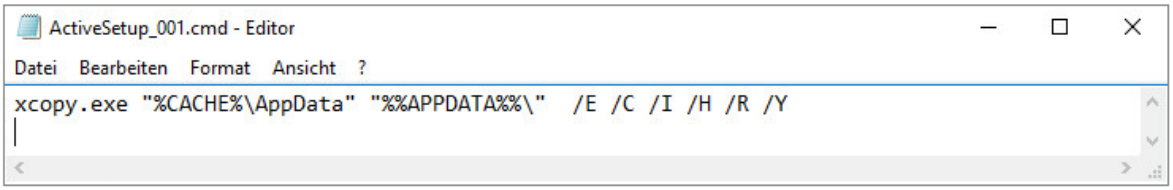
6. Hier wird das lokal abgespeicherte *msiexec.exe* für die Ausführung im Kontext des Benutzers bei der nächsten Anmeldung am Computer integriert. Für einige Fälle muss *CopyLocal* auf *True* stehen, wenn *Windows Installer* Reparaturen, die Verfügbarkeit von lokalen Ressourcen nötig machen!!

   Die Variable *%PRODUCTCODE%* weist den *Package-Launcher* an, den effektiven *ProductCode* aus der MSI-Datei dieser Revision in der Datei abzuspeichern.

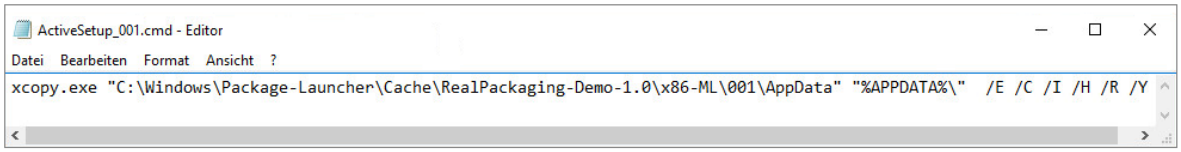


7. Sollen in einem *ActiveSetup_00x.cmd* Environmentvariablen für den geplanten Ausführungszeitpunkt verwendet werden, so sind diese mit doppeltem %% zu maskieren. Dadurch wird sichergestellt, dass der *Package-Launcher* diese Variable bei der Paketausführung und lokalen Speicherung der Datei nicht übersetzt.

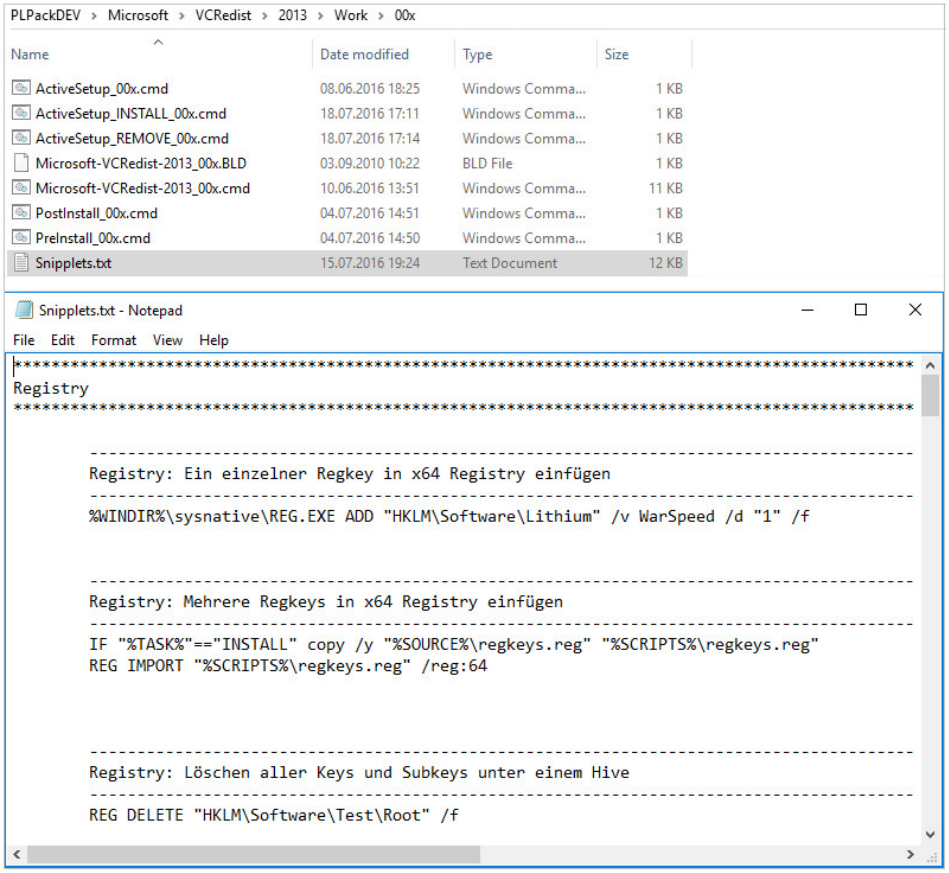
   Die Datei mit folgende Anweisung…



   …speichert der *Package-Launcher* im SCRIPTS-Verzeichnis in der folgenden Form…



8. Im Verzeichnis *\Work\00x* befinden sich in der Datei *Snipplets.TXT* Vorlagen für wiederkehrende Aufgaben. Diese lassen sich per copy/paste in die jeweiligen Scripte einbauen.



3.9 Formen der Ressourcen

Im *Revisions*verzeichnis können durch den Softwarepaketierer verschiedene Setup-Arten zur direkten Installation abgelegt werden. Direkt werden durch den *Package-Launcher* folgende Typen unterstützt:

MSI	*Windows Installer* Setup. Dies entspricht dem Standardtyp.
MST	*Windows Installer Transformation*. Diese kann nur zusammen mit einer *MSI*-Datei eingesetzt werden. Siehe Kapitel *3.7.1 MSI und MST Dateien*
PostInstall	Befindet sich eine Datei mit der Bezeichnung *PostInstall_00x.**EXE*** im *Revisions*-verzeichnis wird diese im Rahmen einer *MSI*-Installation als *Custom Action* ausgeführt. Das Script *AddProperties.vbs* integriert diese in die *MSI*-Datei. Die Ausführung erfolgt im Systemkontext mit erhöhten Privilegien nach der Installation aller Ressourcen durch die *MSI*-Datei.
MSP	*MSP*-Dateien sind *Windows Installer* Patches. Diese können einzeln oder in Kombination mit einer *MSI*-Datei im selben *Revisions*verzeichnis verwendet werden. Der *Package-Launcher* überprüft das komplette *Revisions*verzeichnis nach dem Vorkommen aller Patches. Sind mehrere Patchdateien im selben *Revisions*verzeichnis vorhanden, werden alle Patches (gleichzeitig) dem Installationsprozess übergeben. Achtung: Dieses Verfahren ist erst ab Patches der *Revision* 3.x möglich! (siehe Kapitel *3.11 Anwendung von Patches und Transformationen*). Wird im selben Verzeichnis sowohl eine *MSI*-Datei, als auch eine oder mehrere *MSP*-Dateien vorgefunden, so wird zuerst die *MSI*-Datei installiert und erst danach kommt der/die Patches zur Anwendung. Kommt eine neue Software zum Einsatz, die noch nicht verteilt ist, so wird das Erstellen eines *In-Place-Updates* als Vorbereitung empfohlen. D.h., dass in diesen Fällen eine *MSP*-Datei gleich auf die *MSI*-Datei angewendet, bzw. die *MSI*-Datei mit der/den *MSP*-Datei/en gepatcht wird. Ist die Software bereits verteilt, darf dieses Verfahren in der Vorbereitung nicht angewendet werden, hier ist die isolierte Patchdatei in einem neuen *Revisions*verzeichnis abzulegen!
PreInstall	Befindet sich eine Datei mit der Bezeichnung *PreInstall_00x.**EXE*** im *Revisions*-verzeichnis, wird diese vor allen allfälligen *MSI* oder *MSP*-Installationen ausgeführt. Befindet sie sich alleine im Verzeichnis, so werden nur die Programmanweisungen, die dort aufgeführt sind, ausgeführt. *PreInstall*-Dateien eignen sich zudem für den Aufruf von *Legacy-Setups*.
Scripts	Befindet sich ein Script im *Revisions*verzeichnis, welches über die *INI*-Datei des Softwarepakets angegeben (*ExecuteFile=<Script.ext>*) wurde, oder die zulässige Namensgebung *PreInstall_00x.**cmd***/*PostInstall_00x.**cmd*** verwendet, wird dieses vor allen allfälligen *MSI* oder *MSP*-Installationen und auch vor einem allfälligen *PreInstall_00x.**exe*** ausgeführt. Befindet es sich alleine im Verzeichnis, so werden nur die Programmanweisungen, die dort aufgeführt sind, ausgeführt. Script-Dateien eignen sich für den Aufruf von *Legacy-Setup*s Vorbereitungsaktionen oder Nachbearbeitungsautomatismen.

MSU

Hotfixes in Form von *.MSU*-Dateien können einfach in das Revisionsverzeichnis abgelegt werden, sofern der INI-Bezeichner *EnableMSU* gesetzt wurde (in der Regel durch *CreatePackage* standardmässig implementiert).

Beachten Sie, dass für MSU-Pakete, die später wieder deinstalliert werden sollen, der Bezeichner *CopyLocal=True* erforderlich ist!

In Einzelfällen lassen sich MSU-Dateien mit den Standardmechanismen nicht deinstallieren. Für diese Fälle markieren Sie in der INI-Datei in der *UnInstall*-Section den Bezeichner *EnableMSU* mit *False*…

```
[UnInstall]
EnableMSU=False
ExecuteFile=remove.cmd
```

…um im Anschluss mit einem eigenen Script die Deinstallation vorzunehmen. Im remove-Script (hier im Beispiel *remove.cmd*) kann dann ein Verweis in der folgenden Art durchgeführt werden:

```
wusa /uninstall /kb:2819745 /quiet /norestart
```

Klappt der vorherige Aufruf unter x64 nicht, so bietet sich an, im x64er *remove.cmd* stattdessen folgenden Aufruf zu platzieren:

```
%windir%\sysnative\wusa.exe /uninstall /kb:2819745 /quiet /norestart
```

App-V

Es werden sowohl App-V 4.6, als auch App-V 5 Ressourcen unterstützt . Diese können nativ in das Revisionsverzeichnis 001 kopiert werden. Bei der Integration von App-V 5 Paketen gibt es zwei Varianten: Die *App-V Full Integration,* die ein Streaming unterstützt oder das *standalone model*, wo Pakete wie reguläre Pakete behandelt werden. Im letzteren Modell kann das Paket mit den selben Tools verwaltet werden (*RPI, History.LOG Viewer*), wie gewöhnliche Pakete.

Achtung

MSI oder Legacy?

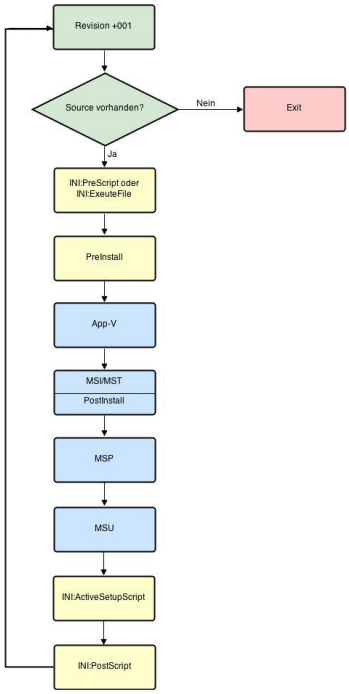
Das primäre Setup-Format sollte aus Gründen…

1. der Transparenz,
2. der Erweiterbarkeit,
3. den Möglichkeiten zur Anpassung und zur Direktmanipulation durch den Paketierer,
4. der Risikominimierung betreffend Erscheinen eines unerwarteten Dialogs im silent-Modus,
5. der *Rollback* und Reparaturfähigkeit,
6. der Verwaltung durch einen zentralen Dienst, etc.

immer das *MSI*-Format sein. Ein Ausführen eines *Legacy-Setups* wird nur dort empfohlen, wo es sich beim Setup um eine hardwarenahe Treiberanbindung oder ähnliches handelt, die sich nicht oder nur sehr schwierig mittels *Snapshottechniken* aufzeichnen und in eine *MSI*-Datei überführen lässt oder wo der Aufwand zur nativen Paketerstellung oder zur Paketerstellung mittels *Snapshottechniken* unverhältnismässig gross ist!

3.9.1 Vereinfachtes Ablaufschema

Gilt für die Installation, sowie für die DeInstallation



3.10 MSI-Spezialfälle – minimale Aktualisierungen

Windows Installer kennt drei mögliche Aktualisierungsformen. Dies sind...

- Small Update (für Aktualisierungen geringen Ausmasses)
- Minor Upgrade (für grössere Aktualisierungen, die auch eine Erhöhung der *Minorversion* rechtfertigen)
- Major Upgrade (für komplexe Aktualisierungen)

Alle diese Aktualisierungsformen können mit dem Servicemodell des *Package-Launchers* verwendet werden. *MSI*-Basisinstallationen, welche ein neues Produkt darstellen (RTM), werden immer in Form eines *Major Upgrades* (bzw. *Package-Upgrades*) realisiert. Dieser enthält einen neuen *ProductCode* und deinstalliert (wenn nicht das Verfahren des *Package-Upgrades* gewählt ist) im Umfeld des Unternehmens allfällig vorliegende Vorversionen samt aller *Revisionen*.

Minimale Aktualisierungen können in einer *Revision* angewendet werden. Unter diese Kategorie fallen *Small Updates* und *Minor Upgrades*. *Minimale Aktualisierungen* werden durch *Windows Installer* immer in Form einer Reinstallation und gleichzeitiger Anwendung des *Recache-Flags* angewendet. Die lokal vorliegende gecachte Version der Basisinstallation wird hierbei durch die neue Version ersetzt. Die entsprechende vorausgehende *Revision* kann daher im Installationsablauf bei einer Deinstallation nicht mehr wirklich vom System entfernt werden, da dies bereits mit dem Entfernen des Updatepaketes geschehen ist. Der *Package-Launcher* erkennt diese Situation automatisch und reagiert nach aussen transparent. D.h., dass wir also auch bei der

Anwendung von *minimalen Aktualisierungen* entsprechende korrekte Einträge im *History.LOG* finden und auch das *SCCM*-Softwareinventar korrekt ausgelöst wird. Sogar Reparaturen (virtuell) der durch den Update ersetzten *Revisionen* lassen sich ausführen, ohne Nebeneffekte auszulösen.

Bei der Anwendung des Updatepakets wird automatisch die *Windows Installer Property MSIENFORCEUPGRADECOMPONENTRULES* auf 1 gesetzt, wodurch nur regelkonforme Aktualisierungen ermöglicht werden. Das Aktualisierungspaket muss daher den Richtlinien über die Erstellung von minimalen Aktualisierungen entsprechen. Werden die Richtlinien vom Aktualisierungspaket nicht eingehalten, erscheinen Fehlermeldungen im *History.LOG* wie diese:

```
msiexec: Upgrade of feature MainFeature has a missing component.
```

```
msiexec: New upgrade feature MainFeature must be a leaf feature.
```

Die Anwendung der Property *MSIENFORCEUPGRADECOMPONENTRULES* kann man über die *MsiInstallProperties* übersteuern. Dies wird aber nicht empfohlen! Die Erkennung, ob in einer *Revision* eine *Minimale Aktualisierung* vorliegt oder nicht, erledigt der *Package-Launcher* automatisch durch einen Vergleich verschiedener Properties (*ProductCode, PackageCode, PLRevision, PLPackage*) zwischen einem allfällig vorliegend installierten Produkt und einem durch die *Revision* anzuwendenden Paket. Sollte erkannt werden, dass ein allfällig vorliegend installiertes Produkt, den gleichen *ProductCode* registriert hat, wie das anzuwendende Paket, ist aber der *PackageCode* gleich oder die Property *PLPackage* verschieden oder die *PLRevision* gleich, so wird dieser Zustand hingegen so interpretiert, dass auf diesem Gerät die „gleiche" Software durch anderweitige Mechanismen bereits installiert wurde (bspl. Handinstallation der Originalsoftware) und die Transaktion wird mit einer entsprechenden Fehlermeldung im *History.LOG* quittiert:

```
This application is already installed manually or by other processes…
```

Achtung: bei minimalen Aktualisierungen kann keine neue Transformation verwendet werden! Es wird bei der Anwendung auf allfällig recachte MSTs der Basisinstallation zurückgegriffen, bzw. wenn bei der Installation einer Initialversion keine Transformation verwendet wurde, dann wird auch bei der Anwendung des *Small Updates* oder *Minor Upgrades* keine Transformation verwendet. Sollten zusätzliche Änderungen am Design der *MSI* innerhalb derselben *MSI*-Datei erfolgen, so könnte dies nur durch nur eine Anpassung des *Update-MSIs* unter Einhaltung der Richtlinien über die Erstellung von minimalen Aktualisierungen erfolgen! Minimale Aktualisierungen bedeuten für den Paketierer lediglich, dass das Updatepaket innerhalb einer neuen *Revision* zu implementieren ist. Es müssen keine *Properties* definiert oder spezielle Vorkehrungen vorgenommen werden!

3.11 Anwendung von Patches und Transformationen

Innerhalb der *Revision* werden alle vorgefundenen Patches Transformationen (bei Transformationen sind die Ausnahmen unter Kapitel *3.7.1 MSI und MST-Dateien* beschrieben) in einer Transaktion angewendet. Die Zusammensetzung des/der Namen spielt für den *Package-Launcher* an sich keine Rolle. Dieser verwendet alle im Paketverzeichnis vorgefundenen Transformationen und Patches.

Transformationen werden nach dem Sicherheitsmodell *Secure-At-Source* angewendet. Dadurch ist gewährleistet, dass Transformationen die im lokalen Zwischenspeicher nicht zur Verfügung stehen, ausschliesslich aus dem Stammverzeichnis des *Windows Installer* Pakets (entspricht

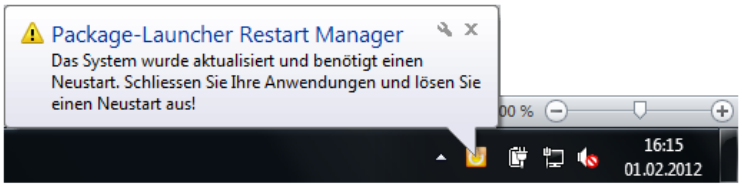
unserem *Revisions*verzeichnis) verwendet werden. Diese Absicherung dient gegen unautorisierten Gebrauch und gegen Missbrauch von Transformationsdateien. Die Verwendung mehrerer Patches innerhalb der selben *Revision* kann mit gutem Gewissen nur bei Patches der Version 3.x angewendet werden, da hier Implementationen durch den *Windows Installer* vorliegen, welche es dem Patchautor mit einfachen Bordmitteln ermöglichen, nur eine Zielversion bei Vorliegen von mehreren Service Packs (*Minor Upgrade Patch*) anzugeben. Patchdateien der Version 3.x müssen eine Datenbanktabelle mit dem Namen *MsiPatchSequence* verwenden, wodurch der Patch als Patch der Version 3.x identifiziert werden kann. Der Paketierer kann dies durch Öffnen des Patches mit File/Open der *MSP*-Datei in *Orca* erkennen. Bei der Anwendung mehrerer Patches der Version 2.0 muss gewährleistet sein, dass es sich um einen *Multi-Target Patch* handelt. Wird keine gültige Zielversion erkannt, wird die Transaktion im *History.log* mit der Meldung…

msiexec: The installer cannot install the upgrade patch because the program being upgraded may be missing or the upgrade patch updates a different version of the program. Verify that the program to be upgraded exists on your computer and that you have the correct upgrade patch

…quittiert (*ERROR_PATCH_TARGET_NOT_FOUND*).

3.12 Umgang mit Computerneustarts

Der *Package-Launcher* verwaltet Neustarts in der Regel selbständig, ohne dass sich der Paketierer darum kümmern muss. Wird durch ein Paket eine Neustartanforderung per Exitcode 3010 ausgelöst, erscheint nach der Installation der *Package-Launcher Restart Manager,* der den Benutzer in regelmässigen Abständen anweist, einen Neustart durchzuführen.



Der Prozess der Paketinstallation gibt in diesem Szenario immer den Rückgabewert 0 an SCCM zurück. Neben dieser Selbsterkennung kann mit dem INI-Eintrag *Reboot=True* der *Package-Launcher* angewiesen werden, den *Restart Manager* unabhängig des Rückgabewertes aus einer enthaltenen *Revision*, mit einer Neustartaufforderung zu starten.

Sind **in Ausnahmefällen** für die Verteilung mit SCCM **zwingende** Neustarts **ohne Interaktion** mit dem Benutzer gefordert, ist nach den Ausführungen der folgenden Kapitel vorzugehen:

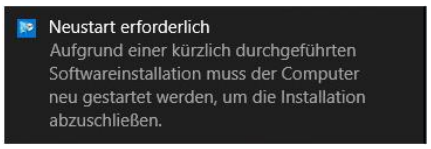
3.12.1 CommitRebootExitCode

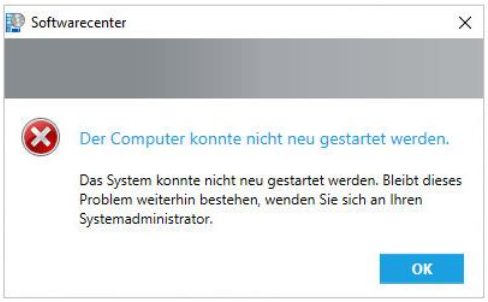
Neustart**anforderungen** lassen sich mit dem Bezeichner *CommitRebootExitCode=True* in der INI-Datei und einem Exitcode 3010 oder 1641 innerhalb des Softwarepakets umsetzen. Der Neustart wird dann von *SCCM* oder vom *Package-Launcher App-Installer* durchgeführt.

SCCM kennt zwei Arten von Neustarts aufgrund Rückgabewerten: Den sogenannten *Soft Reboot (3010)* und den *Hard Reboot (1641)*. Per Definition ist der Unterschied der, dass bei *Hard Reboots (1641)* keine weitere Installation mehr vollzogen wird, bis der Neustart ausgeführt wurde. Bei *Soft Reboots (3010)* wird der Neustart (in vielen Fällen) erst nach der Ausführung aller anstehenden Deployments gefordert.

3.12.1.1 Neustartanforderung innerhalb und zwischen Revisionen

Soll während der Installation zwischen zwei *Paketrevisionen* ein Neustart ausgeführt werden, so ist der Bezeichner *CommitRebootExitCode=True* in der *Install*-Section der INI-Datei zu applizieren. Zusätzlich muss innerhalb einer *Revision* dann der Exitcode 3010 oder besser **1641** initiiert werden (Script/MSI), um den Neustart effektiv auszulösen. Sowohl unversionierte, als auch versionierte *Applications* können nach diesem Verfahren integriert und zugewiesen werden. **Es ist wichtig, dass auf die Neustart auslösende *Revision* immer eine folgende *Revision* (00x + 1) folgt, welche im Fall eines Erfolgs 0 zurückgibt**, sonst kann es vorkommen, dass folgende Neustartaufforderung von SCCM (Abbildung links) auch nach mehrmaligem Neustart nicht verschwindet und bei manueller Auslösung über den *Software Center* in einer Fehlermeldung resultiert (Abbildung rechts).





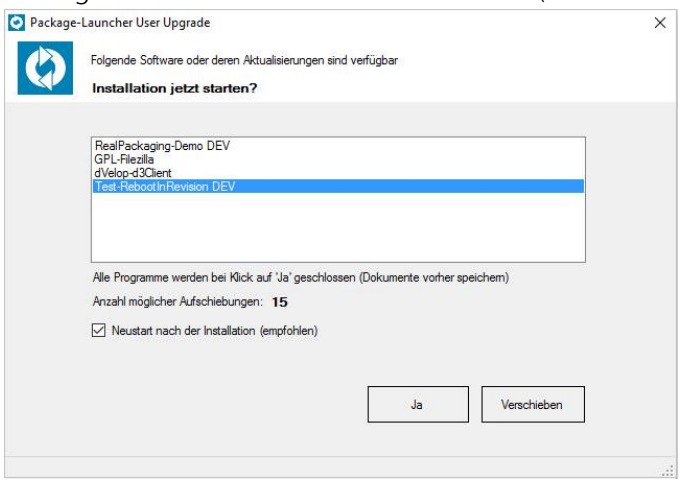
Machen Sie sich Gedanken darüber, ob Sie für Neustarts innerhalb *Revisionen* einen sofortigen Neustart nach Auslösung des Neustarts in der *Revision* wünschen (Rückgabewert 1641) oder ob der Neustart nach Beendigung aller Deploymenttransaktionen (Rückgabewert 3010) genügt. Im Zweifelsfall entscheiden Sie sich hier für die **erste Variante (Rückgabewert 1641)**, stellen aber sicher, dass eine *Revision* mit Rückgabewert 0 dieser *Revision* folgt.

Beispiel mit dem Package-Launcher App-Installer und Rückgabewert 3010

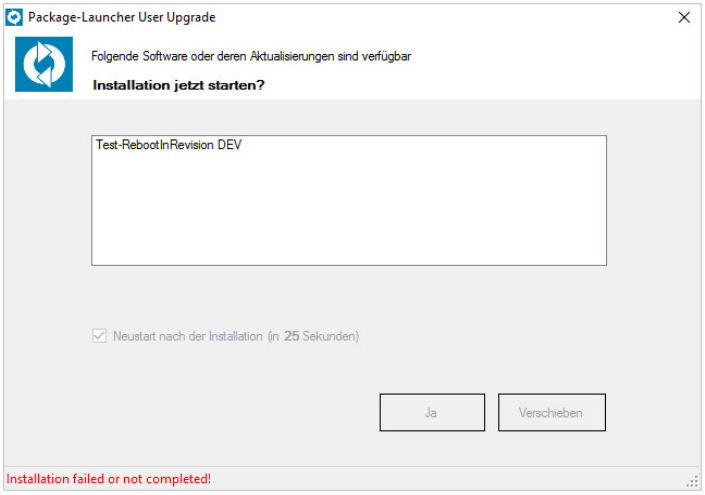
Hier wird das Paket *Test-RebootInRevision-1* mit zwei *Revisionen* installiert. Die erste beendet mit 3010, die zweite mit 0 (*PreInstall*-Script mit 'exit 3010' bzw. 'exit 0')

Im Deploymentablauf mit dem *Package-Launcher App-Installer* zeigt sich folgendes Verhalten:

1. Anzeige vor dem Auslösen der Installation (mehrere Deployments anstehend):



2. Alle Deployments wurden installiert. Die Installation der *Revision* 001 von *Test-RebootInRevision* ist erfolgreich. Ein Neustart wird automatisch durchgeführt.



Folgende Anzeige würde sich zu diesem Zeitpunkt im History.LOG zeigen:

22.12.2016 17:52:01	APPL	INSTALL	RealPackaging-Demo-1.0	001	Success	Operation completed successfully
22.12.2016 17:52:02	APPL	UPDATE	RealPackaging-Demo-1.0	002	Success	Operation completed successfully
22.12.2016 17:52:31	APPL	INSTALL	GPL-Filezilla-3.2	001	Success	Operation completed successfully
22.12.2016 17:52:33	APPL	INSTALL	Test-RebootInRevision-1	001	Success	Operation completed successfully (reboot commited, installation not completed)
22.12.2016 17:52:47	APPL	INSTALL	dVelop-d3Client-7.1	001	Success	Operation completed successfully

3. Nach dem automatisch durch den *Package-Launcher App-Installer* durchgeführten Neustart installiert SCCM nach wenigen Minuten die folgenden *Revisionen* (hier im Beispiel Rev. 002) **selbständig**, ohne dass mittels dem *Package-Launcher App-Installer* die Transaktion nochmals ausgelöst werden muss (Zeile mit Zeitstempel 17:56)!

Anzeige im History.LOG. Rote Linie markiert den automatischen Neustart:

22.12.2016 17:52:33	APPL	INSTALL	Test-RebootInRevision-1	001	Success	Operation completed successfully (reboot commite...
22.12.2016 17:52:47	APPL	INSTALL	dVelop-d3Client-7.1	001	Success	Operation completed successfully
22.12.2016 17:56:40	APPL	UPDATE*	Test-RebootInRevision-1	002	Success	Operation completed successfully

Achtung

Gleichzeitig zugewiesene andere Deployments (im obigen Beispiel *GPL-Filezilla*, *dVelop-d3Client*, *RealPackaging-Demo*) werden bei Rückgabewert 3010 in der gleichen Transaktion installiert. Damit eine oder mehrere *Revisionen* nach einem Neustart installiert werden und damit keine Neustartschleife entsteht, muss gewährleistet sein, dass die Neustartanforderung nicht in der letzten *Revision* eines Pakets erfolgt! Allenfalls hilft das Anfügen einer zusätzlichen Dummy*revision* mit Rückgabewert 0.

Beispiel für eine korrekte Vorgehensweise in initialen Paketen (RTM):

Revision:	001	002	003	004
Exitcode:	0	3010	0	0

Ein Rückgabewert von 3010 unterliegt weiter folgender Einschränkung, dass dieser nur in **initialen Paketen** (RTM) angewendet werden kann! In *Revisionen*, die **erst nach der Erstüberführung** appliziert werden (Updates), funktioniert der Mechanismus nicht mehr, da jede weitere einzelne *Revision* in *Updates* nach der Erstüberführung durch eine eigenständige Application verkörpert wird. Diese wird nicht vor Beendigung abgebrochen und deren Installation wird in einer anderen *Revisions*application (00x + 1) fortgesetzt – auch wenn 3010 als Prozessrückgabewert einer *Kindrevision* resultiert. Alle angetriggerten *Applications* und *Deployments* werden – wie wir wissen - mit dem *Soft Reboot* (3010) grundsätzlich installiert. Wird in einem nach der Erstüberführung vorgesehenen Update ein zwingender Neustart benötigt, ist ein Rückgabewert 1641 *(Hard Reboot)* mit einer darauf folgenden *Revision* mit Rückgabewert 0 erforderlich.

Beispiel einer zielführenden Umsetzung nach einer Erstüberführung:

Revision:	001	002	003	004	005	006
Exitcode:	0	0	0	0	1641	0
Überführung:	RTM Erstüberführung				Update	
Application:	RTM				005	006

Das Verhalten ist nach der SCCM-Integration mit SCCM gründlich zu prüfen!

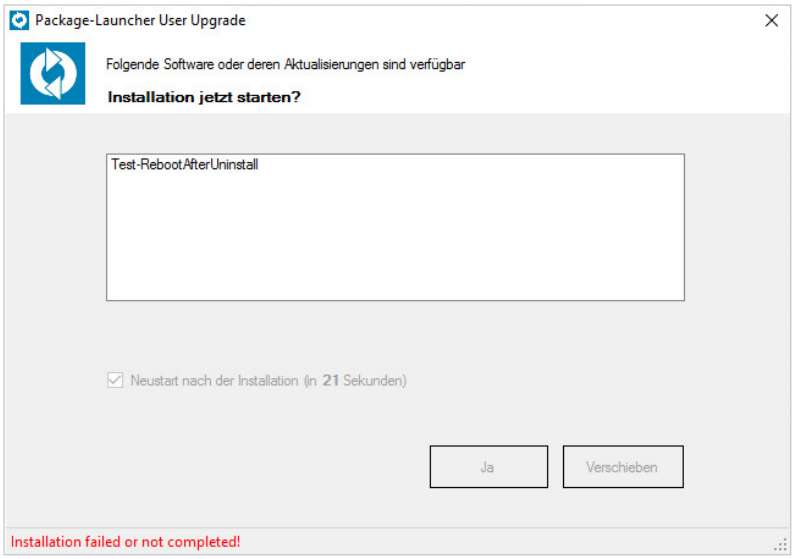
3.12.1.2 Neustartanforderung nach REMOVE

Oft ist es im Rahmen eines *Upgrades* nötig, dass nach der Deinstallation einer Vorversion ein Neustart ausgelöst werden soll, bevor eine neue Version installiert wird. Dieses Begehren ist nach dem selben Prinzip umzusetzen:

1. Es ist sicherzustellen, dass in der INI-Datei des alten Pakets der Bezeichner *CommitRebootExitCode=True* in der ***UnInstall***-Section deklariert oder dass der Bezeichner im neuen Paket in der ***Install***-Section implementiert ist (einfachere Variante).
2. Es ist sicherzustellen, dass irgendeine *Revision* beim *REMOVE* des alten Pakets mit Rückgabewert 3010 beendet. Allenfalls hilft das dortige Anfügen einer zusätzlichen Dummy*revision* mit Rückgabewert 3010. (Die im obigen gelben Absatz ausgeführten Beschränkungen betr. Nachfolge*revision* mit Rückgabewert 0 gelten hier nicht)
3. Das neue Paket verteilen. Nach dem vollständigen REMOVE des alten Pakets wird die Installation vom *Package-Launcher* mit 3010 abgebrochen. Nach einem Neustart wird die Installation automatisch (nach ein paar Minuten) durch SCCM fortgesetzt.

Beispiel nach REMOVE mit dem Package-Launcher App-Installer

Nach der Installation des neuen Pakets bleibt der Dialog für 10 Sekunden stehen, bevor automatisch ein Neustart ausgeführt wird.



Anzeige im History.LOG

Die rote Linie markiert den automatischen Neustart. Nach dem Neustart wurde um 10:21 das neue Paket automatisch durch SCCM installiert:

23.12.2016 10:03:02	APPL	INSTALL	Test-RebootAfterUninstall-1.0	001	Success	Operation completed successfully
23.12.2016 10:07:53	APPL	UPDATE	Test-RebootAfterUninstall-1.0	002	Success	Operation completed successfully
23.12.2016 10:17:49	APPL	REMOVE	Test-RebootAfterUninstall-1.0	002	Success	Operation completed successfully (reboot required)
23.12.2016 10:17:49	APPL	REMOVE	Test-RebootAfterUninstall-1.0	001	Success	Operation completed successfully (reboot commited)
23.12.2016 10:21:23	APPL	INSTALL*	Test-RebootAfterUninstall-2.0	001	Success	Operation completed successfully

Achtung

Beim *REMOVE* wird nie zwischen einzelnen *Revisionen* ein Neustart ausgelöst. Gibt irgend eine *Revision* beim *REMOVE* einen Rückgabewert 3010 zurück, wird zuerst der *REMOVE* vollständig bis Rev. 000 beendet, bevor ein Neustart inklusive Fortsetzung der Nachfolgeinstallation zur Anwendung gelangt.

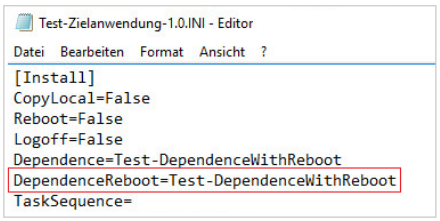
Bei diesem Szenario können Sie in jedem Fall einen *Soft Reboot* (3010), in der alten Version des Pakets einsetzen. Rückgabewerte mit 1641 sind hier nicht erforderlich.

3.12.1.3 Neustartanforderungen nach Abhängigkeitsinstallationen

Für die Installation einer Software, für die ein Neustart nach einer Abhängigkeitsinstallation erforderlich ist, ist genau nach folgendem Ablauf vorzugehen. Andere Umsetzungsvarianten können zu einem unbeabsichtigten Verhalten führen.

Folgende Schritte sind durchzuführen:

1. Die Abhängigkeit ist nach den Richtlinien von Kapitel *3.12.1.1 Neustartanforderung innerhalb und zwischen Revisionen* umzusetzen. Das heisst, dass in der Abhängigkeit, hier im Paket *Test-DependenceWithReboot-1.0* der INI-Eintrag *CommitRebootExitCode=True* in der *Install*-Section gesetzt werden muss und sichergestellt ist, dass dort in einer *Revision* 3010 (eher nicht empfohlen) oder 1641 (empfohlen) zurückgegeben wird. Eine nachfolgende *Revision* muss den Rückgabewert 0 aufweisen.
2. Im Paket *Test-Zielanwendung-1.0* ist zusätzlich zum *Dependence*-Eintrag der INI-Eintrag *DependenceReboot=Test-DependenceWithReboot zu* setzen. So überprüft der *Package-Launcher*, ob diese Abhängigkeit installiert wurde und danach ein Neustart durchgeführt wurde.

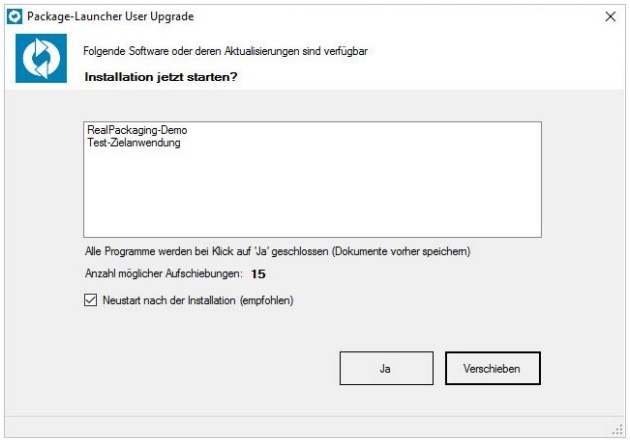


In diesem beschriebenen Testfall dürfte die Installation des Pakets *Test-Zielanwendung-1.0* erst nach einer Installation von *Test-DependenceWithReboot-1.0* und nach darauffolgendem Neustart durchgeführt werden.

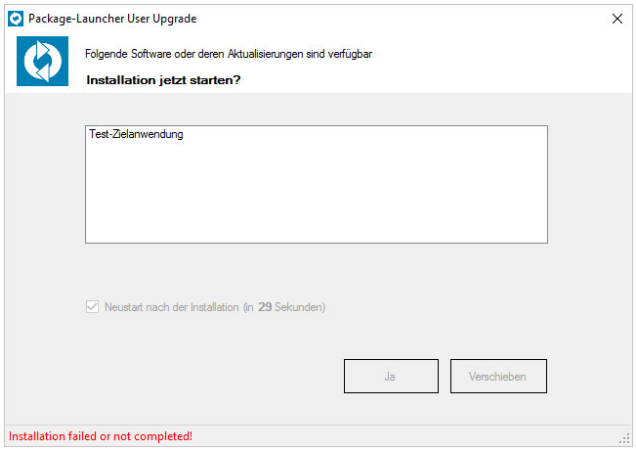
Beispiel nach Abhängigkeit mit dem Package-Launcher App-Installer

Im Deploymentablauf mit dem *Package-Launcher App-Installer* zeigt sich das Verhalten folgendermassen:

1. Erstaufruf. Anzeige aller angebotenen *Applications,* inkl. der Zielanwendung, die eine Abhängigkeit mit Neustartaufforderung enthält:



2. Anzeige nach der Installation durch den Benutzer (vor dem automatischen Neustart)



3. Nach dem Neustart werden die fehlenden *Revisionen* der Abhängigkeit *Test-Dependence-WithReboot-1.0* und die fehlende Zielversion *Test-Zielanwendung-1.0* **automatisch** nachinstalliert (ohne Benutzerinteraktion mit dem *Package-Launcher App-Installer*).

Anzeige im History.LOG

Im *History.LOG* wird nach dem ersten Installationsversuch von *Test-Zielanwendung-1.0,* bzw. nach deren Abhängigkeitsinstallation *Test-DependenceWithReboot-1.0* die Installationsfortsetzung unterbrochen und ein Neustart wird verlangt/ausgeführt (hier in der Zeitachse mit der roten Linie markiert).

23.12.2016 13:25:03	APPL	INSTALL	Test-DependenceWithReboot-1.0	001	Success	Operation completed successfully (reboot commited, installation not completed)
23.12.2016 13:25:04	APPL	INSTALL	Test-Zielanwendung-1.0	001	Error	This package requires a reboot after the installation of following products: Test-DependenceWithReboot
23.12.2016 13:29:56	APPL	UPDATE*	Test-DependenceWithReboot-1.0	002	Success	Operation completed successfully
23.12.2016 13:29:58	APPL	INSTALL*	Test-Zielanwendung-1.0	001	Success	Operation completed successfully

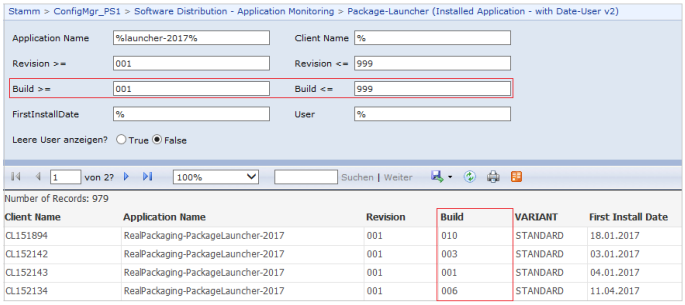
Nach dem Neustart werden die fehlenden *Revisionen* aus *Test-DependenceWithReboot-1.0* und die fehlende Zielapplication *Test-DependenceWithReboot-1.0* automatisch nachinstalliert.

3.13 Build-Informationen

Mittels einem speziellen *Build*bezeichner in der INI-Datei (Ausnahme, bspl. *Build=002*) oder einer speziellen *Build*-Datei (Standard), abgelegt im *Revisions*verzeichnis, kann der *Build* eines Softwarepakets erhöht werden. **Eine *Build*-Datei muss nur dann verwendet werden, wenn an einer produktiven *Revision* Veränderungen durch den Paketierer durchgeführt werden.** Wird bei der Erst-Inbetriebsetzung eines Softwarepakets (*RTM*) beispielsweise eine Konfiguration verwendet, die später nicht im Rahmen eines *Revisionsupdates* korrigiert werden kann, sondern die Anpassung des bestehenden Paketes unerlässlich macht, so erstellt der Softwarepaketierer gleichzeitig eine *Build*-Datei (*PACKAGE_REVISION.BLD*) mit einer 3-stelligen fortlaufenden *Buildnummer*, welche er im angepassten *Revisions*verzeichnis anfügt. Die *Buildnummer* ist ein Zähler der sich auf das ganze Produkt inkl. aller *Revisionen* bezieht und pro Anpassung um ‚1' erhöht wird (beginnend mit 002 bei der ersten Korrektur).

Eine Volage dieser Datei befindet sich im Work-Verzeichnis des Softwarepakets, welche von dort kopiert und in das entsprechende *Revisions*verzeichnis eingefügt werden kann. Mit der *Build*-Datei lassen sich modifizierte *Revisionen* später im Softwareverteilungswerkzeug identifizieren, wodurch Aussagen gemacht werden können, wer vorher die Software installiert hatte (*Build* 001) und wer die neue Variante verwendet (bspl. *Build* 002). Auch die *Package-Launcher* Reports zeigen die Informationen an. Darüber lassen sich zudem genaue Abfragen über die Suche realisieren.

Neben den Anzeigeoptionen können mithilfe der *Build*nummer auch *Inplace-Updates* durchgeführt werden. Das bedeutet, dass das Paket in einer neuen Ausprägung einen *REPAIR* auslöst und Ressourcen auf dem Client aktualisieren kann. Siehe Kapitel *3.13.1 DMBuild (Inplace-Update)*



Die *Buildnummer* wird unter dem Registrykey *HKLM\Software\Real Packaging\Package-Launcher\Packages\<PACKAGE>\Build* geführt und wird bei jeder Operation (*INSTALL*, *REPAIR*, *REMOVE*), erfolgreich oder nicht, neu geschrieben, solange das Produkt als installiert markiert gilt.

Achtung
Nach einer produktiven Softwareeinführung sind in der Regel alle Änderungs- und Erweiterungsanträge kumulativ in einer neuen *Revision* abzubilden. Nur wenn dieses Vorhaben nicht realisierbar ist, darf die Veränderung bestehender, bereits produktiver *Revisionen* in Betracht gezogen werden!

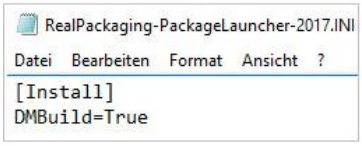
3.13.1 DMBuild (Inplace-Update)

Mittels dem Bezeichner *DMBuild=True* in der INI-Datei und der Applizierung einer *Build*-Datei in einer RTM-*Revision* (eine *Revision*, die bei der produktiven Erstüberführung im Paket enthalten

war), lässt sich ein ***Inplace-Update*** durch einen REPAIR der *RTM-App* auslösen. Zu diesem Zweck wird die SCCM-Detection der *RTM-Application* mit der Prüfung der *Build*-Nummer erweitert.

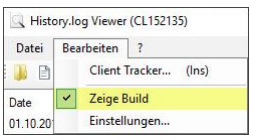
Es ist vorgängig zu prüfen, ob die Aktualisierung die Anforderungen erfüllt und ob alle notwendigen Ressourcen durch den Aktualisierungsprozess aktualisiert werden (manueller Test der einszelnen Schritte).

Das Paket *RealPackaging-PackageLauncher-2020* ist ein solches Paket, welches diese Mechanismen verwendet. Es enthält nur eine *Revision* 001, welche bei Bedarf mit neuen Ressourcen aktualisiert wird.



3.13.2 Vorgehen zur Erstellung eines Inplace-Updates

1. Erstellen Sie ein Backup Ihres bestehenden Softwarepakets
2. Ändern Sie die Ressourcen des bestehenden Softwarepakets. Überschreiben Sie dabei gleich benannte Dateien.
3. Fügen Sie eine *Build*-Datei mit einem um einen Zähler erhöhten Wert in ein *Revision*sverzeichnis ein, das bei der ersten RTM-Überführung bereits existierte. In der Regel handelt es sich um die *Revision* 001.
4. Fügen Sie den Bezeichner *DMBuild=True* der bestehenden INI-Datei hinzu.
5. Testen Sie das Verhalten: Installieren Sie die alte Ausprägung des Pakets. Setzen Sie den Test fort, indem Sie die neue Ausprägung des Pakets "darüber" installieren.
6. Prüfen Sie die Tasks mit dem *History.LOG Viewer*. Wählen Sie dabei die Option "Zeige Build", damit überprüft werden kann, ob die richtige *Build*nummer zur Anwendung gelangt ist.



Date	Type	Task	Package	Revision	Build	Status	Text
01.10.2018 12:49:25	APPL	REPAIR	RealPackaging-PackageLauncher-2020	001	003	Success	Operation completed successfully

7. Überprüfen Sie, ob alle Ressourcen auf dem Gerät durch den Aktualisierungsprozess korrekt installiert wurden.
8. Überprüfen Sie auch die isolierte Einzelinstallation des veränderten Softwarepakets auf einem Endgerät, wo die Software noch nicht installiert war.

9. Überprüfen Sie zudem, ob die Installation der neuen Ausprägung auch nach der vorletzten oder noch früheren Varianten des Pakets erfolgreich ist.

Achtung

Bedenken Sie, dass Sie mit der Aktualisierungsmethode des *Inplace-Updates* aus Sicht der isolierten Paketbetrachtung zwar Transparenz gewinnen, auf der anderen Seite verlieren Sie die chronologische Transaktions- und Änderungsübersicht. Daher ist es empfehlenswert, dass Sie die alte Ausprägung des Softwarepakets vorgängig sichern.

Überdies können nicht alle Ressourcen in dieser Form appliziert werden: Enthält das Softwarepaket eine MSI-Datei, dann können nur geänderte und bestehende Ressourcen über die MSI-Datei aktualisiert werden. Werden in der bestehenden MSI-Datei hingegen Erweiterungen (bspl. neue Dateien) hinzugefügt, werden diese neuen Dateien in der Regel im Rahmen des Inplace-Updates über den *REPAIR* nicht kopiert.

3.13.3 CopyOnlyScriptsOnBuild

Mittels dem Bezeichner *CopyOnlyScriptsOnBuild=00x* in der *Install*-Section der INI-Datei, **zusammen mit DMBuild**, lassen sich die Anforderungen umsetzen, …

- die INI-Datei,
- den lokalen Cache aller RTM-*Revisionen* (nur diese),
- sowie alle Scripts der RTM-*Revisionen* unter %PL%\..\Scripts zu aktualisieren…

…ohne die Software selbst neu zu installieren oder reparieren zu müssen.

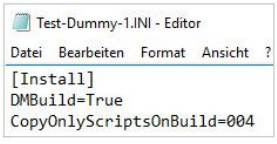
Im *History.LOG* zeigt sich eine solche Transaktion dann folgendermassen:

13.06.2017 15:31:31	APPL	COPYBLD*	Test-Dummy-1	001	Success	Operation completed successfully
13.06.2017 15:31:31	APPL	COPYBLD*	Test-Dummy-1	002	Success	Operation completed successfully
13.06.2017 15:31:32	APPL	COPYBLD*	Test-Dummy-1	003	Success	Operation completed successfully
13.06.2017 15:31:32	APPL	UPDATE*	Test-Dummy-1	004	Success	Operation completed successfully

Es ist sichergestellt, dass auch bei Clients, welche nach der Überführung des *COPYBUILDS* bis zu einem späteren Revisionsupdate, die Aktualisierung nicht erhalten haben (bspl Benutzer in Ferien), zuerst die Aktualisierung der Scripts, Content und INI aus den *RTM-Revisionen* appliziert bekommen, bevor ein allfälliges, in der Zwischenzeit entstandenes *Revisions*update folgt (siehe *History.LOG* aus der letzten Abbildung).

Diese Implementation macht Sinn, wenn INI-Einträge geändert haben oder wenn Scripts – insbesondere für eine erfolgreiche spätere DeInstallation – aktualisiert werden müssen. Oft will man in solchen Fällen die Software nicht erneut installieren.

Die Integration in der INI sieht folgendermassen aus:

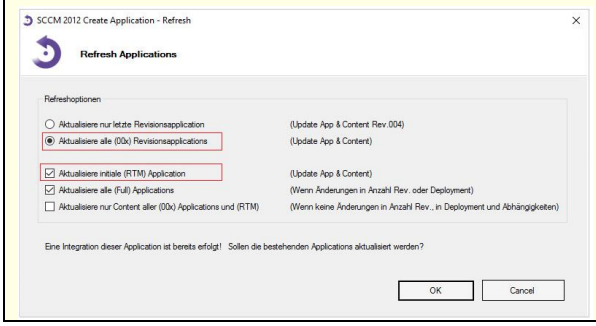


Über *CopyOnlyScriptsOnBuild* wird genau die *Build*-Nummer angegeben, bei welcher die Transaktion nur die INI, Content und Scripte aktualisieren soll. Lokal muss zudem die Bedingung erfüllt

sein, dass dort zum Zeitpunkt der Transaktion die registrierte *Build*nummer < der hier angegebenen *Build*-Nummer beträgt.

Achtung

Bei der Überführung mit *SCCMCreateApp* sollten beide rot markierten Einstellungen markiert werden!



3.13.3.1 Unterschiede bei der Aktualisierung

Sollen INI, Content oder Scripte aktualisiert werden, kann folgende Tabelle die Unterschiede veranschaulichen:

Implementierung	was wird aktualisiert			was wird sonst noch gemacht
Normales Revisionsupdate	INI	Content der neuen Revision in Cache, wenn CopyLocal=True	alle Scripts der neuen Revision	+ installiert neue Revision
Inplace-Update mit DMBuild	INI	Content der RTM-Revisionen in Cache, wenn CopyLocal=True	alle Scripts aller Revisionen	+ REPAIR aller Revisionen[1)]
dito mit DMBuild + CopyOnly…	INI	Content der RTM-Revisionen in Cache, wenn CopyLocal=True	alle Scripts aller Revisionen	**+ installiert nichts**

[1)] bei CopyLocal=True, sonst nur RTM-Revisionen

3.13.4 Build in INI

Wird die *Build*-Nummer in der INI-Datei zusammen mit dem Bezeichner 'AutoBuild=True' einmalig integiert, dann erhöht S*CCMCreateApp* die *Build*-Nummer selbständig bei jeder produktiven Überführung (Detection Method und in der Paket-INI-Datei). Seien Sie vorsichtig und verwenden Sie *Build*-Informationen der Transparenz wegen nicht an beiden Orten (INI und .BLD-Datei).

3.13 Software-Inventarisierung

Für die korrekte Softwareinventarisierung über *SCCM* ist der *Package-Launcher* zuständig. Der *Package-Launcher* ist der erste Prozess, der über die Änderung des Installationsstatus einer Software Bescheid weiss. Er agiert somit als Vermittlerinstanz gegenüber *SCCM* und verwaltet den Status aller Softwareinstallationen.

Die Aktualisierung des Softwareinventars an *SCCM* erfolgt als *Delta* nach jeder Software-Statusänderung und nur, wenn die Softwareinstallation nicht im Rahmen einer Computer-Erstinstallation (Environmentvariable *_SMSTSType*=2) erfolgt und der Registrykey *HKLM\SOFTWARE\Real Packaging\Package-Launcher\StopLauncherInventory* nicht mit einem Wert belegt ist. *StopLauncherInventory* kann folgende Ausprägungen besitzen:

StopLauncherInventory = Leer oder „False“: Inventory wird durch den *Package-Launcher* ausgeführt **(Standardeinstellung).**

StopLauncherInventory = True	Für mindestens 2 Stunden ab der Installation des nächsten Softwarepakets wird das *Inventory* durch den *Package-Launcher* nicht mehr ausgeführt **(Standardausnahme).**
StopLauncherInventory = Always	Das *Inventory* wird auf diesem Computer durch den *Package-Launcher* nie mehr ausgeführt (nicht empfohlen).

3.13.1 Die für SCCM massgeblichen Schnittstellenregistrykeys

Folgende Registrykeys werden im Rahmen der Inventarisierung an *SCCM* übermittelt:

1. HKLM\Software\Real Packaging\Package-Launcher\Packages\<PACKAGE>\Revision

   Gibt den Installationsstatus wieder. Jeder Wert > 000 besagt, dass das Softwarepaket auf dem Computer installiert ist. Entsteht bei der Erstinstallation eines Softwarepakets bei der *Revision* 001 ein Fehler, so wird der Key danach immer noch leer sein (*Revision*=""). Bei einer vollständigen Deinstallation trägt der Key den Wert „000".

2. HKLM\Software\Real Packaging\Package-Launcher\Packages\<PACKAGE>\Build

   Wird das Paket bei einer Korrektur nicht mit einer neuen *Revision* ergänzt, sondern Korrekturen an einer bestehenden *Revision* gemacht, erhöht sich die *Build*-Nummer. Standardmässig steht dieser Eintrag nach einer Softwareinstallation auf 001. Die *Build*-Nummer kann sich bei allen Installationstransaktionen nur erhöhen. Siehe Kapitel *3.13 Build-Informationen*.

3. HKLM\Software\Real Packaging\Package-Launcher\Packages\<PACKAGE>\Error

   Rückgabestring bei einer fehlerhaften Installationstransaktion. Der *Package-Launcher* gibt einen Rückgabewert > 0 zurück.

   - Bei einer *MSI*-Installation enthält dieser Key die Fehlermeldung aus der *Windows Installer* Protokolldatei als *Reintext*.
   - Bei *Legacy-Installationen* enthält dieser Key die Fehlermeldung die vom Paketierer vorgesehen ist und in der *PreInstall_00x.EXE* abgebildet ist.
   - Der *Package-Launcher* protokolliert allgemeine Fehler auch unter diesem Key als *Reintext*.

4. HKLM\Softw..\Real Packaging\Package-Launcher\Packages\<PACKAGE>\LastAccess
   Der Ausführungszeitpunkt der letzten Transaktion dieses Softwarepakets

5. HKLM\Softw..\Real Packaging\Package-Launcher\Packages\<PACKAGE>\Variant
   Die Bezeichnung der Paketvariante. Siehe Kapitel *7.1 Paketvarianten*

4 Best-Practice Regeln und Limitierungen

Viele der hier skizzierten Empfehlungen sind als allgemeine Vorschläge zu betrachten und verstehen sich als *Best-Practice*-Regeln.

4.1 Zielverzeichnis

In der Regel sollten alle Anwendungen in einem Unterverzeichnis von *%ProgramFiles%* installiert werden. Für die Unterverzeichnisse in *%ProgramFiles%* werden die Standardvorgaben des Herstellers verwendet.

In einzelnen Ausnahmefällen, wo dies nicht möglich ist (bspl. Oracle), muss dies entsprechend dokumentiert werden.

4.2 Startmenü und ShortCuts

Es werden ausschliesslich Verknüpfungen in Unterverzeichnissen vom *All Users Startmenü* erstellt. Verknüpfungen aus verschiedenen Softwarepaketen, die aber einen logischen Zusammenhang ergeben, werden falls möglich in einem gemeinsamen Unterordner abgelegt. Für alle anderen Fälle kann der Standardpfad des Anbieters übernommen werden.

Beispiele von Unterverzeichnissen:

- Adobe, Microsoft

Folgende Shortcuts werden falls möglich unterdrückt oder entfernt:

- Verknüpfungen auf dem Desktop
- Online Updates
- Online Registrierung
- Deinstallation

Die Anpassungen erfolgen durch den Softwarepaketierer über eine *MST*-Datei oder via *Pre/PostInstall*.

4.3 Lizenzen

Es können nur Programminstallationen automatisiert werden, welche einen interaktionslosen Installations-, Konfigurations- und ggf. Aktivierungsmechanismus bieten. Dies gilt speziell auch im Umfeld von Lizenzierungsvorgängen. Während des Installationsvorgangs selbst ist es in der Regel nicht möglich, eine entfernte, aber verfügbare Lizenzquelle (Server) zu prüfen (Installation im Systemkontext!). Sofern die Setup-Routine der Anwendungssoftware über einen interaktionslosen Aktivierungsvorgang verfügt, wird dieser in das Softwarepaket integriert. Bei repaketierten Softwarepaketen ist darauf zu achten, dass die integrierte Lizenzaktivierung nicht auf den Entwicklungscomputer und Entwicklungsbenutzer beschränkt, sondern allgemein gültig ist.

4.4 Abhängigkeiten (Middlewares)

Die Abhängigkeiten zwischen einzelnen Anwendungen müssen vom Softwarepaketierer erkannt und dokumentiert werden. Zu beachten ist, dass bei Herstellersetups oftmals im Hintergrund Abhängigkeiten installiert werden, deren Installation nicht offensichtlich ist. Da im Vorbereitungsprozess aus Sicht der Paketierung durch Applikationsverantwortliche oder Testmanager die Produkte nicht selten zu wenig analysiert werden, finden sich denn auch häufig weit mehr Abhängigkeiten, als im Auftrag dokumentiert ist.

Abhängigkeiten werden in zwei Gruppen unterschieden:

1. Globale Abhängigkeit:

   Abhängigkeiten, die von mehreren Softwarepaketen verwendet werden können: All diese Anwendungen müssen als eigenständige einzelne Pakete behandelt und separat paketiert werden.

2. Lokale Abhängigkeit

   Abhängigkeiten, *Middleware* und Produkte-Teilinstallationen, die nur von der zu paketierenden Haupanwendung verwendet werden und auch künftig kaum von Drittapplikationen installiert werden: Diese Abhängigkeiten werden in der Regel im selben Softwarepaket in einer Vorrevision platziert. So können bei der Gestaltung des initialen Softwarepakets auch mehr als nur eine *Paket-Revision* gleichzeitig implementiert werden.

   Beispiel:

   Revision 001 -> lokale Abhängigkeit 1
   Revision 002 -> lokale Abhängigkeit 2
   Revision 003 -> Hauptanwendung

Auf die Möglichkeiten zur Lokalisation von Abhängigkeiten wird im Kapitel *6.3.1 Auspacken von Installationselementen aus einem Bootstrapper* näher eingegangen.

Im Kapitel *6.10.3 Umgang mit Abhängigkeiten durch Verwendung des „Dependence"-Eintrages* finden wir zudem Hinweise, wie die Prüfung auf Abhängigkeiten mit einem Softwarepaket realisiert wird.

Schliesslich muss bei globalen Abhängigkeiten in *SCCM* noch eine *Customized Task Sequence* eingerichtet werden. In dieser werden die voneinander abhängigen *SCCM*-Pakete/Install-Programme in der korrekten Reihenfolge für die Installation zusammengefasst. Das Script *Create_SCCM_Package_Link.vbs* (*SCCM 2007*) erstellt diese *Task-Sequenz* automatisch anhand der *Dependence*-Einträge.

4.5 Versionshandling

Für das Versionshandling wird folgende Anwendung empfohlen:

Für die Softwareablage und durch Scripts anschliessend auch für *SCCM* übernommene Namens-, bzw. Versionsbezeichnung hat der Softwarepaketierer folgende Syntax zu verwenden:

Major.Minor (bspl. 1.1, 12.2, aber auch 10.55)

In der Regel wird die gekürzte technische Version aus der *MSI*-Datei *(Property ProductVersion)* als Basis für die Versionsbezeichnung verwendet. Bei repaketierten Softwarepaketen kann als *MSI-ProductVersion*-Property die im Auftrag, in der Installationssource oder auf der Herstellerseite angegebene Version verwendet werden, welche schliesslich als Basis für die Softwareablage zu verwenden ist.

Allgemeine Namensrichtlinien sind im Kapitel *3.7 Namensrichtlinien* dokumentiert.

4.6 Umgang mit .HLP-Dateien

Um 32-Bit-Hilfedateien mit der Dateinamenerweiterung "*.HLP*" auf *Windows 7* anzeigen zu können, wird ein installiertes *WinHlp32.exe* benötigt. Diese Anwendung steht in Form eines Softwarepakets zur Verfügung, welches punktuell als Abhängigkeit bei diesen Softwarepaketen zu implementieren ist, wo zwingend ein Zugriff auf *.HLP*-Dateien benötigt wird. Zusätzlich ist das Softwarepaket in den *Dependence*-Bezeichner der *INI*-Datei aufzunehmen, wodurch die Abhängigkeit nach Überführen mit *SCCMCreateApp_Link.vbs* auch in die *Application* eingebaut wird. Das Softwarepaket, welches *WinHlp32.exe* installiert heisst ***Microsoft-Helpfiles-KB917607***.

4.7 Umgang mit VirtualStore

Siehe Kapitel *6.7.1 Datei- und Registrierungsvirtualisierung*

4.8 Installationskontext

Es werden nur Installationen unter dem Kontext *per-machine* empfohlen.

4.9 Firewall

Die Ausnahme-Regelung wird global entweder per *Group-Policies* gesteuert und innerhalb des Softwarepakets nach Vorlage definiert (siehe \Work\00x\Snipplets.TXT).

Achtung: In einzelnen Fällen werden auch nur bei der manuellen **Installation** Verbindungen gestoppt. *Firewall*-Ausnahmen sind aber nur erforderlich, wenn diese für den **Betrieb** der Software

benötigt werden. In der Paketdokumentation ist auf die Ausnahmeregelung **deutlich hinzuweisen**!

Tip: Portinformationen sind beispielsweise über das Sysinternals-Tool *TCPView* ermittelbar. Beim Erscheinen der Firewall-Meldung werden mittels diesem Tool die Art des Protokolls und der Port einsehbar, auf den ein Programm zugreift.

4.10 Paketierungsarten

Grundsätzlich kann ein Softwarepaket auf verschiedene Weise erstellt werden. Prinzipiell ist die **erste Wahl die Verwendung von Hersteller-*MSI*-Dateien**, die im Rahmen eines *Customizing* an die Unternehmensbedürfnisse angepasst werden. Oft befinden sich auch in *Setup.EXE* Dateien *MSI*-Dateien, die der *Setup-Bootstrapper* während der Installation auspackt (siehe Kapitel *6.3 Verwenden eines Hersteller MSIs*.

Zwar erlaubt die ***Re**paketierung* eines Setups ein einfaches und meist schnelles Erstellen von Softwarepaketen mit benutzerdefiniertem Verhalten. Es darf aber nicht vergessen werden, dass ein *repaketiertes* Setup eine komplett andere Logik, als das Herstellersetup aufweist und auch nur eine eingefrorene Zustandsänderung von Registrykeys, Dateien, Services, etc. beinhaltet. *Windows Installer* Setups sind meist viel komplexer aufgebaut, als dass sie nur eine Summe dieser besagten Ressourcen wären.

Die Paketierungsart ist unbedingt nach der unten dokumentierten Reihenfolge zu wählen:

1. Hersteller-*MSI*
2. Snapshot-*MSI* (Repaketierung) oder AppV
3. Legacy-Setup, ausgeführt über Kommandozeilenparameter via *PreInstall* oder eigenem Script

Ist die Verwendung eines Hersteller-*MSIs* nicht möglich (*Legacy-Setup*) oder ist diese Paketierungsart nur mit unverhältnismässig grossem Aufwand zu bewerkstelligen, so kann als zweite Wahl das Paket mittels Techniken der *Repaketierung (Snapshot)* erstellt werden. Bei diesem Verfahren erhält man eine *MSI*-Datei, die gegenüber einem direkt aufgerufenen *Legacy-Setups* immer noch folgende Vorteile bietet (nicht abschliessend):

- Sie ist transparent.
- Man kann globale Einstellungen (beispielsweise Einstellungen für *Add/Remove Program (ARP)*) und Applikationseinstellungen (bspl. Servernamen) anpassen.
- Es können Dateien und Komponenten aus der Installation ausgeschlossen werden.
- Die Installation kann auf Softwarekonflikte getestet werden *(Conflict Explorer 2009)*.

Die Verwendung von *Legacy-Setups* über einen Aufruf im *PreInstall* oder eigenem Script (*INI:ExecuteFile*) ist nur in Ausnahmefällen zulässig. Für hardwarenahe Treibersoftware kann dies eventuell sinnvoller sein, als die *Repaketierung* des *Legacy-Setups*.

4.11 Pfadlänge

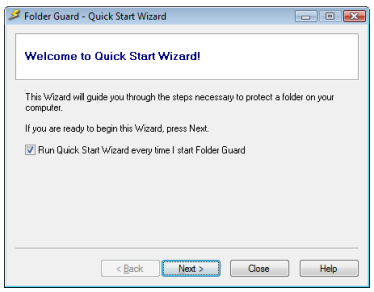
Die maximale Pfadlänge welche von NTFS unterstützt wird beträgt 32’767 Zeichen (individuelle Elemente im Pfad: max. 255 Zeichen).

Das eigentliche Windows API unterstützt aber nur 260 Zeichen für die maximale Pfadlänge (*http://msdn.microsoft.com/en-us/library/aa365247.aspx*)! Das heisst, dass der Pfad zu den *SCCM* Paket-Sourcedateien diesen Wert nicht überschreiten darf.

Bei Hersteller-*MSI*-Paketen, die externe Dateien in Unterverzeichnissen verwenden, wird zur Verkürzung der Pfadlänge die Verwendung von *MakeCab.vbs* empfohlen, damit man *Cabinettdateien* erstellen und die extern vorliegende Source anschliessend löschen kann. Die Ausführung ist zu diesem Zweck auf einem fixen Laufwerk empfohlen, damit man nicht auch bei der Ausführung von *MakeCab* an die Pfadlängengrenzen kommt. Siehe Kapitel *6.3.12 Verwendung von MakeCab.vbs*

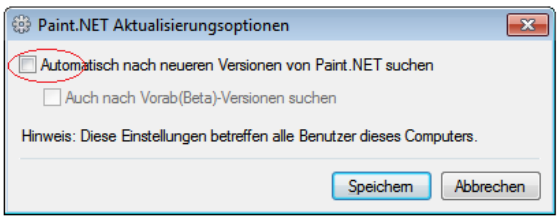
4.12 Automatisierung der Benutzereinstellungen

Notwendige Benutzereinstellungen sind nach Vorgabe zu implementieren. Zudem sollten unnötige Dialogboxen, die während dem Starten der Anwendung erscheinen und die sich ausblenden lassen, automatisiert für alle Benutzer ausgeblendet werden (Siehe auch Kapitel *6.5 Umgang mit Benutzerressourcen*). Beispiel einer auszublendenden Anzeige:



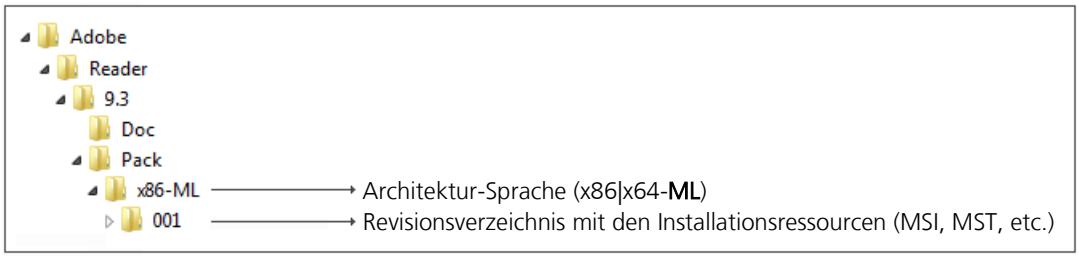
4.13 Keine automatischen Updates

Bietet das Programm Einstellungen zur Auswahl, ob Updates automatisch heruntergeladen oder installiert werden sollen, so sind diese Einstellungen durch den Paketierer im Softwarepaket auf „keine Updates hinunterladen/installieren“ zu setzen. Manchmal entspricht der Funktionalität auch nur ein entsprechender Registrykey. Handelt es sich bei den Einstellungen um Benutzereinstellungen (bspl. HKCU), dann ist das Vorgehen aus Kapitel *6.5 Umgang mit Benutzerressourcen* anzuwenden.

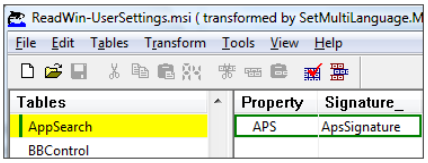


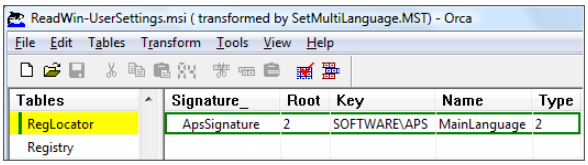
4.14 Sprachen und Spracheinstellungen

Handelt es sich bei der Software um eine *MUI*-fähige Version, wo die Sprache des *GUI* nicht automatisch ändert, sondern innerhalb der Applikation ausgewählt werden kann und findet sich eine einfache Repräsentation der Einstellung in der Registry oder über eine editierbare Konfigurationsdatei (bspl. *INI*-Datei), so ist das Design des Softwarepakets so vorzusehen, dass dieses die **Standardsprache nach Sprachausprägung des Betriebssystems automatisch einstellt**. In diesem Fall ist der Registrykey *HKLM\Software\Real Packaging\Package-Launcher\MainLanguage* auszulesen (Variable %LANGUAGE%). Dieser Registrykey/Variable enthält den Sprachcode „*GE*", „*FR*", „*IT*" oder „*EN*". Je nach Inhalt dieses Registrykeys können nun die erforderlichen Einstellungen durch das Softwarepaket vorgenommen werden. Ein *MUI*-fähiges Softwarepaket ist im Paketunterverzeichnis *Architektur-Sprache* abzulegen, also beispielsweise unter ***x86-ML***.



Das gleiche Verfahren ist auch anzuwenden, wenn es sich nicht um eine *MUI*-fähige Software im klassischen Sinne handelt. D.h., wo die *GUI*-Sprache der Software zwar nach der Installation nicht veränderbar ist, wo aber das Installationssetup oder die Hersteller-*MSI*-Datei mehrere Sprachvarianten enthält. In diesem Fall ist durch den Softwarepaketierer ebenfalls der Registrykey *HKLM\Software\Real Packaging\Package-Launcher\MainLanguage* auszulesen und die Sprachkomponenten sind dynamisch je nach Inhalt des Registrykeys zu installieren. Anbei finden Sie ein Beispiel zum Auslesen des *MainLanguage*-Registrykeys aus einer *MSI*-Datei mittels einer *MST*-Manipulation:





Werden hingegen verschiedene Sprachvarianten vom Hersteller geliefert (beispielsweise in Form verschiedener *MSI*-Dateien), so sind durch den Softwarepaketierer verschiedene Sprachverzeichnisse im Softwarepaket zu erstellen, wo die Ressourcen eingepflegt werden. Der *Package-Launcher* ermittelt hier automatisch, welche Installationsressourcen aus welchem Ordner bei der Installation angewendet werden sollen.

Weitere Informationen zu der Ablagestruktur finden Sie im Kapitel *5.1 Entwicklungsumgebung*.

4.15 Installation im Systemkontext

Softwarepakete müssen sich im Systemkontext installieren lassen. Zu beachten ist, dass im Systemkontext insbesondere der Zugriff auf externe Netzwerkressourcen und Netzwerkverzeichnisse nicht möglich ist. Es gibt Hersteller-Setups, die anders reagieren, wenn sie unter dem Systemkontext installiert werden!

4.16 Silent Installationen

Alle Softwarepakete müssen silent installierbar sein. Problematisch sind insbesondere auch Anzeigedialogboxen, die sich während einem Fehler zeigen und die die weitere Ausführung der Installation verhindern können, wenn diese Softwarepakete ohne Desktop mittels *SCCM* installiert werden.

4.17 Verwendung von variablen Servernamen

Servernamen sind nach Möglichkeit variabel über die *INI*-Datei des Softwarepakets zu implementieren. Dadurch lässt sich bei einem Wechsel des Servernamens eine schnelle Anpassung des Softwarepakets an einem zentralen Ort bewerkstelligen (Siehe auch Kapitel *3.5 INI-Datei*)

5 Handling der Upgrades und Revisionsupdates

Der *Package-Launcher* verwaltet alle Installationen, Software*upgrades* und –*updates* selbständig. Für den Softwarepaketierer bedeutet dies, dass er bei einem *Update*auftrag all seine Änderungsabsichten immer in einem (oder in Ausnahmefällen mehreren) zusätzlichen *Revisions*verzeichnis/sen abbilden muss. Das Vorgehen hierzu ist recht einfach: Durch das Kopieren der Installationsressourcen *(MSI, MST, MSP*, etc.*)* in das neue *Revisions*verzeichnis ist nämlich ein grosser Teil der Aufgabe bereits erledigt.

Für den Softwarepaketierer ist es erforderlich, dass er die Grundlageninformationen aus dem Kapitel 2 & 3 gut kennt. Wir finden insbesondere im Kapitel *2.2 Updates und Upgrades* und im Kapitel *3.6 Realisierung eines Upgrades* weitere Ausführungen zu diesem Thema.

Einige Beispiele der Umsetzung von Aufträgen in das Design des Softwarepakets:

Form des Softwarepakets:	Initialversion
Auftrag:	Auftrag über neue Software, die es in der Umgebung des Unternehmens in keiner anderen Version gibt und die kein Vorprodukt ablöst. Sogenannte *RTM (Release To Manufacturer)*
Umsetzungsbeispiel:	Neue Version mit *CreatePackage* erstellen und Bedürfnisse dort in der *Revision 001* (Unterverzeichnis *001*) abbilden.
Form des Softwarepakets:	Upgrade
Auftrag:	Auftrag einer neuen Software, wo es in der Umgebung des Unternehmens ein Softwarepaket einer Vorversion gibt und wo sich die Bedürfnisse nicht in einem *Revisionsupdate* (siehe unten) abbilden lassen.

Umsetzungsbeispiel:	Neue Version mit der selben Namensbezeichnung wie das Softwarepaket der Vorversion (ohne Version) mit *CreatePackage* erstellen und Bedürfnisse dort in der *Revision 001* (Unterverzeichnis *001*) abbilden. Ev. Upgradefunktionalität über den Bezeichner *Upgrade* der Softwarepaket-*INI*-Datei anpassen. Siehe Kapitel *6.10.1 Upgrade Handling*.
Form des Softwarepakets:	**Revisionsupdate** (Beispiel 1)
Auftrag:	Neue Software für die es in der Umgebung des Unternehmens bereits ein Softwarepaket in einer Vorversion gibt und die in Form von Anpassungen und Erweiterungen zu dieser Vorversion implementiert werden kann (bspl. Kopieren zusätzlicher oder neuerer Dateien, etc.). Der Auftrag selbst muss nicht zwingend als Updateauftrag übergeben worden sein!
Umsetzungsbeispiel:	Das Softwarepaket *HP-Demo-1.0* existiert in der *Revision 001*. Nun käme ein Auftrag der Version 1.1, in der einige *DLL*-Dateien aktualisiert werden müssten. Hier kann das bestehende Softwarepaket *HP-Demo-1.0* mit einer zusätzlichen *Revision 002* erweitert werden. In das neu zu erstellende Verzeichnis *002* kopiert der Softwarepaketierer die neue *MSI*-Datei, welche diese einzelnen Dateien kopiert. Die Bezeichnung des Version-Verzeichnisses wird nicht geändert (1.0)!
Form des Softwarepakets:	**Revisionsupdate** (Beispiel 2)
Auftrag:	Nachdem für eine Software ein Softwarepaket erstellt und dieses in die Produktion überführt wurde, müssen Anpassungen an Einstellungen (bspl. *Registry*, *INI*, *Shortcuts*, etc.) vorgenommen werden.
Umsetzungsbeispiel:	Das produktive Softwarepaket *HP-Demo-1.0* existiert in der *Revision 001*. Die neuen Anforderungen **sind nicht** durch Änderung in dieser *Revision 001* umzusetzen! Auch hier kann das bestehende Softwarepaket *HP-Demo-1.0* mit einer **zusätzlichen *Revision 002*** erweitert werden. In das neu zu erstellende Verzeichnis *002* kopiert der Softwarepaketierer die neue *MSI*-Datei, ein *PreInstall*oder Script, welches die Änderungsaufgaben übernimmt. Die Bezeichnung des Version-Verzeichnisses wird nicht geändert (1.0)!
Form des Softwarepakets:	**KeepRevision**
Auftrag:	Nach der Verteilung eines *Upgrades* wurde ein Fehler festgestellt und der weitere Verteilungsauftrag gestoppt. In einer *Revision 001* eines **produktiven** Softwarepakets, welches in Form eines *Upgrades* konzipiert wurde, sollte nun eine Korrektur gemacht werden, welche eine lokal unterschiedliche Einstellungsdatei der Vorversion der Software vor einer künftigen *Upgradeinstallationen* sichert und nach dem *Upgrade* wieder auf das Zielsystem kopiert.

	Die Änderungsabsichten lassen sich nicht in einer zusätzlichen *Revision* realisieren, da die zu sichernde Datei bei der chronologischen Installationsabfolge der *Revisionen* bei Beginn der letzten *Revisions*installation bereits gelöscht, bzw. mit dem Standardinhalt neu geschrieben wäre.
Umsetzungsbeispiel:	Die Änderungen für künftige Installationen sind am bestehenden Softwarepaket der *Revision 001* vorzunehmen. Zusätzlich wird eine (neue) *Build*-Datei in das *Revisions*verzeichnis eingefügt, wo die Änderungen gemacht wurden. In unserem Fall im Verzeichnis der *Revision 001*. Die Bezeichnung der *Build*-Datei unterliegt der Namenskonvention und trägt den Namen des Softwarepakets, gefolgt vom *Revisions*-Bezeichner und der Erweiterung *.BLD* (Bspl. *HP-Demo-1.0_001.BLD*). In der Datei ist ein fortlaufender, 3-stelliger Zähler anzufügen. Bei der ersten Veränderung wäre dieser also auf *002* zu stellen. Im Kapitel *3.13 Build-Informationen* ist die Verwendung der *Build*-Datei detaillierter erklärt.

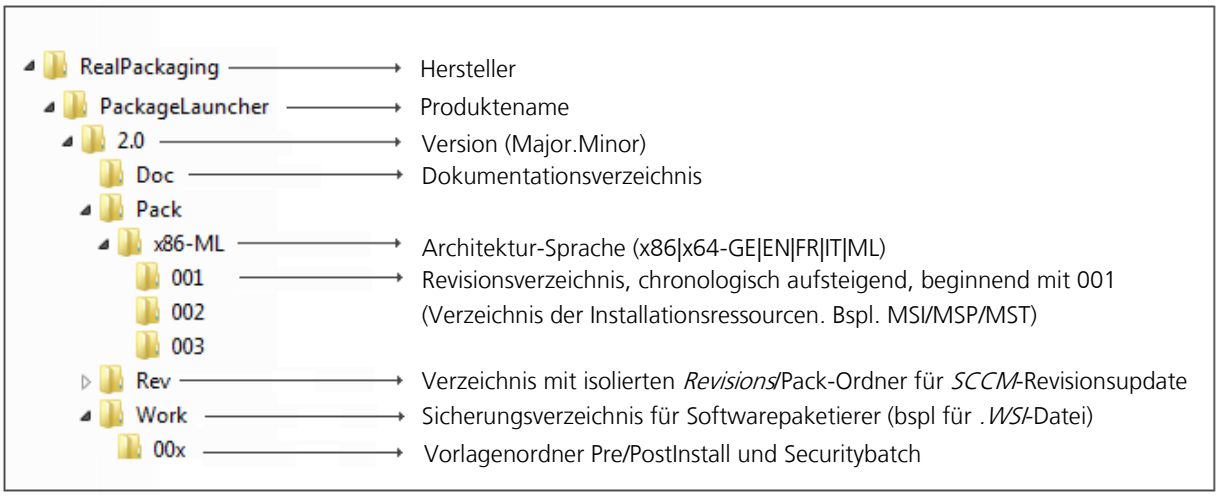
Achtung:

Die Verwendung von ***KeepRevisionen*** sollte **nur wenn zwingend nötig, ausnahmsweise** zum Einsatz kommen! *KeepRevisionen* können auch über Inplace-Updates ausgerollt werden. Siehe Kapitel *3.13.1 DMBuild (Inplace-Update)*

5.1 Entwicklungsumgebung

Für die Entwicklung der Softwarepakete ist ein eigener Share vorgesehen (*PLPackDEV*). Die Erstellung der Ablagepfade der Paketentwicklung erledigt ein Tool mit dem Namen *CreatePackage.EXE.* Im Kapitel *6.2 CreatePackage* wird auf die Bedienung dieses Tools eingegangen.

Beispiel der Paketentwicklungsumgebung:



5.2 Produktionsumgebung

Die Ablagestruktur der Produktionsumgebung wird durch das Script *SCCMCreateApp_Link.vbs* automatisch erstellt und die für dort erforderlichen Daten kopiert (PLPackPRD). Hierbei wird der *Pack*-Ordner inkl. Unterordner und der „*Rev*"-Ordner in einen Ordner/Unterordner des Produktionsshares überführt.

5.3 Namensbezeichnungen

Hersteller: Handelt es sich um eine Freeware *(General Public License)*, so ist als Herstellernamen die Bezeichnung *GPL* zu verwenden. Falls der Herstellername nicht bekannt ist, ist die Bezeichnung *Misc* erforderlich. In allen anderen Fällen verwendet man den Namen des Herstellers.

Produktename: Name der Software

Version: Technische Versionsbezeichnung nach der Syntax *Major.Minor*. In der Regel verwendet man einstellig gekürzte *Minorbezeichnungen*. In Ausnamefällen kann aber auch eine mehrstellige *Minorversion* zum Einsatz kommen.

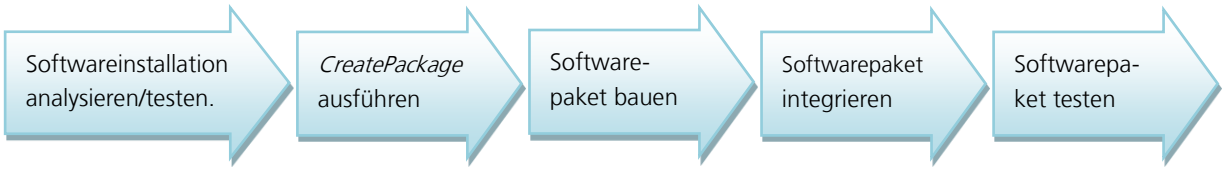
5.3.1 Limitierungen und Einschränkungen

Die Gesamtzeichenanzahl ist auf 39 Zeichen *(Hersteller*, *Name*, *Version*, ohne Bindestriche) beschränkt (nur *SCCM 2007*). Zudem sind folgende Zeichen bei den Namen nicht erlaubt:

- Leerzeichen `" "`
- Bindestrich `"-"` (Wird nur als Paketbezeichnungstrennzeichen *Hersteller-Name-Version* eingesetzt)
- Optionenbezeichner `"(" ")"` Optioneneinleitungszeichen sind für den Package-Launcher reserviert. Beispielsweise werden Shortcut-Pakete damit identifiziert *(S)*.

6 Phasen der Paketerstellung

Die Softwarepaketierung erfolgt in mehreren Schritten und fängt mit dem Auftragseingang, bzw. mit der Zuweisung dieses Auftrages an den Softwarepaketierer an und endet üblicherweise nach diversen Abschlusstests und der Distribution auf die Integrations- oder Produktionsplattform.



Die meisten der nachfolgenden Ausführungen dieses Kapitels werden in der Reihenfolge beschrieben, wie sie im Softwarepaketierungsprozess zum Einsatz kommen.

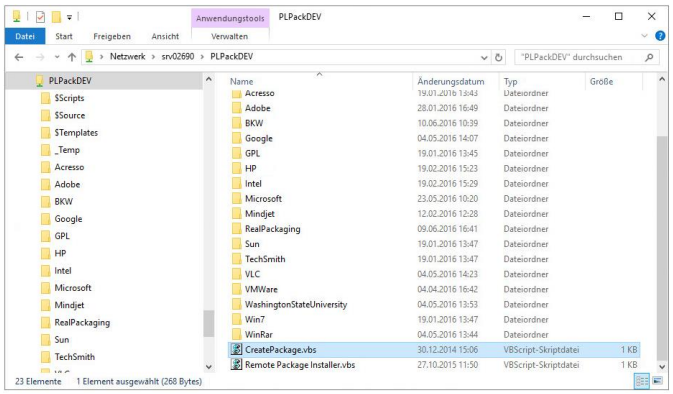
6.1 Vorarbeiten

Bevor mit der eigentlichen Softwarepaketierung begonnen wird, informiert sich der Softwarepaketierer, was er zu erledigen hat, indem er den Auftrag und allfälligen Mailverkehr betreffend der zu implementierenden Anwendung studiert. Zudem macht sich der Softwarepaketierer ein Bild über die Software, die er zu paketieren hat, indem er diese auf seiner *VM-Guest* gem. Anleitung installiert, die Anwendung startet und sich mit allfälligen Eigenschaften des Produktes auseinandersetzt.

6.2 CreatePackage

Mit dem Tool *CreatePackage.EXE* kann die **initiale** Ablagestruktur der Softwarepaket-entwicklungsumgebung erstellt werden *(Revision 001)*. Zudem werden alle für den Software-paketierer notwendigen Vorlagen und Automatisierungsscripts kopiert.

Im Wurzelverzeichnis des *PLPackDEV*-Shares befindet sich das Startscript zum Aufruf von *CreatePackage.* Mit einem Doppelklick starten Sie das Programm:

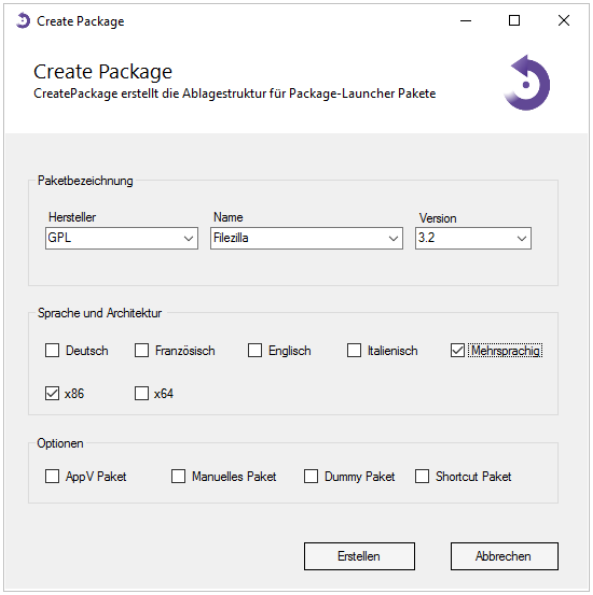


Geben Sie anschliessend die erforderlichen Kenndaten ein. Unter *Sprache* und *Architektur* können mehrere Optionen gleichzeitig getätigt werden. Wird ein Paket erstellt, welches mehrsprachig ist oder die Sprachausprägung in einer Softwareeinstellung durch den Benutzer gewählt werden kann oder wo die Software die Clientsprache selbständig erkennt und dadurch die Spracheinstellungen für die Software selbständig vornimmt, ist als Sprachvariante „Mehrsprachig“ zu wählen.

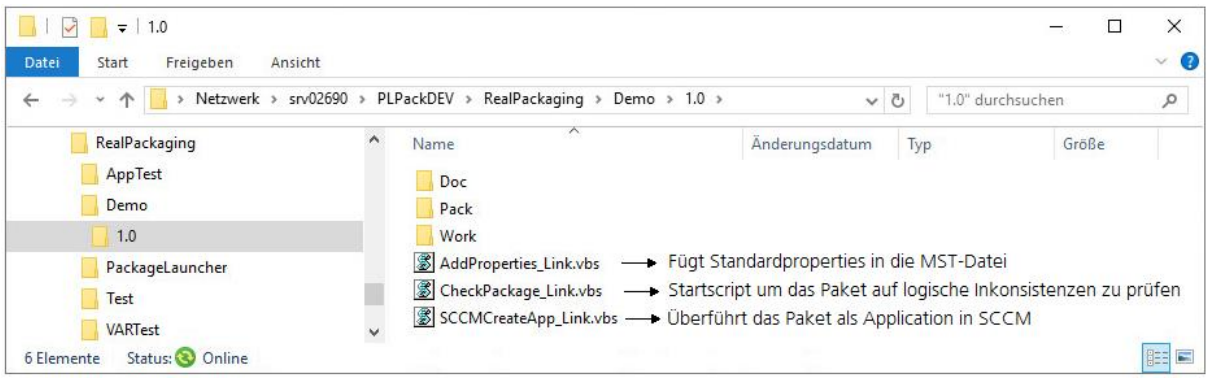
Die Architekturoptionen beziehen sich auf die Installationsinstanz und das Setup. Zielt das Setup auf x86 Rechner, ist hier x86 zu wählen (solche Setups können auch unter x64 eingesetzt werden). Ist das Design des Setups hingegen für x64 Rechner ausgelegt (volle x64er Kompatibilität), wählt man x64 (solche Setups können nur unter x64 installiert werden). Paketiert der Paketierer beide Varianten in einem Paket (zwei verschiedene Setups), so sind beide Optionen auszuwählen.

Die Option „AppV Paket“ fügt dem Paketnamen zusätzlich noch ein „(V)“ an. Mit "Manuelles Paket" kann ein Paket mit Interaktion mit dem Benutzer erstellt werden. "Shortcut Paket" ermöglicht, Pakete mit Shortcuts zu erstellen. Diese Shortcuts können in einer INI-Datei deklarert werden.

Ansicht des Dialogs von *CreatePackage*:



Ablagestruktur, welche mit *CreatePackage* erstellt wird:



6.3 Verwenden eines Hersteller MSIs

Wird vom Hersteller die Installation über eine *MSI*-Datei angeboten, so sind diese Ressourcen als Basis für das Softwarepaket zu verwenden. Siehe auch Kapitel *4.10 Paketierungsarten*. In solchen Fällen wenden wir keine *Repaketierung* an. Ausserdem ist der direkte Einsatz einer Hersteller-*MSI*-Datei auch der Installation eingebetteter *MSI*-Dateien via *Bootstrapper* vorzuziehen. ***Bootstrapper* (Setup.EXE) sind daher immer darauf zu analysieren, ob diese eingebettete *MSI*-Dateien zur Installation enthalten,** welche für die Paketierung verwendet werden können! Zudem wird in der Regel erst durch eine solche Analyse ersichtlich, welche Abhängigkeiten von der Produkteinstallation vorausgesetzt und installiert werden. Siehe hierzu auch Kapitel *4.4 Abhängigkeiten (Middlewares)*.

Liegt die übergebene Source bereits in Form von *MSI*-Dateien vor, so kann mit Kapitel *6.3.3 Protokolldatei analysieren* weitergefahren werden. Ist hingegen nur eine *Setup.EXE* verfügbar, so sind die Arbeitsschritte der nächst folgenden Kapitel durchzuarbeiten.

6.3.1 Auspacken von Installationselementen aus einem Bootstrapper

Analysieren Sie jedes Setup! Oft werden *Windows Installer* Setups nicht als native *MSI*-Dateien angeboten, sondern sind in *Bootstrapper* eingepackt, welche die zum Betrieb der Applikation erforderlichen Abhängigkeiten installieren *(Redistributables)*, bevor dann die effektive *MSI*-Datei abgearbeitet wird.

Wenn sich die *MSI*-Datei nicht über Parameter auspacken lässt, so findet man die extrahierte Source nach dem Starten des Setups mit Benutzer-Interface und nach Erscheinen des ersten *Windows Installer* Dialogs lokal auf dem Client. Starten Sie das *Setup.EXE* hierzu auf einer *Clean Machine*, damit Sie auch erkennen können, welche Abhängigkeiten durch das Setup installiert werden. Siehe auch Kapitel *4.4 Abhängigkeiten (Middlewares)*. Der Initialisierungsteil des Client-Prozesses von *Windows Installer*, der während der Abarbeitung der Dialoge aktiv ist, kopiert die *MSI*-Datei als eine der ersten Operationen vom (extrahierten) Quellenverzeichnis normalerweise in den *%TEMP%*-Ordner. Lokalisieren Sie daher beim Erscheinen der ersten *Windows Installer* Dialogmaske auf dem Client die extrahierte *MSI*-Datei. Erweitern Sie Ihre Suche aber auch auf andere Orte der Festplatte. Es kann sein, dass sich extern vorliegende *Cabinett*-Daten und andere zur Installation gehörende Ressourcen in einem anderen Verzeichnis befinden. Wenn Sie die *MSI*-Datei, samt ihren zusätzlich benötigten Daten gefunden haben, kopieren Sie diese in ein sicheres Arbeitsverzeichnis. Die Dateien im *%TEMP%*-Verzeichnis werden nach Beendigung des Setups automatisch gelöscht! Achten Sie auch darauf, dass sie nur Dateien kopieren, die Sie zur Ausführung der *MSI*-Datei benötigen. Denn *Bootstrapper* erstellen in der selben Struktur oft auch andere temporäre Dateien, die für das *Customizing* nicht nötig sind *(InstMsi*.EXE*, etc.). Zudem werden während der Ausführung des Setups auch Daten aus der *Binary*-Tabelle der *MSI*-Datei ausgepackt, auf welche wir getrost auch verzichten können (*CustomActions*, Dateien der *InstallScript Engine*, etc.).

6.3.2 Ermitteln des Paketierungsumfangs

Wurden alle *MSI*-Dateien, die ausgepackt wurden, beiseite kopiert, dann kann mit *Orca* ermittelt werden, um was für *MSI*-Dateien es sich bei dieser Installation handelt. Zur Analyse können auch die im *%TEMP%*-Verzeichnis entstandenen Protokolldateien (bei eingeschalteter Protokollierung *Logging=voicewarmupx in HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer)* analysiert werden. Unterteilen Sie die Source in die zwei Kategorien…

1. Abhängigkeiten
2. Installation der Anwendung

…und identifizieren, bzw. trennen Sie lokale von globalen Abhängigkeiten (siehe Kapitel *4.4 Abhängigkeiten (Middlewares)*). Prüfen Sie zudem, welche der Abhängigkeiten im Umfeld des Unternehmens schon paketiert sind und notieren Sie deren Namen für die Implementation in der zur Zielanwendung gehörenden *INI*-Datei (*Dependence*-Eintrag).

6.3.3 Protokolldatei analysieren

Oft werden durch den *Bootstrapper* auch *Properties* übergeben, die sich in der temporären Protokolldatei *%TEMP%\MSI?????.LOG* wiederfinden (bei eingeschalteter Protokollierung *Logging=voicewarmupx*). Diese Datei ist auf die übergebenen *Properties* im Startbereich unter *Command Line:* zu prüfen. Die erweiterten *Properties* können später mittels der *INI*-Datei oder der

MST-Datei übergeben werden. In dieser Phase sind diese zumindest zu notieren. (Achtung: *CURRENTDIRECTORY, CLIENTUILEVEL, CLIENTPROCESSID, %HOMEPATH, %HOMEDRIVE* und *%HOMESHARE* können dabei ignoriert werden. Diese *Properties* sind in jeder Übergabezeile zu finden)

6.3.4 In-Place-Update von Patches (Splipstreaming)

Wird ein initiales Softwarepaket erstellt, also nicht ein Update *(Revisionsupdate)* eines bestehenden produktiven Pakets, und werden beim gelieferten Herstellersetup *MSP*-Patches mitgeliefert, bzw. lassen sich diese aus der *Setup.EXE* auspacken, so kann und sollte die Hersteller-*MSI*-Datei gleich mit allen erforderlichen *MSP*-Dateien chronologisch in Form von *In-Place-Updates* gepatcht werden (sofern möglich). Der Vorteil dieses Verfahrens ist der, dass die *MSI*-Installation und der Patch später in einer einzigen Installationstransaktion installiert werden können.

1. Zu diesem Zweck ist eine Administrativinstallation zu erstellen:

   ```
   msiexec /a <msipath\msiname> TARGETDIR=<AdminPoint> /L*v <logdatei> /qb-
   ```

Verwenden Sie unbedingt ein *TARGETDIR*, welches sich vom Verzeichnis von <msipath> unterscheidet, ansonsten wird die administrative Installation mit einer Fehlermeldung quittiert.

Achtung: Ist das *Attribut-Flag* in der *Components*-Tabelle irgend einer Komponente auf *Never-Overwrite* (128) gesetzt und sind diese so markierten Komponenten auf dem System registriert, wo die administrative Installation ausgeführt wird, so entpackt der Parameter */a* die damit verbundenen Dateien nicht. Bei späteren Zugriffen würde diese Datei fehlen und man könnte das Produkt auch nicht mehr fehlerfrei rekompilieren (erneutes Erstellen von *Cabinett*-Dateien).

2. Schliesslich wird das *In-Place-Update* auf die Administrativinstallation angewendet:

   ```
   msiexec /p <msppath\mspname>  /a <msipath\msiname>
   ```

Dieser Vorgang ist für jede *MSP*-Datei in der richtigen Reihenfolge durchzuführen. Die so entstandenen Ressourcen können dann als Basis für das Softwarepaket verwendet werden und sind zusätzlich im *Work*-Ordner des Softwarepakets zu sichern. Nach Anwendung eines *In-Place-Updates* ist bei der erstmaligen Installation der so entstandenen *MSI*-Datei die Anwendung darauf zu prüfen, ob diese den nötigen *Patch-Level* ausweist!

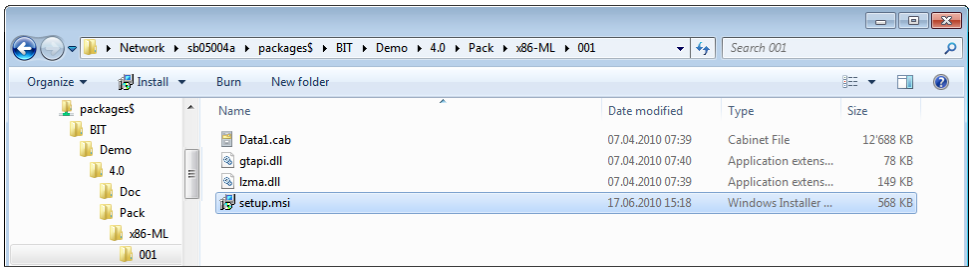
6.3.5 Verwendung von InstallShield-Setups

Auch viele *InstallShield*-Setups enthalten eine lauffähige *MSI*-Datei, die entpackt und anstelle des *Setup-Bootstrappers* verwendet werden kann. Beim Paketieren eines *InstallShield*-Setups ist dieses zwingend auf das Vorliegen einer eingebetteten *MSI*-Datei zu analysieren. Sollte eine *MSI*-Datei zu entpacken sein, so ist diese Datei anstelle des *Bootstrappers* zu verwenden. Die *MSI*-Datei kann mit entsprechender *MST*-Datei in das *Revisions*verzeichnis eingefügt werden.

Grundsätzlich ist auch hier das im Kapitel *6.3.1 Auspacken von Installationselementen aus einem Bootstrapper* und die in den Kapitel *6.3.2-6.3.3* beschriebenen Verfahren anzuwenden. Zusätzlich ist die *MSI*-Datei nach dem Auspacken auf das Vorkommen der *CustomActions ISVerify-ScriptingRuntime* und *OnCheckSilentInstall* zu prüfen. Wenn diese in der *MSI*-Datei zum Einsatz kommen, dann kann die Property *ISSETUPDRIVEN=1* in die *Property*tabelle der zu erstellenden *MST*-Datei eingefügt werden, um einen Installationsablauf ohne *Setup-Bootstrapper* zu ermöglichen. Die *CustomActions* würden dazu dienen, eine Fehlermeldung anzuzeigen, wenn die Installation ohne *Setup.EXE* ausgeführt wird.

6.3.6 Installationssource kopieren

Für jedes im Rahmen dieses Auftrages zu erstellende einzelne Softwarepaket (Zielanwendung inkl. Abhängigkeiten) ist die durch Auspacken entstandene Source in das *Revisionsverzeichnis* des Softwarepakets zu kopieren. Zum Erstellen der Ablagestruktur verwenden Sie *CreatePackage* (siehe Kapitel *6.2 CreatePackage*)



6.3.7 Erstellen einer MST-Datei für alle weiteren Customizing-Arbeiten

Änderungen an der jetzt eingepflegten *MSI*-Datei sind aus Gründen der Transparenz in eine *MST*-Datei zu implementieren. Zudem verfällt in vielen Fällen der Herstellersupport, wenn Anpassungen direkt in der *MSI*-Datei erfolgen würden. Weitere Details im Zusammenhang mit *MST*-Dateien ersehen Sie aus dem Kapitel *6.6 Grundregeln bei der Anwendung von MST Dateien*.

6.3.8 Spezielle Einstellungen über das Benutzerinterface des Herstellersetups

Müssen Eingabeoptionen über Setup-Dialogfelder konfiguriert werden, können entweder Funktionen des Authoringtools verwendet werden oder die Ermittlung basiert auf das Erkennen von *Property*veränderungen nach manueller Ausführung des Setups (bei eingeschalteter Protokollierung *Logging=voicewarmupx*. Alle Veränderungen der *Properties*, ausser diese, die über „*MsiHiddenProperties*" in der *Property*tabelle der *MSI*-Datei vermerkt sind, sollten so ermittelbar sein. Die so ermittelten *Property*veränderungen sind dann in die *MST*-Datei zu übertragen.

6.3.9 Wahl von Features mit INSTALLLEVEL

Für einfachere Wahl/Abwahl von *Features* verwenden Sie die *INSTALLLEVEL*-Property. https://msdn.microsoft.com/en-us/library/aa369536(v=vs.85).aspx)

Level-Einträge in der *Feature* Tabelle grösser der *INSTALLLEVEL-Property* in der *Property*-Tabelle deselektiert das entsprechende *Feature* bei der Installation.

6.3.10 ShortCuts

Löschen oder Verändern Sie die *Shortcuts* nach Vorgaben des Applikationsverantwortlichen und den Regeln, die im Kapitel *4.2 Startmenü und ShortCuts* dokumentiert sind. Die Veränderungen sind in der *MST*-Datei in der *ShortCut*-Tabelle anzuwenden. Für alle Veränderungsabsichten, die nicht direkt in der *MST*-Datei durchgeführt werden können, ist die Umsetzung über eine *PostInstall_00x.cmd* empfohlen.

6.3.11 Probleme mit der Silentinstallation

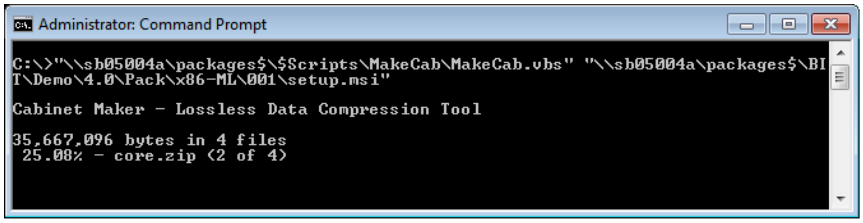
Sollten sich betriebsauswirkende Unterschiede im Installationsablauf ergeben, je nachdem, ob die Ausführung silent initiiert oder das Setup mit BenutzerInterface gestartet wurde, dann müssen sie sich Gedanken darüber machen, was denn nun mit der UI anders läuft als ohne:

Überprüfen Sie die *Conditions* in der *LaunchCondition*-, *Condition*-, *Component*- und *InstallExecuteSequence*-Tabelle nach Verweisen auf den *UILevel* oder auf *Properties*, die in der *ControlEvent*-Table gesetzt werden. Überprüfen Sie vor allem auch die *ControlEvent*-Tabelle nach "*DoAction*"-Ereignissen, welche auf *Controls* in Ihren ausgewählten Dialogen folgen. Überprüfen Sie so ermittelte *CustomActions* auf Existenz in der *InstallExecuteSequence* und fügen Sie diese falls erforderlich dort über die Transformdatei ein. Überlegen Sie sich gut, in welcher Sequenz sie diese einfügen. Wenn es sich um benutzerdefinierten Aktionen aus Dialogen vor der effektiven Installation handelt, so sind diese Aktionen am besten vor der *Scripterstellung*, vor *InstallInitialize* einzufügen. Andere *CustomActions* die erst über einen Abschlussdialog zur Ausführung kämen, wären nach *InstallFinalize* einzutragen.

6.3.12 Verwendung von MakeCab.vbs

Von externen MSI-Ressourcen, die im Rahmen einer Administrativinstallation entstanden sind oder in dieser Form vom Hersteller geliefert wurden, können mit dem Script *MakeCab.vbs Cabinett*-dateien erstellt werden, ohne dass die *MSI*-Datei des Herstellers mit einem Authoring Tool - unter der nachteiligen Erweiterung der *MSI*-Datei mit zusätzlichen Anpassungen um vieler Properties und Tabellen - kompiliert werden muss.

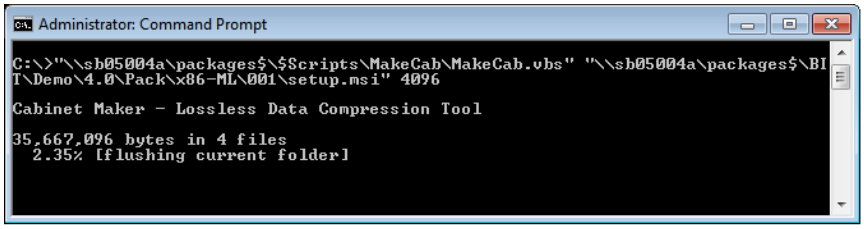
Zur Ausführung ist eine Kommandozeile mit dem Script *MakeCab.vbs* und der Verwendung des *MSI*-Pfades in einer DOS-Konsole abzusetzen:

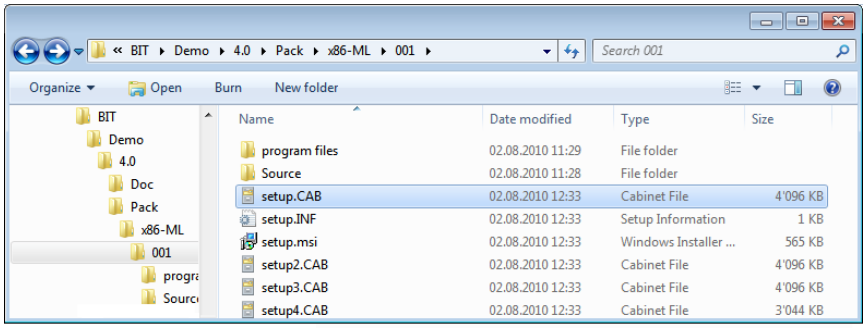


Das Script erstellt dann die *Cabinett*dateien im Wurzelverzeichnis der *MSI*-Datei. Die externen Source-Daten können nach der Operation ins Work-Verzeichnis verschoben werden.

Grösse der Cabinettdateien

Standardmässig werden durch das Script 100MB externe *Cabinett*-Dateien erstellt. Auf Wunsch kann aber die Grösse mittels einer zweiten Kommandozeilenoption angepasst werden. Hier wäre die Grösse der *Cabinett*-Datei in Kilobyte anzugeben.





Achtung: Die Grösse einer *Cabinett*datei ist auf 2GB beschränkt!

6.4 Paket repaketieren

Ist in den Installationsmedien keine eingebettete *MSI*-Datei vorhanden und ist durch den Hersteller auch keine *MSI*-Datei verfügbar, so wird ein Softwarepaket repaketiert. Als *Repaketierung* wird der Prozess bezeichnet, welcher verwendet wird, um bei einem *Legacy-Setup* oder bei Rohdaten per *Snapshot*-Verfahren ein *Windows Installer Setup* zu erhalten.

6.4.1 Regeln im Zusammenhang mit repaketierten Software-Paketen

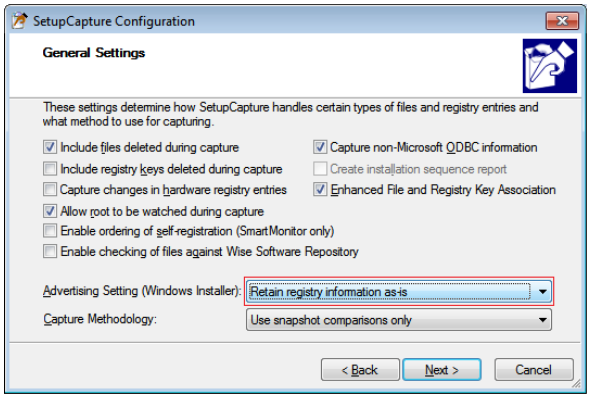
Bei der Repaketierung gelten alle im Kapitel *4 Best-Practice Regeln und Limitierungen* aufgeführten Regeln. Zusätzlich sind folgende Regeln einzuhalten:

6.4.1.1 Selbstregistrierung von Dateien und Verwendung von Advertising-Tabellen

Die Einbindung von *SelfReg*-Informationen in der *SelfReg*-Tabelle der *MSI*-Datei ist abzuschalten. *SelfReg*-Informationen sollen nur über die *Registry*-Tabelle implementiert werden. Auch alle anderen *Advertising*-Tabellen sollten nach dem Snapshot-Prozess leer sein: *AppID, ClassID, Extension, Mime, ProgID, SelfReg, TypeLib, Verb.*

Überprüfen Sie die Einstellungen in Ihrem Authoringtool, die diese Eigenschaften ermöglicht.

Hier ein Beispiel von *Wise Package Studio.* Dort verhindert das Abfüllen dieser Tabellen die folgende Option: *Advertising Setting:* ***Retain registry information as-is****:*



6.4.1.2 Keine virtualisierten Daten (VirtualStore)

Während des *Snapshot*-Prozesses dürften eigentlich keine virtualisierten Daten entstehen, da dieser als Administrator gestartet wird. Vorsichtig muss man erst bei einer nach der Anwendungsinstallation durchzuführenden Konfiguration sein. Alle Prozesse, die dann im Rahmen einer Konfiguration gestartet werden, beispielsweise auch der Start der Hauptapplikation, müssen als Administrator ausgeführt werden, um eine allfällige **Datei- und Registrierungsvirtualisierung zu verhindern!**

Zudem sind beim Anzeigen der Ressourcen nach dem *After*-Snapshot, diese auf Einträge im *VirtualStore* zu prüfen. Dort vorgefundene **Ressourcen im *VirtualStore* sind zwingend im Originalverzeichnis/Originalregistryhive zu implementieren!** Gegebenenfalls ist der *Repaketierungs*prozess neu zu starten.

6.4.1.3 Mergemodule

Der Einsatz von *Mergemodulen* wird beim *Repaketierungs*prozess in der Umgebung des Unternehmens **nicht empfohlen.**

6.4.1.4 Umgang mit Abhängigkeiten

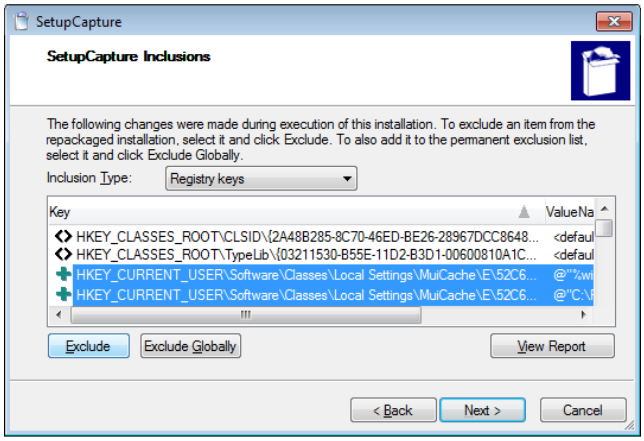
Auch bei einer *Repaketierung* sind Software-Abhängigkeiten aus dem *Legacy-Setup* zu identifizieren und zu isolieren. Erkennt der Softwarepaketierer eine Abhängigkeit, die mit dem *Setup* installiert wird, welche sich entweder separat verpacken lässt oder wovon vom Hersteller ein isoliertes Setup erhältlich ist, so ist dieses einzeln zu paketieren.

Zudem sind allfällige Abhängigkeiten **vor dem Snapshot** der fokussierten Hauptapplikations-installation auf das System zu installieren und die Abhängigkeit ist in der Softwarepaket-*INI*-Datei als „*Dependence*"-Erweiterung anzubringen (bei globalen Abhängigkeiten).

6.4.2 Computerneustarts bei der Repaketierung

Wenn ein technischer Computerneustart während des *Repaketierungs*prozesses für den Betrieb der Applikation notwendig ist, so führen Sie diesen nach Abschluss der Installation, aber vor einer allfälligen Softwarekonfiguration und vor allem vor dem Abschluss des Repaketierprozesses aus, sofern dies das Authoring Tool unterstützt.

6.4.3 Anschlussarbeiten

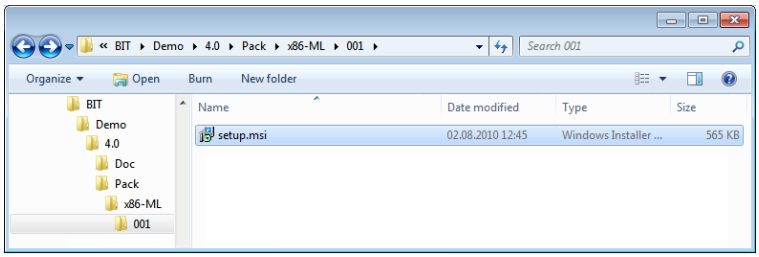


Die wichtigste Tätigkeit nach einem Snapshot ist die Bereinigung der *MSI*-Installation. Ziel ist es, nur diese Ressourcen im erstellten Softwarepaket zu implementieren, die direkt für die Anwendungsinstallation notwendig sind. Viele Prozesse, die der Paketierer oder das System während des Aufzeichnungsverfahrens startet, hinterlassen Spuren, die der Softwarepaketierer jetzt identifizieren und löschen muss.

Achten Sie in dieser Phase auch auf virtualisierte Ressourcen, die möglicherweise durch den *Snapshot* aufgezeichnet wurden (siehe Kapitel *6.4.1.2 Keine virtualisierten Daten (VirtualStore)*).

Kontrollieren Sie auch, ob Ihr Benutzername nativ in irgendwelchen Tabellen eingetragen wurde und löschen Sie diesen oder verwenden eine variable Benutzerbezeichnung.

Ist der Prozess der Anschlussarbeiten abgeschlossen, so können Sie die Rohdaten zu einer MSI-Datei kompilieren und in das Softwarepaketverzeichnis im *Work*-Ordner (am besten unter dem entsprechenden *Revisions*verzeichnis) ablegen, sofern dies noch nicht erfolgt ist. Zudem Kopieren Sie die beim Kompilierungsprozess erstellte *MSI*-Datei in das *Revisions*verzeichnis des Softwarepakets.



Alle weiteren *Customizing*-Aufgaben sind in der Regel in einer *MST*-Datei abzuspeichern, damit die Transparenz und Lesbarkeit gewährleistet ist.

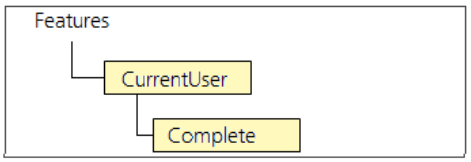
6.5 Umgang mit Benutzerressourcen

Zweifelsohne ist eines der heikelsten Anforderungen an das Softwarepaket die Umsetzung der Installation von Benutzerressourcen (*HKCU*, *%APPDATA%*, etc.). In Umgebungen, die den *Package-Launcher* einsetzen, können solche Benutzerressourcen neben der Implementation über die Reparatur des Basisproduktes via *Advertising* im Benutzerkontext, auch über

ActiveSetup_00x.cmd-Erweiterungen hinzugefügt werden. Dieses Verfahren stellt meist die einfachere Implementationsmöglichkeit dar.

Für das Design von *MST/MST*-Dateien, die Benutzerressourcen in Form einer *Windows Installer* Reparatur installieren sollen (bspl. über *advertised Shortcuts*), ist auf folgende Punkte zu achten:

- Die Benutzerressourcen sind in einem eigenen *Windows Installer Feature* isoliert abzubilden Dieses *Feature* ist als *Top-Level Feature* zu markieren. Als Name des *Features* eignet sich ein sprechender Name wie „*CurrentUser*"



- Der *KeyPath* der den Benutzerressourcen zugewiesenen Komponenten muss auf einen Registrykey in *HKCU* verweisen. Dies ist insbesondere auch bei Dateiressourcen so zu bewerkstelligen. Es ist zu gewährleisten, dass der Registrykey, der als *KeyPath* verwendet wird, nicht von der Anwendung oder durch Anwendungselemente selbst geschrieben werden kann (beispielsweise, wenn der Benutzer die *EXE*-Datei der Applikation in der Konsole oder über Start/Ausführen startet). Im Zweifelsfall ist ein entsprechender *Dummy-Key* durch den Softwarepaketierer in *HKCU* durch die *MST*-Anpassung selbst zu implementieren.
- Das Initiieren einer *Active Setup*-Benutzerinstallation ist durch das Script *ActiveSetup_00x.cmd* zu erledigen. Siehe Kapitel *3.8.4 Active Setup_00x*
- Achten Sie auf die Verfügbarkeit aller Benutzerressourcen, auf die das Softwarepaket zum Zeitpunkt der Benutzerinstallation zugreift. Applikationstests sind daher unbedingt auch im Kontext des Benutzers, ohne Administratorrechte und ohne Zugriff auf die primäre Installationssource durchzuführen. Mit dem Bezeichner *CopyLocal=True* in der *INI*-Datei des Softwarepakets kann angegeben werden, dass der *Package-Launcher* die Source des Softwarepakets vor der Erstinstallation lokal zwischenspeichert (sekundäre Installationssource) und die Installation von dort ausführt. Siehe Kapitel *6.10.2 Lokaler Cache*.
- Oftmals benötigen Applikationen zur Installationszeit keine Benutzerressourcen. Es wird daher vor allem bei *repaketierten* Anwendungen empfohlen, Benutzerressourcen beim ersten Installationsversuch des Pakets testweise nicht mit zu installieren.

6.6 Grundregeln bei der Anwendung von MST Dateien

Grundsätzlich sind alle *Customizing*-Arbeiten durch den Softwarepaketierer in der *MST*-Datei und nicht in der *MSI*-Datenbank zu vollziehen. Dies gilt auch bei repaketierten Softwarepaketen, nachdem die Bereinigungsarbeiten erledigt wurden. So wird die Transparenz erhöht.

Zur Erstellung und Pflege der *MST*-Dateien **wird Orca empfohlen**, da dieses Tool die Umsetzung der Bedürfnisse „sauberer" umsetzt. Viele Authoring Tool Editoren erweitern *MST*-Dateien mit einer Reihe von *Properties* und zusätzlichen Tabellen, so dass die Lesbarkeit für andere Personen und die Wartung darunter leidet.

Generell wendet der *Package-Launcher* bei der Installation **alle** im *Revisions*verzeichnis des Softwarepakets befindlichen *MST*-Dateien an. Ausnahme sind *regionenspezifische Varianten*.

Wenn zunächst keine *MST*-Datei vorliegend ist und man allgemeine Bedürfnisse umsetzen möchte wäre das Script *AddProperties.vbs* auszuführen. Dieses erstellt eine Standard-*MST*-Datei mit der Bezeichnung *STANDARD.MST*. Für alle weiteren Anpassungen könnte diese *MST*-Datei entsprechend erweitert werden.

Liegt hingegen durch den Hersteller bereits eine *MST*-Datei mit Anpassungen vor, so ist diese vorgängig auf *STANDARD.MST* umzubenennen, bevor weitere Ergänzungen durch den Softwarepaketierer hinzugefügt werden. Im Allgemeinen ist dies die bevorzugte Variante. Nur, wenn der Softwarepaketierer, die allgemeinen Anpassungen vom Hersteller gegenüber den eigenen Erweiterungen abgrenzen möchte, behält er die Hersteller-*MST*-Datei bei und fügt dem Softwarepaket mit *AddProperties.vbs* eine neue *STANDARD.MST* hinzu.

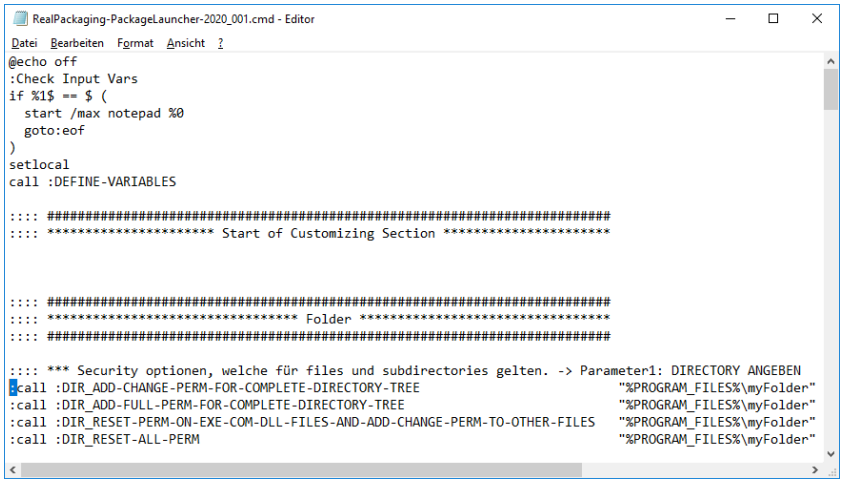
6.7 ACL Lockerungen

Im Einsatz des *Package-Launchers* werden im Softwarepaketierungsprozess *ACL*-Lockerungen am Dateisystem und der Registry mittels eines *Security-Batches* vorgenommen, der Bestandteil des *Package-Launchers* ist. Der *Security-Batch* ist in der unter Kapitel *3.7.3 Bezeichnung des Security-Batch* dokumentierten Namensgebung im entsprechenden *Revisionsverzeichnis* zu verwenden.

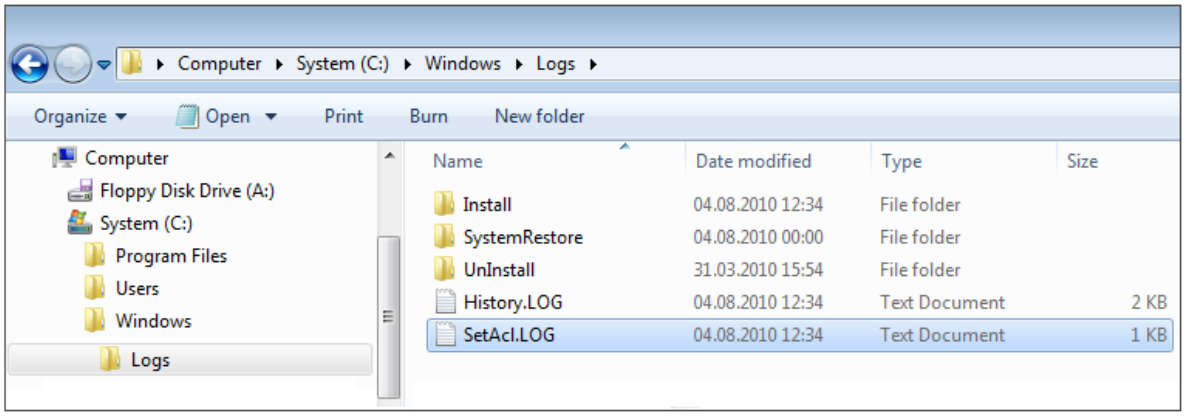
Das Programm *CreatePackage.EXE* (siehe Kapitel *6.2 CreatePackage*) kopiert eine entsprechende Vorlage des *Security-Batches* in das *Work*-Verzeichnis. Diese Vorlage kann nun in das *Revisions*verzeichnis kopiert und der Name der Datei mit dem *Revisions*bezeichner ergänzt werden.

Vergewissern Sie sich, dass der in der Datei verwendete *Revisions*bezeichner dem *Revisions*bezeichner des *Revisions*verzeichnisses genau übereinstimmt. Ansonsten wird der *Package-Launcher* diese Datei nicht anwenden!

Eine Anpassung implementiert der Softwarepaketierer schliesslich, indem in der *Customizing-Section* im oberen Teil der Vorlage der Doppelpunkt „:“ am Beginn des Batches entfernt und das zu öffnende Zielverzeichnis am Ende der Zeile anpasst wird *("%PROGRAM_FILES%\myFolder")*. Bei Anpassungsvorhaben, die auf die Registry zielen, ist im dritten Block der *Customizing-Section* nach genau gleichem Muster vorzugehen. Hier ist am Ende der Zeile der zu öffnende *Registryhive* anzugeben *("HKLM\SOFTWARE\MyApp\User")*



Achtung
Wird in einer Nachfolge*revision* eine Korrektur des ursprünglichen Batches notwendig, dann muss eine neue Variante des Batches in das neue *Revisions*verzeichnis kopiert werden, **die die alten Inhalte, sowie die Erweiterungen enthält**. Der *Package-Launcher* kopiert diese neue *CMD*-Datei bei der Installation der *Revision*, sofern die Datei neuer ist, als die lokal gespeicherte *CMD*-Datei. Lokal liegt immer nur eine *ACL-CMD*-Datei pro Softwarepaket vor. Die Anpassungen werden durch den *Package-Launcher* im Verzeichnis *%WINDIR%\Logs* in die Datei *SetAcl.LOG* geschrieben.



6.7.1 Datei und Registrierungsvirtualisierung

Durch Programme im Kontext des Benutzers durchgeführte Schreiboperationen, die in die Verzeichnisse *%PROGRAMFILES%*, *%WINDIR%*, oder *%WINDIR%\System32,* sowie in die Registry unter *HKLM\Software* zielen, werden unter *Windows Vista / 7 / W10* nach *%LOCAL-APPDATA%\VirtualStore*, bzw. *HKCU\Software\Classes\VirtualStore\MACHINE\SOFTWARE* umgeleitet. Diese Umleitung birgt für die Softwarepaketierung gewisse Risiken, vor allem im Hinblick mit dem Upgrade von Ressourcen, die auf Clients virtualisiert vorliegend sind. Weitere Informationen findet man unter *folgendem Link*.

Zur Reduzierung dieser Risiken werden folgende Massnahmen empfohlen, die der Softwarepaketierer umsetzen soll:

1. Die Berechtigungen von Verzeichnissen, Dateien und Registrierungsschlüssel sind **gem. Auftrag zu öffnen,** sofern die *ACL*-Lockerungen zum erfolgreichen Betrieb der Software nötig sind.

2. Eine *ACL*-Lockerung ist unaufgefordert für alle Ressourcen aus oben beschriebenen Verzeichnissen/Registrierungsbereichen vorzunehmen, die auf dem Computer **zentral nur einmal abgelegt** sein dürfen (bspl. zentrale Datenbankdatei, die in einer Mehrbenutzerumgebung von verschiedenen Benutzern gepflegt würde).

3. Eine *ACL*-Lockerung ist unaufgefordert für alle Ressourcen aus oben beschriebenen Verzeichnissen/Registrierungsbereichen vorzunehmen, die zum Installationsumfang gehören und die durch den Softwarepaketierer als **betriebsbeeinflussende und veränderliche Konfigurationsdateien interpretiert** werden können (bspl. *INI*-Dateien), unabhängig davon, ob die Datei bei einem Schnelltest der Software durch den Softwarepaketierer virtualisiert wird oder nicht.

6.8 PreInstall-Pakete

Softwarepakete **ohne *MSI*-Ressourcen** können mit dem *Package-Launcher* mittels einer *PreInstall*-Datei abgebildet werden. Beachten Sie, dass auf die Ausführung einer *PreInstall*-Datei keine *PostInstall-Datei* folgen muss.

6.8.1 Umgang mit Legacy-Setups, die nicht repaketiert werden

Legacy-Setups werden über ein *PreInstall*-Paket initiiert. Das zu verwendende *Setup.EXE* muss im *Revisions*verzeichnis abgelegt werden und in der *INI*-Datei des Softwarepakets ist der Bezeichner ***CopyLocal*** **auf** ***True*** zu stellen, wenn für die DeInstallation der Zugriff auf das *Legacy-Setup* erforderlich ist.

6.9 Umgang mit Patchdateien

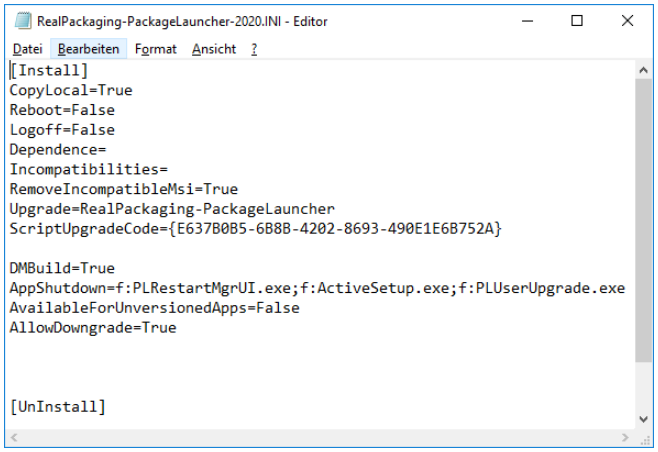
Ein erstes Kapitel über die Integration von Patchdateien in einer *initialen Revision* war bereits einmal Thema in diesem Dokument (siehe Kapitel *6.3.4 In-Place-Update von Patches (Splipstreaming)*). Müssen weitere Patches integriert werden, **nachdem** ein Softwarepaket in die Produktion überführt wurde, kann dies mittels einem *Revisionsupdate* erfolgen. Hier sind die *MSP*-Patchdateien einfach in das *Revisions*verzeichnis zu kopieren. Es können dabei mehrere Patchdateien in das selbe *Revisions*verzeichnis eingefügt werden.

Weitere Aktionen sind nicht erforderlich. Beachten Sie auch die Einschränkungen, die im Betriebshandbuch *Setup-Launcher* im Kapitel *3.11 Anwenden von Patches und Transformationen* dokumentiert sind.

6.10 INI-Datei des Softwarepakets

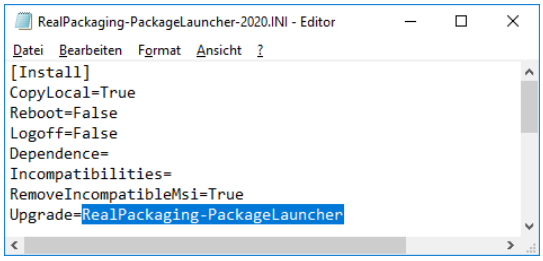
In der *INI*-Datei des Softwarepakets werden Einstellungen definiert, die für die Installation des Softwarepakets relevant sind. Die *INI*-Datei wird im Wurzelverzeichnis des Softwarepakets abgebildet und trägt den Namen des Softwarepakets. Bspl. *Byron-HIP-1.0.INI.*

Das Kapitel *3.5 INI-Datei* widmet sich mit der Ausgestaltung dieser *INI*-Datei. An dieser Stelle wollen wir nur kurz auf einige wenige Bezeichner eingehen.



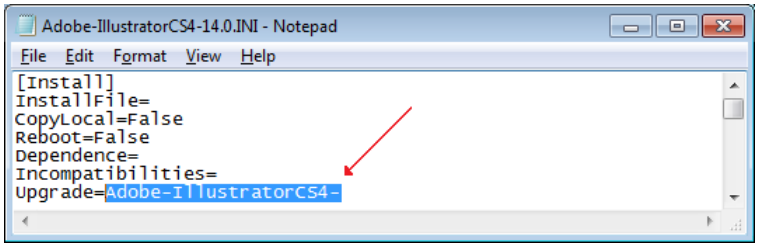
6.10.1 Upgrade-Handling

Über den Bezeichner *Upgrade=xxxx* bestimmt der Softwarepaketierer, welche Softwarepakete der *Package-Launcher* in Form eines *Package-Upgrades* vor einer Installation entfernen soll. Dies ist die favorisierte Variante, Softwareprodukte im Rahmen eines *Upgrades* zu entfernen. Standardmässig erstellt *CreatePackage.EXE* einen Paketbezeichner ohne Version, so dass die Deinstallation aller Softwarepakete aus der gleichen Produktfamilie im Rahmen eines *Package-Upgrades* erfolgt.



6.10.1.1 Eindeutiger Upgradebezeichner

Ist der Name eines Softwarepakets im Namen eines anderen Softwarepakets enthalten, beispielsweise *Adobe-IllustratorCS4* in *Adobe-IllustratorCS4LanguagePack*, so ist in der *INI*-Datei des kürzeren (Paketname) Softwarepakets beim *Upgrade*bezeichner ein Bindestrich anzufügen „-“, wenn dieses Softwarepaket nur Vorversionen seiner eigenen Produktfamile entfernen soll.

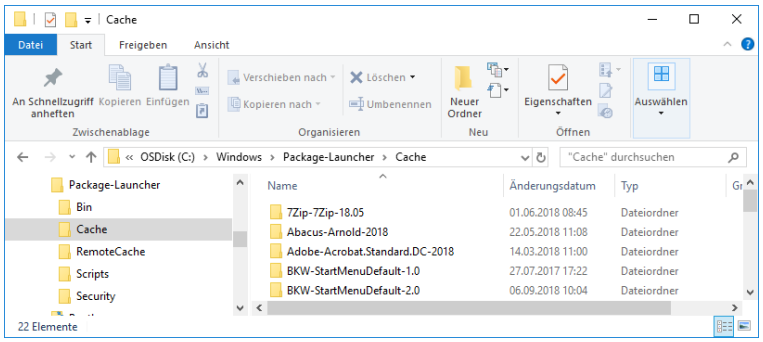


6.10.2 Lokaler Cache

Über eine Einstellung in der *INI*-Datei lässt sich vorgeben, den *Package-Launcher* anzuweisen, das Softwarepaket vor der Installation lokal zwischenzuspeichern (*CopyLocal=True*). Dies ist insbesondere für Softwarepakete notwendig, die Dateien oder andere Ressourcen während einer Benutzerreparatur benötigen (siehe Kapitel *6.5 Umgang mit Benutzerressourcen*). Auch bei *Legacy-Setups*, deren Ausführung über ein *PreInstall*-Paket mittels der Standardvorlage vorgesehen ist, ist eine lokale Zwischenspeicherung erforderlich (siehe Kapitel *6.8.1 Umgang mit Legacy-Setups, die nicht repaketiert werden*). Überdies ist bei MSU-Paketen, die wieder deinstalliert werden sollen, der Bezeichner *CopyLocal=True* zwingend erforderlich.

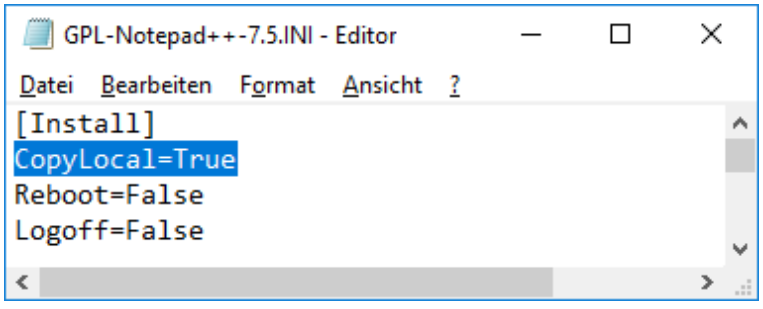
Der lokale Cache wird bei einer kompletten Deinstallation des Softwarepakets mit all seinen *Revisionen* durch den *Package-Launcher* wieder gelöscht. Zu beachten ist zudem, dass nur das komplette Softwarepaket (mit all seinen *Revisionen*) zwischengespeichert werden kann.

Das *Cache*-Verzeichnis befindet sich auf *%WINDIR%\Package-Launcher\Cache*:



In *PreInstall_00x.cmd* und in allen Scriptausprägungen steht für den Zugriff auf dieses Verzeichnis eine Variable mit dem Namen „*CACHE*" zur Verfügung.

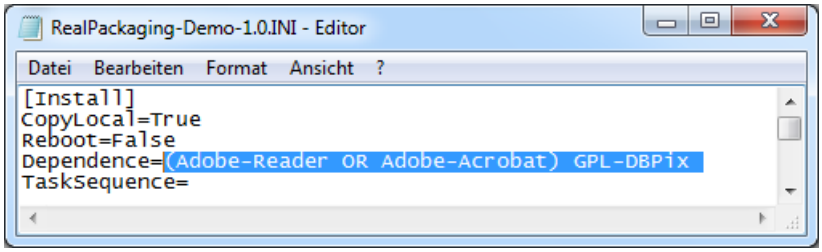
Der Bezeichner zur Anweisung der Zwischenspeicherung in der *INI*-Datei heisst ***CopyLocal*** und ist beim Caching auf ***True*** zu setzen:



6.10.3 Umgang mit Abhängigkeiten durch Verwendung des „Dependence"-Eintrages

Mit dem *Dependence*-Eintrag überprüft der *Package-Launcher* den Computer vor der Basisinstallation auf installierte Abhängigkeiten. Der *Dependence*-Eintrag ist dabei mit den Paketnamen zu ergänzen, welche als Abhängigkeit dieser Applikation fungieren. Der Paketname kann auch nur ein Teilstring der Applikation enthalten. In der Regel verwendet man den Namen

der Paketfamilie, also *Hersteller-Name*, ohne Versionsangabe, damit die Prüfung nicht auf die Version fixiert und auch später gültig ist.



Die verschiedenen Abhängigkeiten sind hier mit einem Leerzeichen aneinanderzufügen. Die Abhängigkeiten können mit dem ‚OR' Operator verknüpft werden (siehe Kapitel *3.5 INI-Datei*). Wenn der ‚OR'-Operator verwendet wird, muss der linke und rechte Bezeichner mit einer Klammer einfasst werden. Bspl. **(**Adobe-Reader **OR** Adobe-Acrobat**)**

Der *Dependence*-Eintrag wird auch von *SCCMCreateApp* während der automatischen Erstellung der *SCCM*-Objekte ausgelesen. Durch diesen Bezeichner werden alle erforderlichen Paketabhängigkeiten in der *(RTM)-Application* platziert.

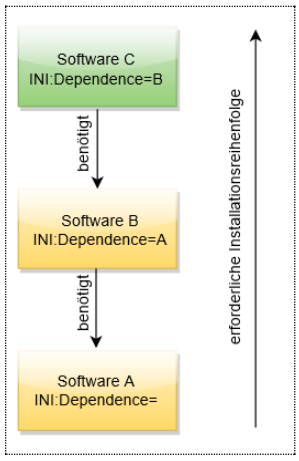
Merke

Verwenden Sie mit *SCCM CB* keine Abhängigkeiten in einer flachen Hierarchie! Wird beispielsweise für eine Software C eine Software B benötigt und für den einwandfreien Betrieb der Middleware B wäre A erforderlich, so wäre die Deklaration *Dependence=A B C* in der Software C das falsche Vorgehen! Verwenden Sie stattdessen ***hierarchische Abhängigkeiten*** im Dependence-Eintrag!

Software C: Dependence=B
Software B: Dependence=A

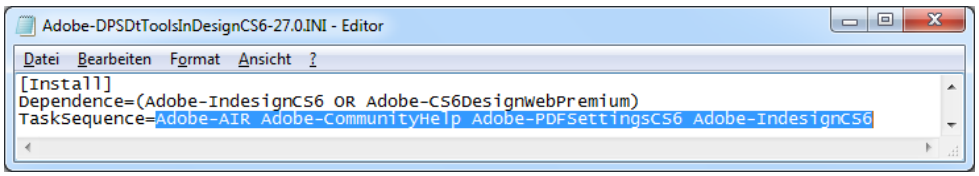
Der Grund hierfür liegt in der Tatsache, dass mit *SCCM CB* und dem *Application model* nur so eine korrekte Reihenfolge über Applicationgrenzen hinweg gesteuert werden kann. Würden die Abhängigkeiten in einer flachen Hierarchie angewendet, wäre die Reihenfolge nicht vorhersehbar.

Korrekte Dependence-Umsetzung:



6.10.4 Die Verwendung des „TaskSequence“-Eintrages

Als **Ergänzung** zum *Dependence*-Eintrag steht **optional** der *TaskSequence*-Eintrag zur Verfügung. Dieser Bezeichner ermöglicht **insbesondere bei hierarchischen Abhängigkeiten** (siehe oben), diese auch für die Installation mit dem *Remote Package Installer* (*RPI*) vorzubereiten. Der *RPI* verlangt im Gegensatz zu *SCCM CB* eine flache Hierarchie der Abhängigkeitsimplementation. Dort müssen Sie also Abhängigkeiten zusätzlich chronologisch aneinandergereiht über den *TaskSequence*-Eintrag einfügen.



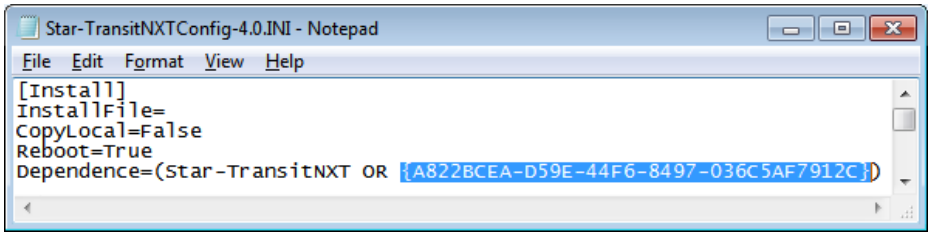
6.10.5 Berücksichtigung von Nicht Package-Launcher konformen Paketen

Für die Verwaltung *Nicht-Package-Launcher-konformer* Pakete sind im Zusammenhang mit der *INI*-Datei vier verschiedene Bezeichner von Wichtigkeit. Dies sind *Dependence*, *Upgrade*, *RemoveIncompatibleMsi* und *RemoveIncompatibleMsiUPG*. Auf die Implementation der nachfolgenden Ausführungen können Sie verzichten, wenn in Ihrer **produktiven** Umgebung nur *Package-Launcher-konforme* Softwarepakete eingesetzt werden.

Dependence

Beim Erweitern des *Dependence*-Bezeichners kann dieser auch mit den Paketen erweitert werden, die als *Nicht-Package-Launcher-konforme* Paketausprägung vorliegen und die installiert sein könnten. Und zwar nach der Form *Dependence=(neu OR incompatible) (neu2 OR incompatible2) …*

Für *Nicht-Package-Launcher-konforme* Paketausprägungen sind nur Einträge mit *{ProductCodes}* zu verwenden. Anbei sehen Sie ein Beispiel:

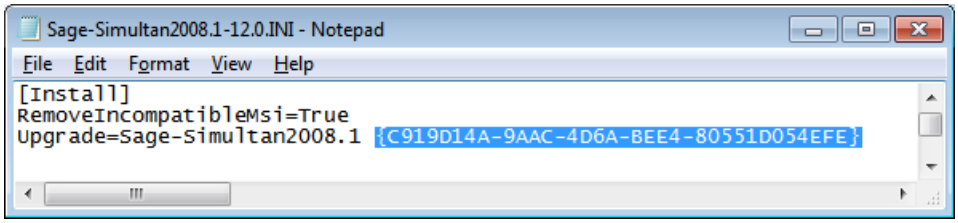


Achtung
Die automatische *SCCM*-Objekterstellung beim Überführen der Pakete in *SCCM* unterstützt keine *Nicht-Package-Launcher-konformen* Softwarepakete! Dies bedeutet, dass alle Einträge in der Form von *{ProductCodes}* beim Überführungsprozess ausgefiltert werden und in der Abhängigkeitsimplementation unberücksichtigt bleiben.

Upgrade

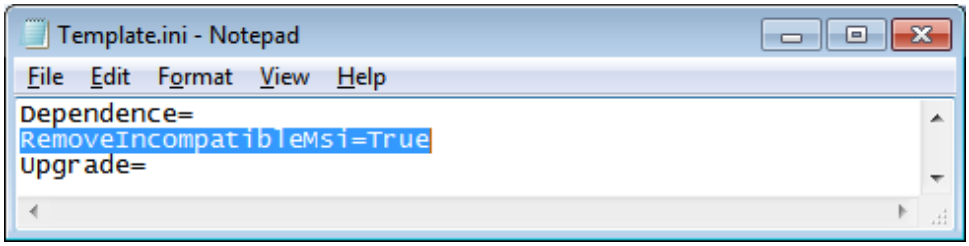
Sind mit einem *Upgrade* auch produktive, *Nicht-Package-Launcher-konforme* Softwarepakete zu aktualisieren, so ist nach einem ähnlichen Muster vorzugehen. Hier erweitert der

Softwarepaketierer den *Upgrade*bezeichner mit allfälligen *{ProductCodes}* der entsprechenden produktiven Versionen:



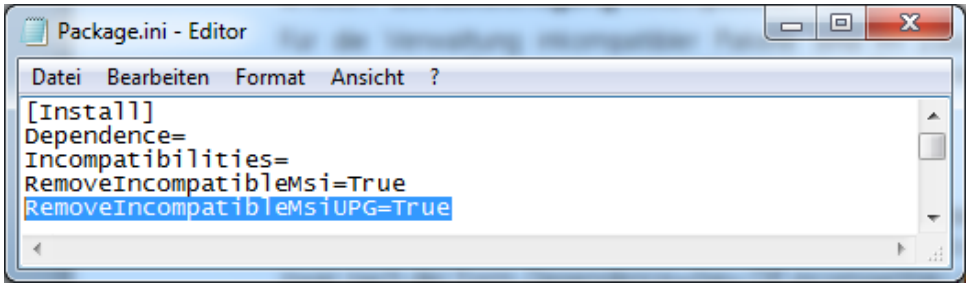
RemoveIncompatibleMsi

RemoveIncompatibleMsi sollte immer gesetzt werden, wenn es **das gleiche** Softwarepaket auch in einer nicht *Package-Launcher*-kompatiblen Version gibt. Dieser Bezeichner sorgt in diesem Fall dafür, dass die Verteilung des neuen Pakets auf Grundlage des *Package-Launchers* auf einem Client, wo noch eine inkompatible Version installiert ist, vorgängig die inkompatible Version entfernt. Der Bezeichner ist auf *RemoveIncompatibleMsi=True* zu stellen:



RemoveIncompatibleMsiUPG

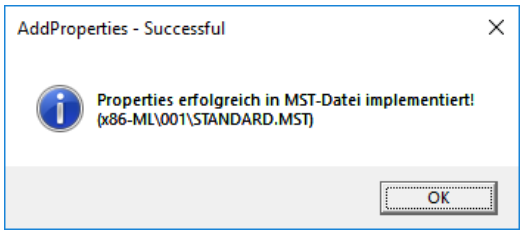
Mit dem Bezeichner *RemoveIncompatibleMsiUPG* kann man per *Windows Installer* installierte Produkte automatisch im Rahmen eines *Package-Upgrades* deinstallieren lassen, wenn diese den selben *UpgradeCode* im Paket verwenden. Gerade in Umgebungen, wo mehrere Paketarten (*Package-Launcher* Paket und andere Formen der gleichen Pakete) zum Einsatz kommen könnten, empfiehlt sich dieser Bezeichner.



6.11 AddProperties

AddProperties.vbs ist ein Script, welches bei ***MSI*-Ressourcen** alle für das Unternehmen erforderlichen *Windows Installer Property*anpassungen vornimmt und Erweiterungen in den anderen *Windows Installer* Datenbanktabellen anfügt, die für den robusten Betrieb mit dem

Package-Launcher notwendig sind. Das Script kann mehrfach per Doppelklick (befindet sich im Stammverzeichnis des Softwarepakets) ausgeführt werden. Die Anpassungen werden dann automatisch in eine *MST*-Datei geschrieben.



Merke
Verwenden Sie in Ihrer Zielrevision keine *MSI*-Datei, sondern nur andere Ressourcen, dann können Sie auf die Ausführung von *AddProperties_Link.vbs* verzichten.

6.11.1 Propertyanpassungen und zusätzliche Erweiterungen

Folgende Werte werden durch *AddProperties.vbs* standardmässig verändert und sind in jedem *MSI/MST* erforderlich:

Property	Inhalt Beispiel	Kurzbeschreibung
ALLUSERS	1	*Per-machine* Installation erzwingen
ROOTDRIVE	C:\	Systemlaufwerk. Kann Probleme geben, wenn nicht angegeben
ARPNOREPAIR	1	Reparatur in *Add/Remove* für Benutzer abschalten
ARPNOREMOVE	1	Remove in *Add/Remove* für Benutzer abschalten
ARPNOMODIFY	1	*Maintenance* in *Add/Remove* für Benutzer abschalten
REBOOT	ReallySuppress	Computerneustart verhindern
PLPackageEngineer	Dominik Oberlin	Informations-Property
PLPackage	Adobe-Reader-9.3	Paketnamen, erforderlich für *Package-Launcher*
PLRevision	001	*Revision*, erforderlich für *Package-Launcher*
(ARPSYSTEMCOMPONENT)	1	Optional, bei allen *Revisionen* > 001, damit diese zusätzliche *Revision* nicht in *Add/Remove* erscheint
Error		
0	{{Fatal error: }}	Erforderlich für *Launcher* Log-Einlesen
1	{{Error [1]. }}	Erforderlich für *Launcher* Log-Einlesen
1708	Installation operation failed.	Erforderlich für *Launcher* Log-Einlesen

CustomAction		
CA_CleanUpByRemove	70|PL_Binary|CleanUp	Erforderlich für *Package-Launcher Upgrade*
InstallExecuteSequence		
RemoveExistingProducts	NEVER	*Windows Installer Upgrade* verhindern
CA_CleanUpByRemove	UPGRADINGPRODUCTCODE	Erforderlich für *Package-Launcher Upgrade*
Binary		
PL_Binary	[Binary Data]	Vb-Script *Custom Action* für Upgrade

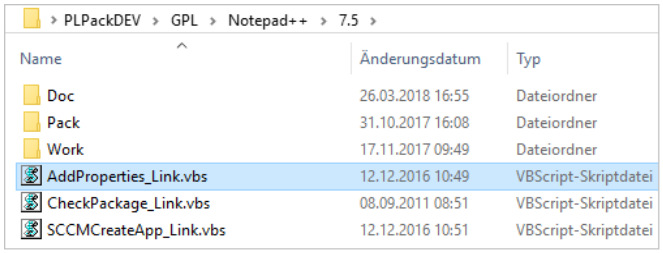
Zusätzlich zu diesen Angaben bettet *AddProperties.vbs* ein allfällig vorliegendes *PostInstall_00x.EXE* als externe *Custom Action* ein und implementiert falls nötig, wenn über die *INI*-Datei des Softwarepaketes angegeben, einen *Windows Installer Major-Upgrade* der auf die Vorversionen zielt (nicht empfohlen).

Standardmässig übt *AddProperties.vbs* die Anpassungen auf die **letzte *Revision*** aus. Wenn mehrere Architektur-Sprachfolder zum Einsatz kommen, werden alle letzten *Revisionen* fokussiert (aus allen Sprach- und Plattformverzeichnissen). Das heisst, dass dann auch mehrere Abschlussmeldungen angezeigt werden. Sollte man *AddProperties.vbs* auf eine andere Revision, als die letzte anwenden wollen, so kann die *MSI*-Datei aus der entsprechenden *Revision* als Kommandozeilenparameter übergeben werden. Zusätzlich kann optional als zweite Kommandozeilenoption auch noch die *Transformation* selektiert werden, auf die die Erweiterungsabsichten zielen.

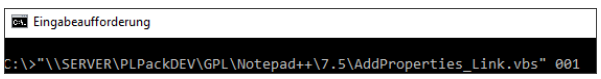
Aus dem folgenden Bild sind die Kommandozeilenoptionen ersichtlich. Auf den vollständigen Pfadnamen wurde hier aufgrund der Darstellungsmöglichkeit verzichtet.

6.11.2 Vereinfachte Ausführung

Prinzipiell ist für jede integrierte *Revision AddProperties_Link.vbs* auszuführen. Existiert nur eine *Revision 001* oder zielen die Anpassungsabsichten auf die letzte *Revision*, so reicht ein Doppelklick auf das Script:



Soll die Ausführung von *AddProperties* auf eine bestimmte *Revision* beschränkt werden, kann das mit dem Argument der *Revision* bewerkstelligt werden:



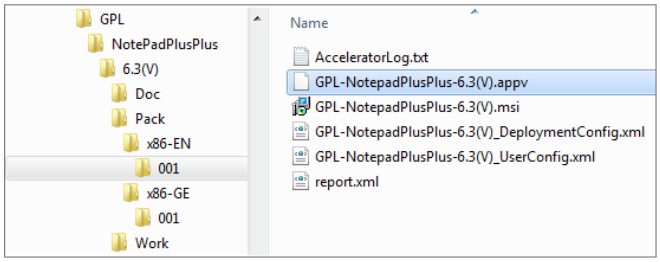
6.12 App-V Pakete

Der *Package-Launcher* unterstützt App-V Pakete. Solche Pakete können auf zwei unterschiedliche Arten in die Softwareverteilinfrastruktur überführt werden: *SCCMCreateApp* integriert AppV Pakete entweder in Form einer *App-V Full Integration* oder über das *standalone model*. Die *App-V Full Integration* bedeutet, dass in SCCM die Objekte mit dem Objekt *AppV5xInstaller* unter Verwendung eines Streamings integriert werden.

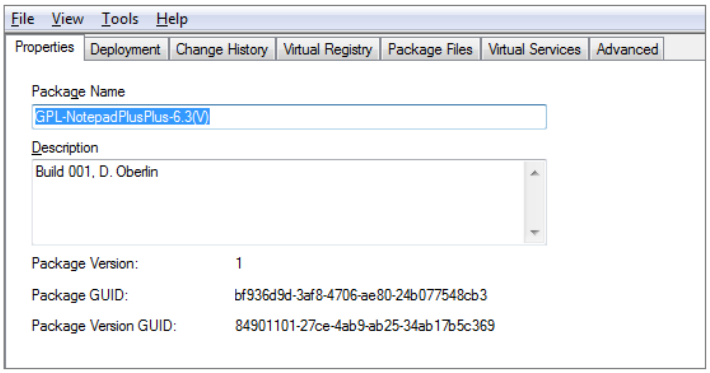
Demgegenüber steht die Integration im *standalone model*, welche das Paket *standalone* per *System* integriert und installiert. Dies ist die prädestinierte Variante in Umgebungen, wo Software und Einstellungen nicht dem Benutzer folgen müssen (bspl. FAT). Diese Art der Installation ermöglicht auch, dass die Paketinstallationen in der Datei *History.LOG* ausgewiesen werden, so wie dies auch herkömmlichen Paketen bekannt ist.

6.12.1 Namensrichtlinien und Verzeichnisstruktur bei App-V Paketen

Die mit *CreatePackage* erstellte und an die Namensrichtlinien des *Package-Launchers 2020* angelehnte Ablagestruktur sieht folgendermassen aus:



Die App-V-Ressourcen werden hierbei in das Revisionsverzeichnis 001 kopiert. Technisch spielen die Namen der ins Verzeichnis eingefügten App-V-Dateien keine Rolle. Der *Package-Launcher 2020* erkennt automatisch, ob es sich um App-V Inhalte handelt - egal, was für Dateinamen verwendet wurden. Hingegen muss der Name im App-V-Paket (siehe Bild unten: *Package Name*) genau dem Namen entsprechen, der mit *CreatePackage* angegeben wurde und der dem Paketnamen entspricht:



Über diesen Namen steuert der *Package-Launcher* seine Installations- und Deinstallationstransaktionen.

6.12.2 Scriptimplementationen

Für die Scriptimplementation stehen zwei Dateien zur Verfügung: die *DeploymentConfig.xml* und *UserConfig.xml*. In der Datei *UserConfig.xml* können Scripts ausgeführt werden, die im Kontext des Benutzers laufen. Hingegen können in die Datei *DeploymentConfig.xml* Scripts implementiert werden, die im Benutzerkontext und Systemkontext ausgeführt werden. Folgende Eintrittspunkte gibt es:

Script Execution Time	Can be specified in Deployment Configuration	Can be specified in User Configuration	Can run in the Virtual Environment of the package	Can be run in the context of a specific application	Runs in system/user context: (Deployment Configuration, User Configuration)
AddPackage	X				(SYSTEM, N/A)
PublishPackage	X	X			(SYSTEM, User)
UnpublishPackage	X	X			(SYSTEM, User)
RemovePackage	X				(SYSTEM, N/A)
StartProcess	X	X	X	X	(User, User)
ExitProcess	X	X		X	(User, User)
StartVirtualEnvironment	X	X	X		(User, User)
TerminateVirtualEnvironment	X	X			(User, User)

UserConfig.xml

Diese Datei ist für Benutzereinstellungen vorgesehen, die in einem *User-Targeting* Umfeld angewendet werden können.

DeploymentConfig.xml

Die Einstellungen aus dieser Datei werden systemweit für alle Benutzer auf dem Client appliziert, wenn das Paket verteilt wird. Dies können System- und Benutzereinstellungen sein.

6.12.3 Updates und Upgrades

Im Rahmen von App-V Paketen können bestehende Pakete aktualisiert werden und unter Beibehaltung der *PackageGUID* abgespeichert werden.

Eine in dieser Form als *In-Place-Update* realisierte Aktualisierung macht insbesondere Sinn bei Anpassungen und Erweiterungen, Einstellungsveränderungen & Patches - oder Updates, die vom Lieferanten als Update geliefert werden.

Vorteil
Benutzereinstellungen und generelle Einstellungen, die nicht im Benutzerprofil abgelegt sind und sich im virtuellen Paket der alten Version befinden, gehen mit dem Update nicht verloren.

Auch ein als *Update* gespeichertes Paket muss aber – im Gegensatz zu der klassischen *Package-Launcher* Paketierung – in ein neues Paket mit einer neuen Version in dessen *Revisions*folder 001 abgelegt werden!

Planen Sie hingegen ein *Upgrade* eines Pakets, sequenzieren Sie einfach von Beginn weg mit der neuen Installation und speichern das Paket unter der Berücksichtigung der Namenskonvention in dem neu erstellten *Revisions*verzeichnis '001'.

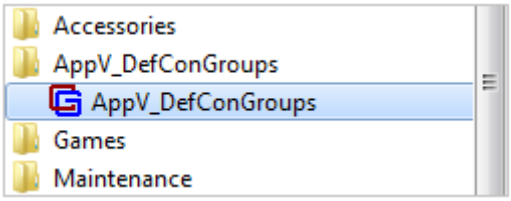
6.12.4 Connection Groups

Standardmässig sollen alle von einer Software benötigten Abhängigkeiten in das virtuelle Softwarepaket mitsequenziert werden, die nicht Bestandteil der Basisinstallation sind. Für Pakete, wo Software auf eine komplexe, schwierig zu erstellende Basis zurückgreift und diese Basis mehrfach verwendet wird, können im Ausnahmefall einzelne Softwareelemente isoliert sequenziert werden, um diese im Anschluss zu verbinden. Es ist möglich, solche Verbundpakete dann in einer einzigen und gemeinsamen virtuellen Instanz laufen zu lassen. Das Verfahren nennt sich in App-V 5 *Connection Groups*.

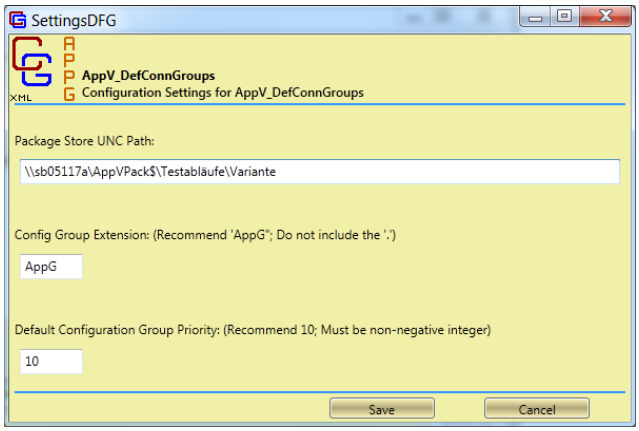
Connection Groups werden in *SCCM CB* mit *Virtual Environments* verbunden und angewendet. Für eine Standaloneinstallation lässt sich eine korrespondierende XML-Datei mit dem Tool *AppV_DefConGroups* erstellen. Die Source dazu befindet sich hier

6.12.5 Vorgehen zum Erstellen einer lokalen Connection Group

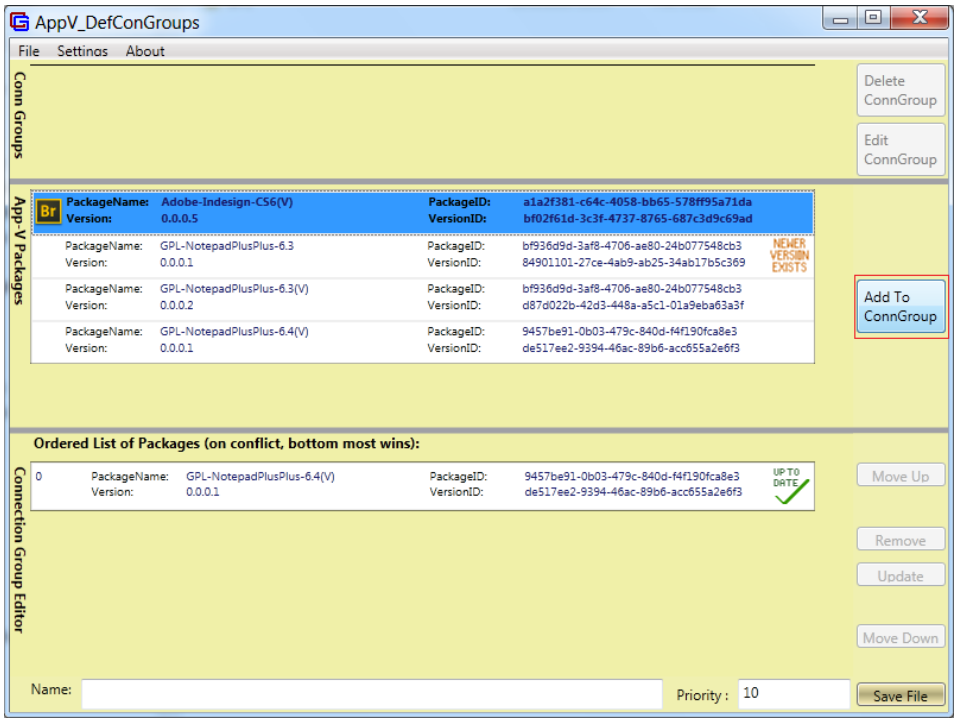
1. Starten Sie DefConGroups



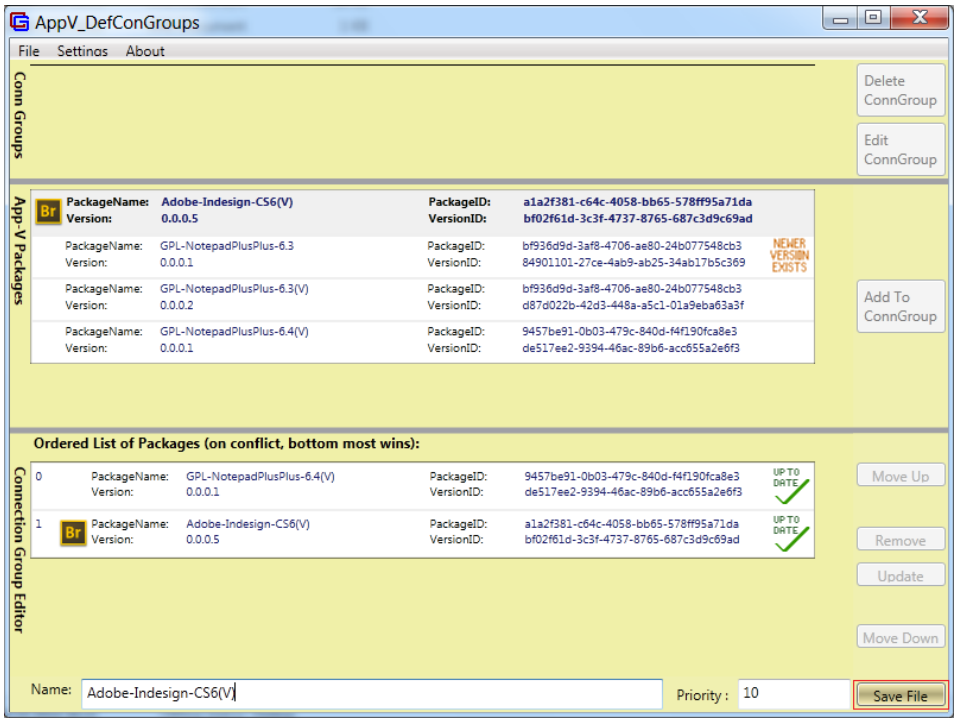
2. Vergewissern Sie sich, dass in den Einstellungen der *Store UNC Path* auf eine Ablage zeigt, die alle in der *Connection Group* zusammenzufassenden Pakete enthält:



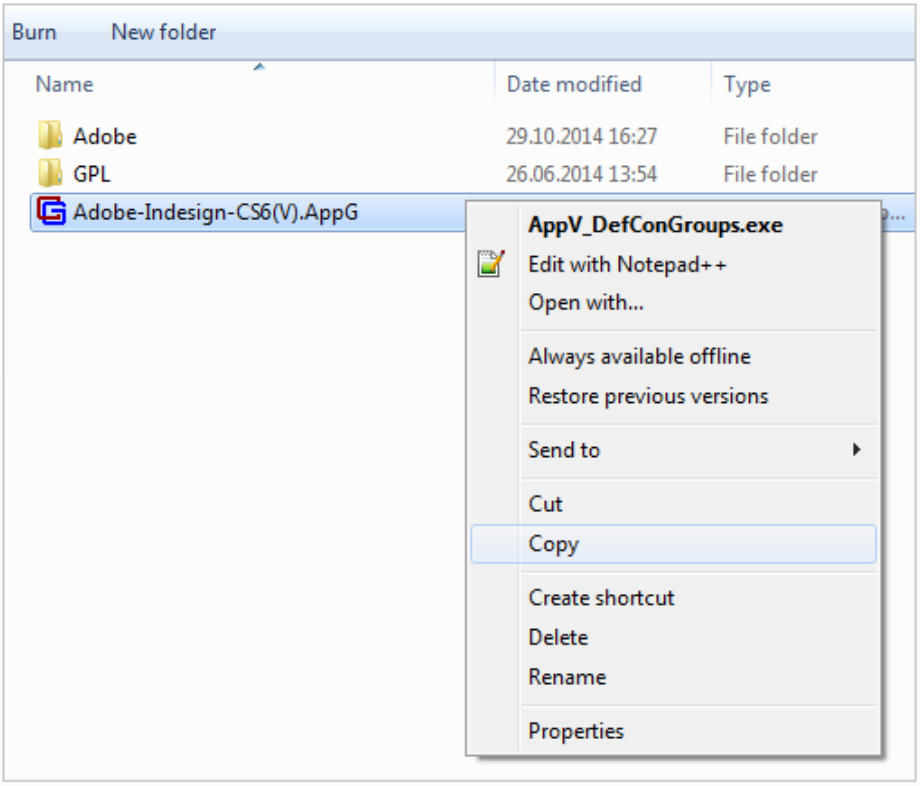
3. Wählen Sie die erforderlichen Pakete aus und klicken auf *Add To ConnGroup*
   Die hier dargestellten Pakete dienen nur zur Veranschaulichung des Prozesses. Ein Verbund dieser Pakete würde wenig Sinn machen.



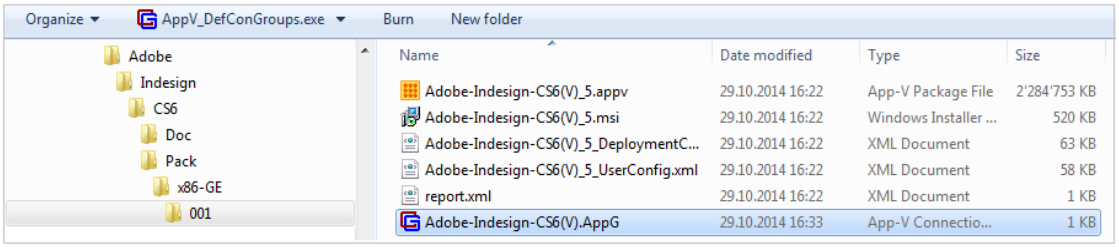
4. Speichern Sie die *Connection Group* mit dem Namen der Hauptanwendung (nicht mit dem Namen der Abhängigkeit)



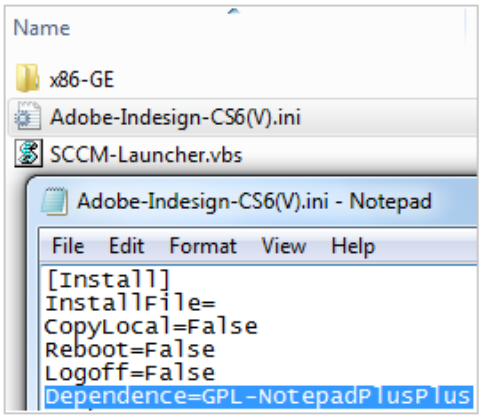
5. Kopieren Sie die Datei…



6. …und legen diese im Softwarepaket ab

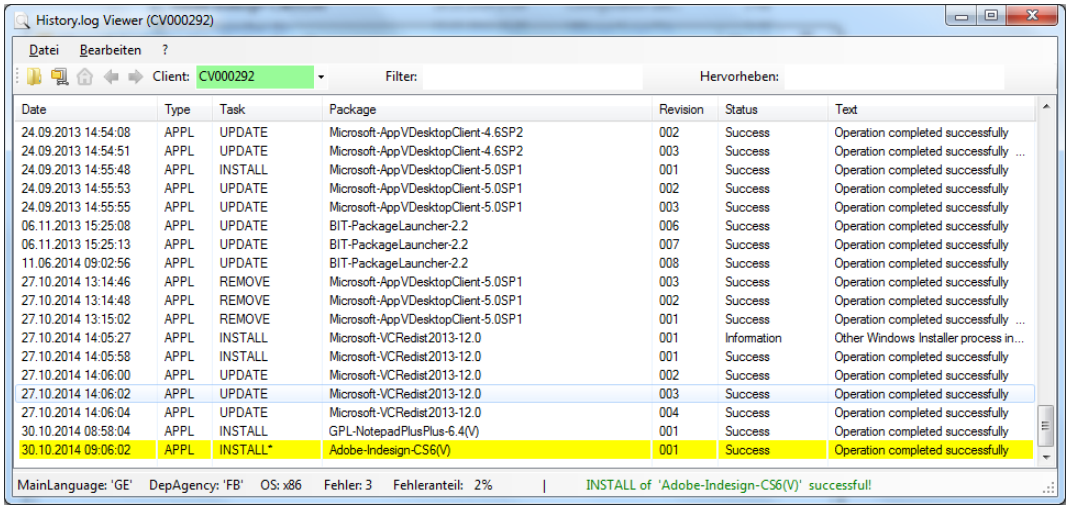


7. Fügen Sie in der Zielanwendung die Abhängigkeit in der INI Datei ein



8. Zum Test installieren Sie **zuerst** die Abhängigkeit durch Klick auf *SCCM-Launcher.vbs.* Danach installieren Sie die Zielanwendung mit der *Connection Group*. Die *Connection Group* wird vom *Package-Launcher* automatisch appliziert.

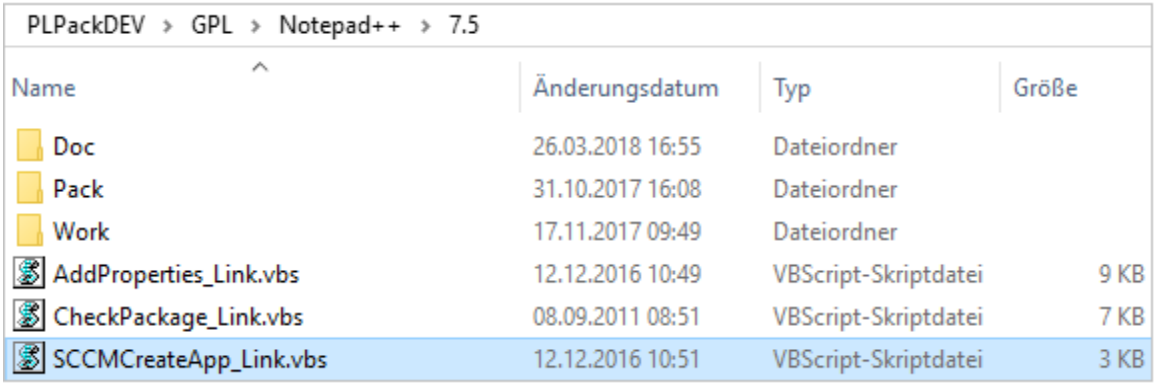
   Sie können auch den *Remote Package Installer* verwenden, welcher automatisch vorgängig alle Pakete installiert.



6.13 SCCMCreateApp – automatisches Überführen in SCCM CB

Nach Abschluss der Paketierungsaufgaben wird mit dem Script *SCCMCreateApp_Link.vbs* (befindet sich im Stammverzeichnis des Softwarepakets) ein Softwarepaket als *Application* in das Softwareverteilungswerkzeug *SCCM CB* überführt. Hauptziel von *SCCMCreateApp* ist es, die notwendigen Aufgaben rationeller und einheitlicher zu erledigen, als wenn diese durch den Softwarepaketierer in manuellen Transaktionen erzeugt werden müssten.

Das Script *SCCMCreateApp_Link.vbs* befindet sich im Stammverzeichnis jedes Softwarepakets:



6.13.1 Umfang der Objekte

SCCMCreateApp erstellt folgende Standard-Objekte:

1. **Application:** Grundsätzlich werden bei der Erstüberführung von Standardpaketen (alle ausser *AppV Full Integration*) 3 verschiedene *Applications* gebaut: Eine *(RTM)-Application*, die alle Paketressourcen zu diesem Zeitpunkt enthält. Weiter werden zwei logische *(Full)-Applications* erstellt. Diese werden für die Zuweisung benötigt.

   Im Rahmen eines *Revisions*updates werden insbesondere bei der produktiven Überführung noch *Revisions*applications gebaut, wo isolierte *Revisionen* integriert sind.

2. **DeploymentType:** Im *DeploymentType* wird eine konforme und einheitliche *DetectionMethod* integriert und die Abhängigkeiten aus dem *Dependence*-Eintrag der INI-Datei implementiert.

3. **Deployment:** Eines für eine komplette versionierte Installation *(Full),* eines für eine unversionierte *(Full)*-Zuweisung und nur in *PRD* eines für die Deinstallation (*Full*).

4. **Collection:** Eine für eine komplette versionierte Installation *(Full Install),* eine für eine unversionierte *(Full Install)*-Zuweisung und nur in *PRD* eine für die Deinstallation (*Full Uninstall*).

Wenn sich in der Datei *SCCMCreateApp.INI* im Bezeichner *PLPackDEV=* kein Eintrag befindet, werden die Objekte für *Revisions*updates nur in der *PRD*-Umgebung erstellt (empfohlen).

6.13.2 Die verschiedenen Environments

Damit in einer produktiven Umgebung nach dem Überführen und Verteilen eines Softwarepakets in Form von *Applications* auch Änderungen und Updateerweiterungen an diesem Paket möglich sind, ohne dass diese Arbeiten inkl. deren Tests die produktive Ausprägung der *Applications* stören, sind verschiedene *Environments* vorgesehen.

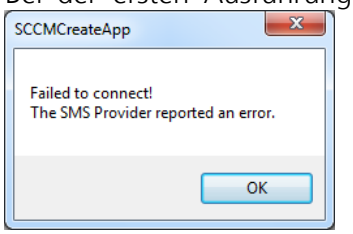
Im Allgemeinen sind im Ablauf folgende Schritte zu vollziehen:

1. Der Softwarepaketierer erstellt, ändert oder updatet sein Paket auf dem *Development*-Share *PLPack**DEV***
2. Der Softwarepaketierer überführt sein fertiges Paket mittels *SCCMCreateApp_Link.vbs* im Environment ***DEV***
3. Die Software wird über die *SCCM*-Objekte in ***DEV*** getestet.
4. Eventuell ergeben sich Korrekturen und die Schritte 1-3 werden wiederholt
5. Ist die Abnahme erfolgreich, erstellt der Überführungsoperator zum Verteilungszeitpunkt die *SCCM*-Objekte mittels *SCCMCreateApp_Link.vbs* im Environment ***PRD***
6. Der Überführungsoperator weist bei einer *RTM*-Integration (Erstüberführung) die Software den Benutzern zu
7. Die *Applications* werden automatisch installiert

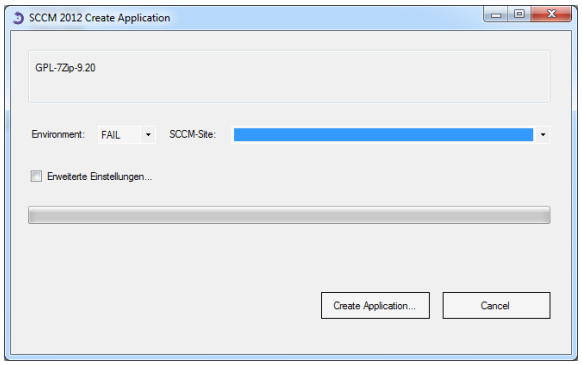
6.13.3 Vorbereitungen zur Bedienung von SCCMCreateApp

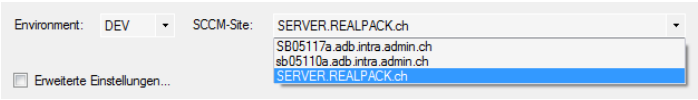
Wie in diesem Hauptkapitel skizziert, wird die Überführung durch das Script *SCCMCreateApp_Link.vbs* aus dem Stammverzeichnis des Softwarepakets, eingeleitet.

Bei der ersten Ausführung der Software kann folgende Fehlermeldung einmalig erscheinen:



Danach ist im Fenster zuerst die *SCCM*-Site, danach das *Environment* auszuwählen:



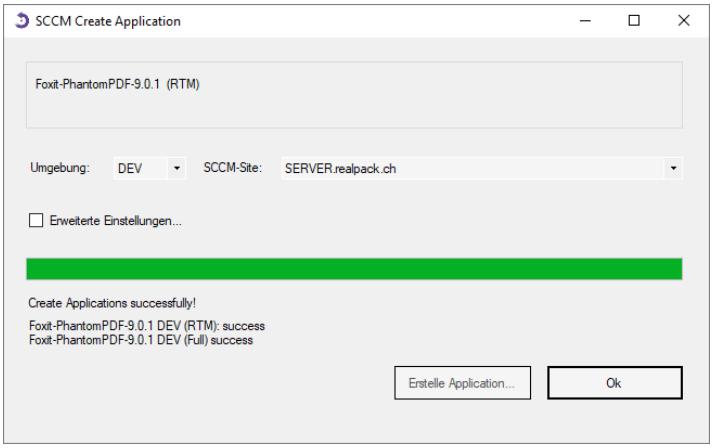


Für die nächste Ausführung werden diese Schritte durch *SCCMCreateApp* automatisch gespeichert.

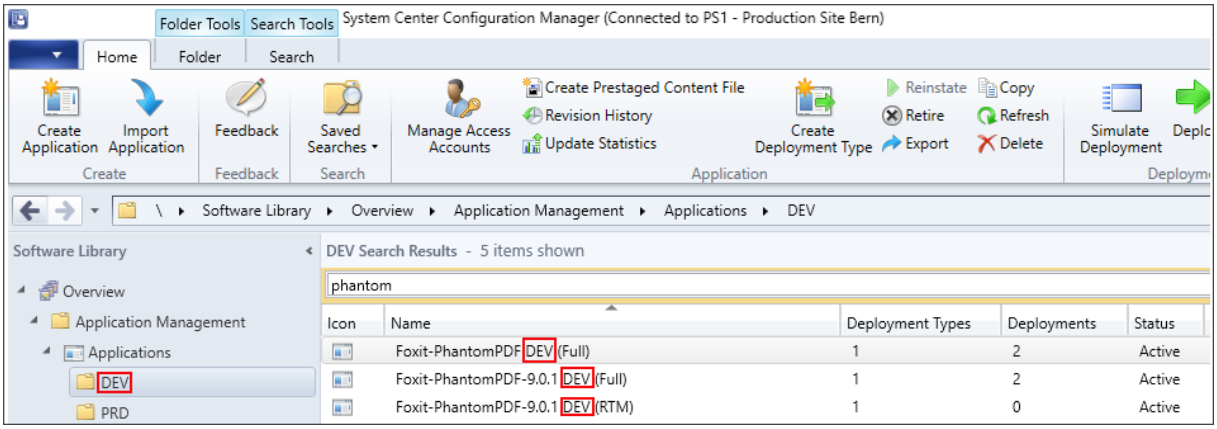
6.13.4 Überführung in DEV und PRD

Doppelklicken Sie im Stammverzeichnis des Softwarepakets auf *SCCMCreateApp_Link.vbs*, wählen Sie das *Environment* und klicken dann auf *Create Applications…*

Innert weniger Sekunden werden alle erforderlichen *SCCM* Objekte auf der Infrastruktur erstellt.



Dabei werden die Objekte in *SCCM* standardmässig in separaten *Environment-Folders* gespeichert. Auch in den Namen der Objekte finden wir das *Environment* wieder (ausser bei PRD nicht).

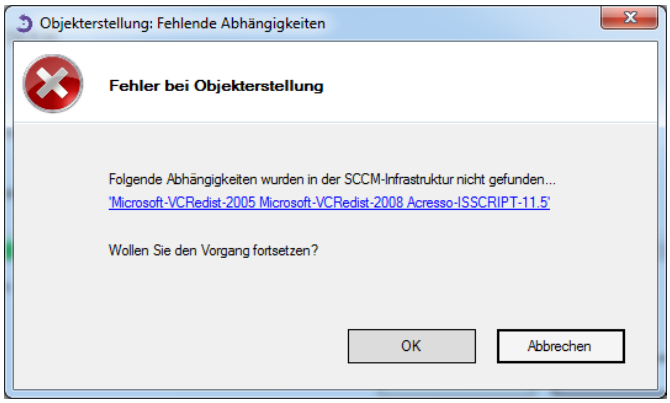


6.13.5 Abhängigkeiten

Abhängigkeiten sind unter Berücksichtigung der Hierarchien einzupflegen (siehe *Merke* im Kapitel *6.10.3 Umgang mit Abhängigkeiten durch Verwendung des „Dependence"-Eintrages*)!

SCCMCreateApp integriert grundsätzlich alle in der Paket-INI-Datei implementierten Abhängigkeiten (Bezeichner *Dependence*) im *DeploymentType* der (*RTM*)-Application, ausser Einträge in der Form von *{ProductCodes}*. Es ist daher erforderlich, dass solche Abhängigkeiten **vorgängig** auf der Infrastruktur integriert wurden.

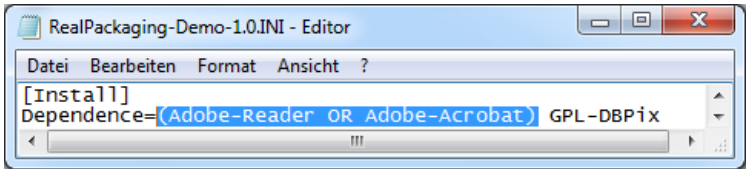
Sollte folgende Fehlermeldung erscheinen…



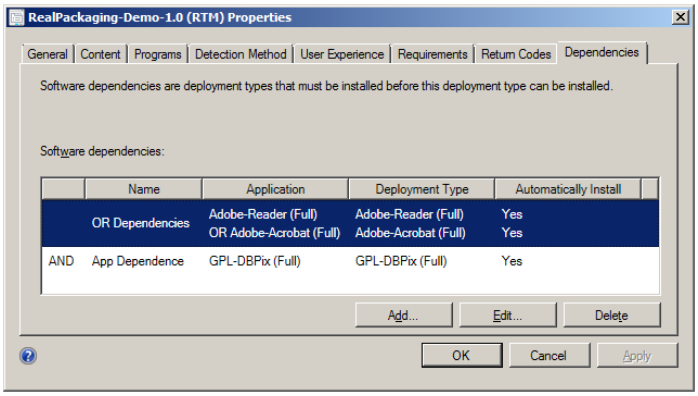
…ist dies darauf zurückzuführen, dass die notwendigen Abhängigkeiten noch nicht überführt wurden. Brechen Sie den Dialog ab und integrieren zuerst die Abhängigkeiten. Danach können Sie abermals das Zielpaket überführen. Jetzt sollte die Fehlermeldung nicht mehr erscheinen.

Verwenden Sie in der INI-Datei Abhängigkeiten mit dem ‚OR'-Operator, werden diese entsprechend der Anweisung und Reihenfolgsprüfung im *Deployment Type* der *Application* integriert.

Ausprägung INI-Datei:



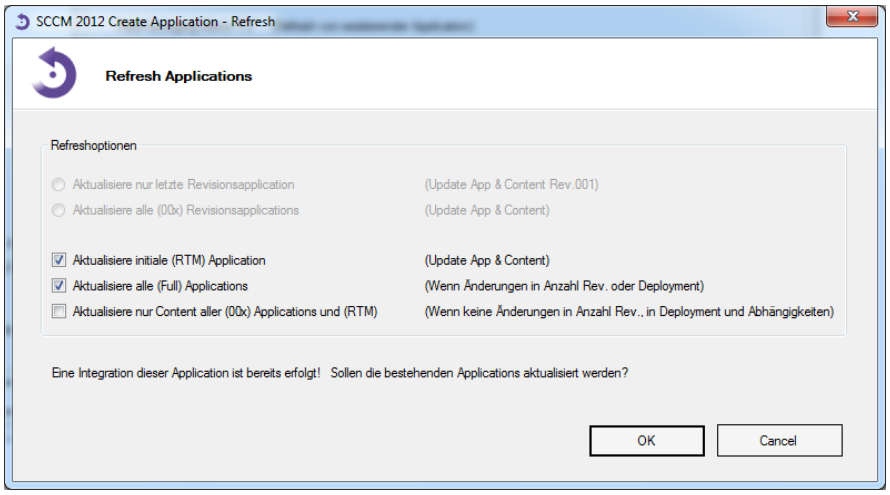
Ansicht *Dependence* im *Deployment Type* der *Application* nach der Überführung des obigen Paketes:



6.13.6 Refresh von Applications

Wird *SCCMCreateApp* für ein Paket ein zweites Mal ausgeführt, so erscheint ein Dialog der folgenden Art:

Optionen der Aktualisierung von Revisionen bei Überführung nach PRD, hier ein Paket mit nur einer Revision:

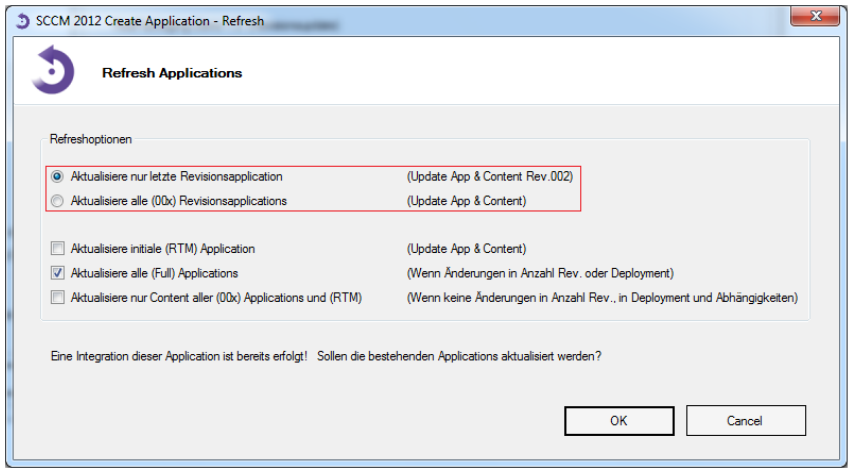


Hier haben Sie die Möglichkeit, anzugeben, welche Objekte genau aktualisiert werden sollen. Da *SCCMCreateApp* nicht weiss, was Sie zwischen der letzten Überführung und der jetzt beabsichtigten Integration für Änderungen am Paket angebracht haben, ist dieser Dialog zur korrekten Umsetzung Ihrer Absichten erforderlich.

Haben Sie Inhalte oder Ressourcen in **bestehenden** *Revisions*verzeichnissen geändert, die zum Umfang der Erstüberführung gehörten, so ist die Option „*Aktualisiere initiale (RTM) Application*" zu markieren.

Gab es Änderungen in der Anzahl an *Revisionen* oder veränderten Sie Deploymenteinstellungen (siehe nächstes Kapitel), so ist die Option „*Aktualisieren alle (Full) Applications*" zu markieren. Bei Refreshoperationen von *Revisions*updates haben Sie zudem die Möglichkeit anzugeben, ob die Aktualisierung nur die letzte *Revision* betrifft oder ob Sie auch auf alte *Revisionen* (*KeepRevision*) angewendet werden soll (*Aktualisiere alle (00x) Revisionsapplications)*. **In der Regel sind die standardmässig automatisch markierten Optionen richtig.** Im Zweifelsfall sollte, wenn man sich nicht sicher ist, die Option *Aktualisiere alle (00x) Revisionsapplications* und *Aktualisiere initiale (RTM) Application* markiert werden.

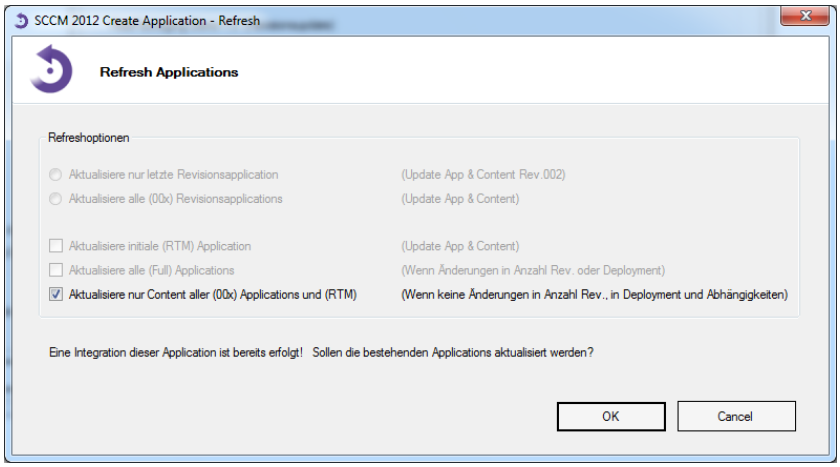
Optionen der Aktualisierung von Revisionen bei Überführung nach PRD:



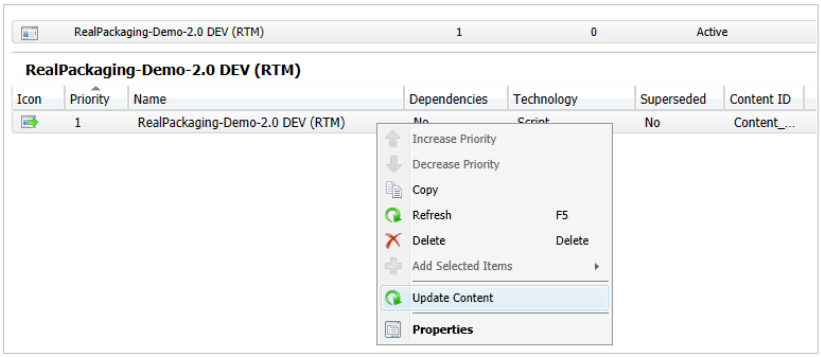
Achtung: Der Dialog unterscheidet sich bei einer Integration auf DEV!

6.13.7 Update Content

Wenn Sie seit der letzten Überführung keine Änderungen in der Anzahl an Revisionen, im *Deployment* (siehe nächstes Kapitel) und in Abhängigkeiten aus der INI-Datei (*Dependence*) vorgenommen haben und eine Aktualisierung des Pakets mit *SCCMCreateApp* vorgenommen wird, so wählen Sie beim angezeigten Dialog „*Aktualisiere nur Content…"* und schliessen Ihre Eingabe mit OK ab.



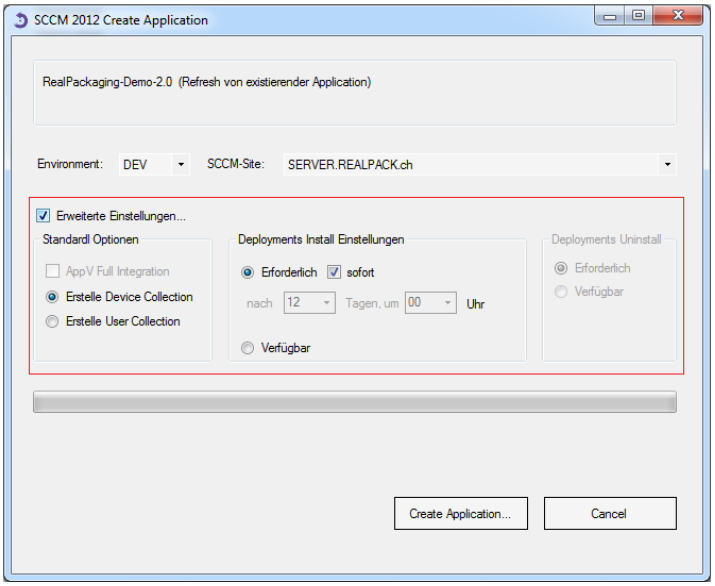
Dies entspricht der Funktion *Update Content* auf dem *Deployment Type* der *Application.*



6.13.8 Deploymenteinstellungen

Wir empfehlen eine Zuweisung per Device mit einer hauptsächlichen Zuweisung von **unversioniert** angebotenen (AVAILABLE) Applications mit dem *Package-Launcher App-Installer* für alle Applications, die nicht zur allgemeinen Basisinfrastruktur gehören. Die Standards applizieren Sie über die Datei *SCCMCreateApp.INI.*

Sollen für einen **Einzelfall** die Einstellungen des *Deployments* geändert werden (nicht empfohlen), kann durch die Markierung der Option „*Erweiterte Einstellungen*" auf einige unterstützte Eigenschaften zurückgegriffen werden:

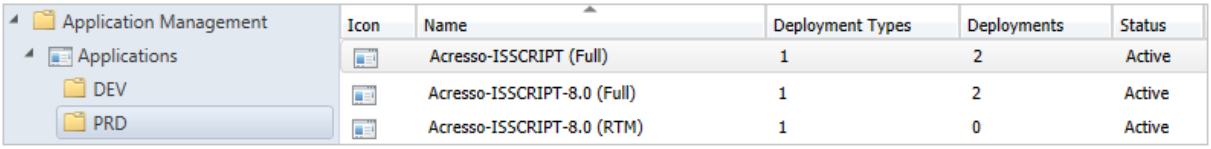


Befinden sich App-V-5 Ressourcen im *Revisions*verzeichnis 001, kann mittels der Option *AppV Full Integration* eine Integration mit Streaming ermöglicht werden. Markiert der Überführungsoperator diese Option nicht, wird das Paket über das *standalone model* wie übliche *Applications* integriert.

Sollen Deploymenteinstellungen **dauerhaft** für alle künftigen Überführungen geändert werden, so sind diese über Eigenschaften in der Datei *SCCMCreateApp.INI* vorzunehmen.

6.13.9 Schematische Darstellung der erstellten Objekte

Ein Standard-*Application*produkt setzt sich initial aus einer *Application Manufacturer-Name-Version (RTM)* mit dem *Content* aller *Revisionen* zum Zeitpunkt der initialen Erstellung, einer *Manufacturer-Name-Version (Full)* und einer unversionierten *(Full)-Application* zusammen. Im *Deployment Type* der *Full-Applications* wird ein **symbolischer** *Content* verwendet, der für eine erfolgreiche Integration von *Applications* in *Task-Sequenzen* erforderlich ist.

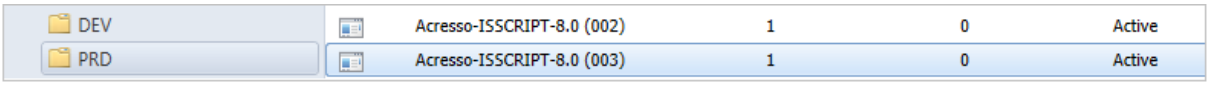


Die versionierte *(Full)-Application* verweist im *Deployment Type* initial als Abhängigkeit auf die (*RTM*)-Application. Die unversionierte *(Full)*-Application hat eine Abhängigkeit auf die versionierte *(Full)*-Application. Produktabhängigkeiten werden durch *SCCMCreateApp* im *Deployment Type* der *(RTM)-Application* eingebaut.

Revisionsupdate

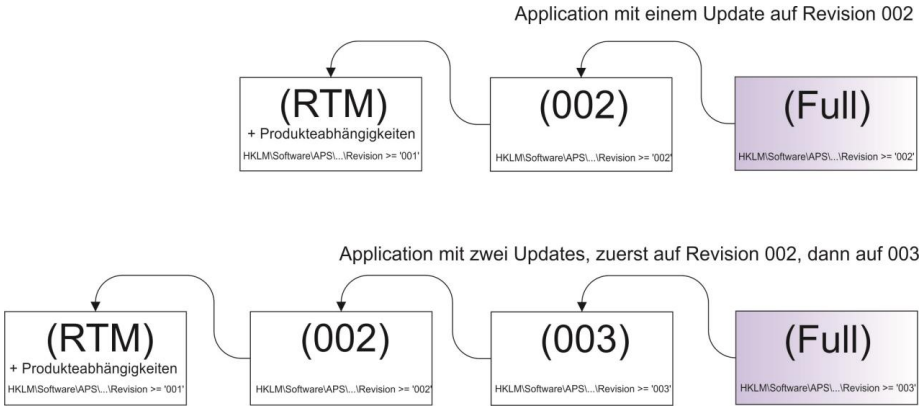
Bei einem *Revisions*update wird für das Update eine neue *Application Manufacturer-Name-Version (00x)* erstellt.

Ansicht der neuen *Updateapplications*:



Zusätzlich wird im *Deployment Type* der *(Full)-Application* die bestehende Abhängigkeit auf *Manufacturer-Name-Version (RTM)* gelöscht, bzw. durch *Manufacturer-Name-Version (003)* ersetzt.

Schematische Darstellung der Updates mit Abhängigkeitsverweisen:

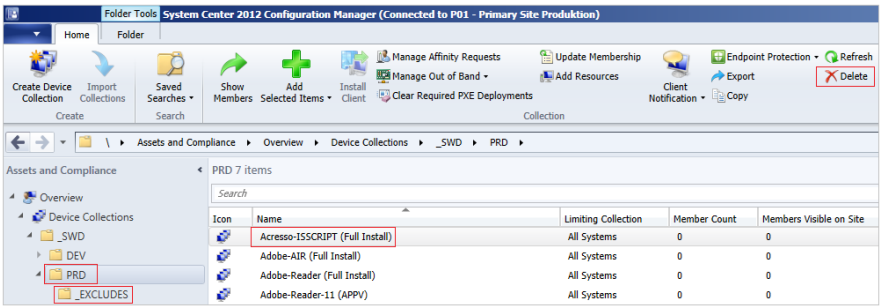


Der Vorteil dieser verketteten Integration ist, dass so eine Reihenfolge in der Abhängigkeitsinstallation vorgegeben werden kann. ***SCCM* ermöglicht applikationsübergreifend keine Steuerung der Reihenfolge von Abhängigkeiten in der selben Hierarchie!**

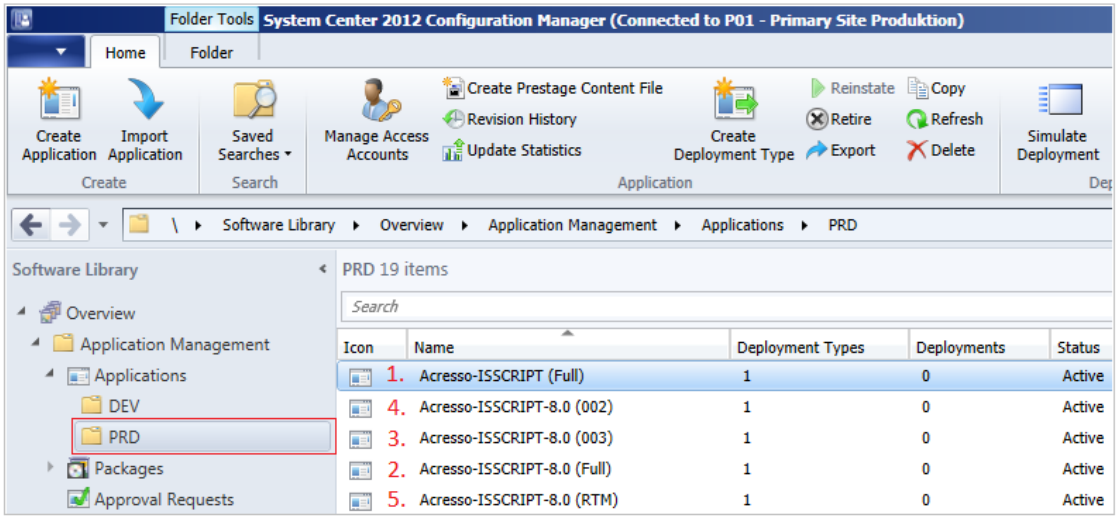
6.13.10 Löschen von Applications

Möchten Sie *Package-Launcher* Produkte in der Infrastruktur löschen, gehen Sie wiefolgt vor:

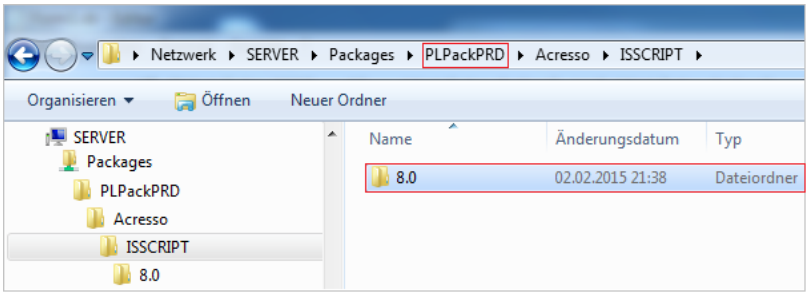
1. Löschen Sie alle *Collections* des zu Löschen vorgesehenen Produktes im *Environment*. Auch die unter *_EXCLUDES*. Sie können dort alle *Collections* gleichzeitig markieren und in einem Schritt löschen. Die Löschoperation löscht gleichzeitig auch alle *Deployments*.



2. Löschen Sie die *Applications* nacheinander in der hier abgebildeten Reihenfolge:
   a. unversionierte *(Full)-Application*
   b. versionierte *(Full)-Application*
   c. wenn vorhanden, *Revisionen* chronologisch absteigend
   d. *(RTM)-Application*



3. Löschen Sie den Ordner des Pakets auf dem *Zielenvironment*, inkl. der darin befindlichen *PRD-Status.INI*.



6.13.11 SCCMCreateApp.INI

In der Datei *SCCMCreateApp.INI* sind in der Regel keine Änderungen vorzunehmen. Zudem sollten Änderungen nur wohlüberlegt und mit grosser Vorsicht durchgeführt werden.

Die wichtigsten Bezeichner:

Key	Settings/SCCMSite
Einsatz	Erforderlich: Verbindung
Beschreibung	Hier sind die Angaben der SCCM Siteserver einzutragen. Werden mehrere verwendet, werden diese durch Semikolon getrennt
Beispiel	[Settings] SCCMSite=SCCMSERVER.RP.ch

Key	**AskForEnvironment**
Einsatz	Erforderlich: Text;Text
Beschreibung	Mittels dieses Bezeichners können die Namen der *Development*- und der *produktiven Umgebung* vorgegeben werden. Standardmässig und empfohlen sind die Namen wie im Beispiel unten.
Beispiel	AskForEnvironment=DEV;PRD

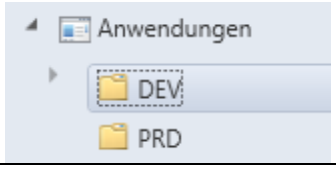
Key	**Scope**
Einsatz	Erforderlich: Verbindung
Beschreibung	Dies sind die Verbindungseigenschaften für den Configuration Manager
Beispiel	Scope=\\G0RSRW-SCCMPS01.source.local\root\Sms\Site_P01

Key	**Site**
Einsatz	Erforderlich: Name der Site
Beschreibung	Verbindungsoption *Site*. Erforderlich für die korrekte Applicationerstellung
Beispiel	Site=P01

Key	**TopLevelFolderCol**
Einsatz	Optional: Foldername
Beschreibung	Der Name des *Top Level Folders* in der Collectionstruktur, worin die *Collections* erstellt werden sollen.
Beispiel	TopLevelFolderCol=Software

Key	**TopLevelFolderApp**
Einsatz	Optional: Foldername
Beschreibung	Der Name des *Top Level Folders* in der Applicationstruktur, worin die *Applications* erstellt werden sollen.
Beispiel	TopLevelFolderApp=Application

Key	**ExcludeDistributionPoint**
Einsatz	Optional: Name;Name
Beschreibung	Namen der Distributionpoints, die beim Überführen ausgeschlossen werden sollen.
Beispiel	ExcludeDistributionPoint=SB032010;SB022010

Key	**DEVPRDFolder**
Einsatz	Erforderlich: True¦False
Beschreibung	Sollen die *Environment*namen als *Folder* in *Collections* und *Applications* eingesetzt werden, so ist dieser Bezeichner auf True zu stellen. Default=True 
Beispiel	DEVPRDFolder=True

Key	**UnversionedUnInstall**
Einsatz	Optional: True¦False
Beschreibung	Mit 'True' werden unversionierte Collections für UnInstall erstellt.Steht der Parameter auf 'False' werden diese nicht erstellt.
Beispiel	UnversionedUnInstall=True

Key	**ManufacturerFolder**
Einsatz	Optional: True¦False
Beschreibung	Sollen pro Hersteller separate Folders in Collections und Applications erstellt werden, ist dieser Bezeichner auf True zu stellen. Default=False.
Beispiel	ManufacturerFolder=False

Key	**UseColSubFolder**
Einsatz	Optional: Pattern;Pattern;Pattern…
Beschreibung	Sollen gewisse Collections, die nicht oft gebraucht werden, in einen Subfolder verschoben werden, kann dies mit diesem Bezeichner bewerkstelligt werden. Es werden folgende Variablen unterstützt: [Versioned]Dies entspricht der versionierten Collection [Unversioned] Dies entspricht der unversionierten Collection [Rev] Dies entspricht einer Revisionscollection
Beispiel	UseColSubFolder=(RTM);[Versioned];[Rev]

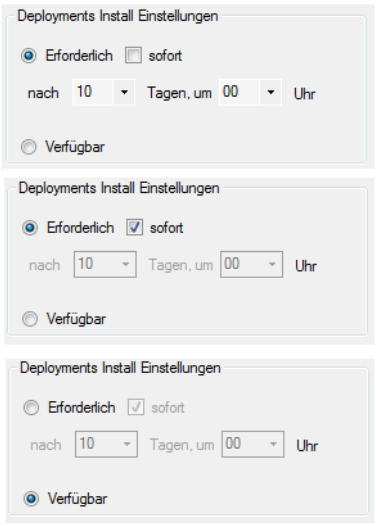
Key	**ExpliciteDontUseSubFolder**
Einsatz	Optional: Pattern;Pattern;Pattern…
Beschreibung	Die Regel *UseColSubFolder* kann für einzelne Namen verhindert werden.
Beispiel	ExpliciteDontUseSubFolder=(APPV)

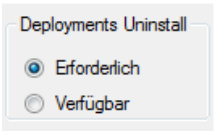
Key	**UseColSubFolderName**
Einsatz	Optional: Name
Beschreibung	Namen des *Exclude* Subfolders in *Collections*.
Beispiel	UseColSubFolderName=_EXCLUDES

Key	**Usercollections**
Einsatz	Optional: True¦False
Beschreibung	Sollen auf der selektierten Infrastruktur *User*- oder *Devicecollections* erstellt werden? Dieser Bezeichner kann auch mit vorangestelltem *Environment*bezeichner verwendet werden, um auf einer Developmentumgebung andere Deploymenteigenschaften zu verwenden, als auf einer produktiven Umgebung. Bspl ***DEV_****Usercollections=False*
Beispiel	Usercollections=False

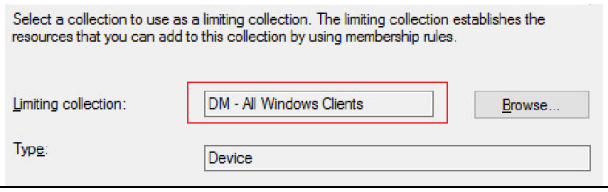
Key	**AvailableForUnversionedApps**
Einsatz	Optional: True¦False
Beschreibung	Regelt, ob unversionierte Apps im Deployment Available oder Required zugewiesen werden sollen. Steht die Eigenschaft auf 'True', dann werden die Deployments für unversionierte Apps als 'Available' markiert.
Beispiel	AvailableForUnversionedApps=True

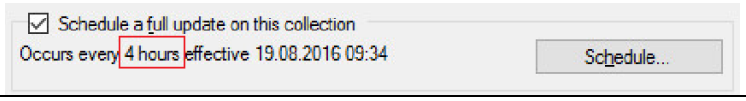
Key	**DeploymentDefaultScheduleDay=10**
Einsatz	Optional: Anzahl Tage (1-99)
Beschreibung	spezifische REQUIRED-Wartezeit in Tagen für Environment einlesen Dieser Bezeichner kann auch mit vorangestelltem *Environment*bezeichner verwendet werden, um auf einer Developmentumgebung andere Deploymenteigenschaften zu verwenden, als auf einer produktiven Umgebung. Bspl ***DEV_****DeploymentDefaultScheduleDay=10*
Beispiel	DeploymentDefaultScheduleDay=10

Key	**DeploymentRequiredINSTALL**
Einsatz	Optional: True¦False¦Mixed
Beschreibung	Einstellungen, ob Deployment *Required* oder *Available*.  Mixed True False Dieser Bezeichner kann auch mit vorangestelltem *Environment*bezeichner verwendet werden, um auf einer Developmentumgebung andere Deploymenteigenschaften zu verwenden, als auf einer produktiven Umgebung. Bspl ***DEV_****DeploymentRequiredINSTALL=True*
Beispiel	DeploymentRequiredINSTALL=True

Key	**DeploymentRequiredUNINSTALL**
Einsatz	Optional: True¦False
Beschreibung	Einstellungen, ob UnInstall-Deployment *Required* oder *Available*. 
Beispiel	DeploymentRequiredUNINSTALL=True

Key	**AvailableForUnversionedApps**
Einsatz	True¦False
Beschreibung	Einstellung, unversionierte Application als AVAILABLE, statt REQUIRED deployed werden sollen (mit *Package-Launcher App-Installer* empfohlen).
Beispiel	AvailableForUnversionedApps=True

Key	**LimitCol=DM - All Windows Clients**
Einsatz	Optional: Collection Name
Beschreibung	Soll in Collections eine eigene *Limited Collection*, statt *All Systems* integriert werden, kann dies über diesen Bezeichner erreicht werden. 
Beispiel	LimitCol=DM - All Windows Clients

Key	**ColHourSpan**
Einsatz	Optional: Collection Schedule
Beschreibung	Soll in *Collections* eine der *Collection Update Cycle* angepasst werden, wird dies über diesen Bezeichner erreicht. 
Beispiel	ColHourSpan=4

Key	**PreventNotifyUser**
Einsatz	Optional: True|False
Beschreibung	Mit diesem Bezeichner werden alle SCCM-Benutzerbenachrichtigungen für alle Applications (versioniert und unversioniert) abgeschaltet.
Beispiel	PreventNotifyUser=True

Key	**SourceDEV**
Einsatz	Optional: Share
Beschreibung	Name des *PLPackDEV*-Shares. Auf diesem Share/Ablage befinden sich die Softwarepakete, die in Entwicklung sind. Ist unter diesem Bezeichner kein Share/Ablage verzeichnet, werden in der DEV-Umgebung nur volle *Applications* mit allen *Revisionen* im Inhalt erstellt. Ein Updatemechanismus funktioniert in diesem Fall nicht.
Beispiel	SourceDEV=\\SERVER\PLPackDEV

Key	**SourcePRD**
Einsatz	Erforderlich: Share
Beschreibung	Name des *PLPackPRD*-Shares. Auf diesem Share/Ablage befinden sich die produktiven Softwarepakete. *SCCMCreateApp* kopiert das Softwarepaket bei der Überführung in *PRD* auf diesen *PLPackPRD*-Ordner.
Beispiel	SourceDEV=\\SERVER\PLPackDEV

Key	ReqPrimaryDevice
Einsatz	Optional: True|False
Beschreibung	Den Applications ein Requirement hinzufügen: "Primary device Equals True". Dies ist für Per-User Deployments sinnvoll.
Beispiel	ReqPrimaryDevice=True

Key	DepPreDeploy
Einsatz	Optional: True|False
Beschreibung	Unterstützung von "PreDeployment to Users Primary Device" in Per-User Umgebungen.
Beispiel	DepPreDeploy=True

Key	AD-LDAPDir
Einsatz	Optional: Text
Beschreibung	Sollen AD-Gruppen erstellt werden, wird hier das LDAP Directory angegeben
Beispiel	AD-LDAPDir=OU=_Groups,OU=CLT,OU=Production,DC=SERVER,DC=RP,DC=ch

Key	AD-Group
Einsatz	Optional: Text
Beschreibung	Der Bezeichner enthält den Namen der zu erstellenden AD-Gruppe. Die Variable *[PACKAGE]* wird aufgelöst und mit dem Collectionnamen ersetzt.
Beispiel	AD-Group=G_APP_[PACKAGE]

Key	Query
Einsatz	Optional: Query (Text)
Beschreibung	Hier kann ein Query (WQL Statement) angegeben werden, welches der Collection hinzugefügt wird. Die Variable *[PACKAGE]* wird bei der Integration aufgelöst und mit dem CollectionNamen ersetzt. Der Bezeichner kann nur zusammen mit AD-LDAPDir verwendet werden!
Beispiel	Query=SELECT * FROM SMS_R_UserGroup WHERE SMS_R_UserGroup.UserGroup-Name LIKE '%[PACKAGE]'

6.14 Testen eines Softwarepaketes

Das Testen des Softwarepakets ist eine zentrale Aufgabe des Softwarepaketierers. Nur durch ein Testing können allfällige Fehler vor einer Paketverteilung erkannt und korrigiert werden.

Der Testablauf kann von Fall zu Fall unterschiedlich ausfallen und ist unternehmensspezifisch.

6.14.1 INSTALL, REPAIR, REMOVE

Das Softwarepaket ist zwingend im Installationskontext *INSTALL*, *REPAIR* und *REMOVE* zu prüfen. Zum Teil reagiert eine Installation beim *REPAIR* anders als bei der Initialinstallation. Geprüft wird, ob der Prozess positiv abgeschlossen wird und ob keine Fehlermeldungen in der Datei *History.LOG* ersichtlich sind.

6.14.2 Upgrade testen

Die Implementationen, die in der *INI*-Datei des Softwarepakets zum Einsatz kommen, sind zu prüfen. Insbesondere stellt der Softwarepaketierer bei einer *Initialrevision* auch ein *Upgrade* nach, wenn eine Vorversion in der Umgebung des Unternehmens vorliegend ist.

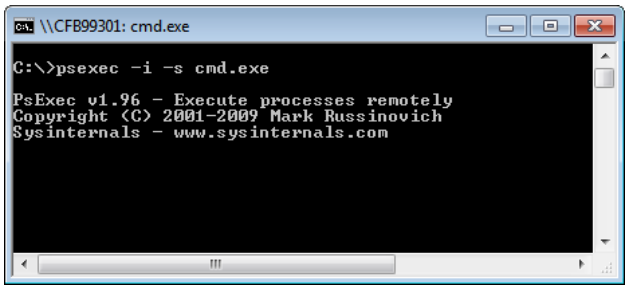
6.14.3 Lizenz und Aktivierungsstatus testen

Wurde im Paket eine Lizenzaktivierung implementiert, ist die Software darauf zu prüfen, ob dieser Aktivierungsprozess erfolgreich war. Durch temporäres Verstellen der Systemzeit auf ein in der Zukunft liegendes Datum kann geprüft werden, ob die Lizenz auf eine gewisse Zeit nach der Installation beschränkt ist.

6.14.4 Manuelles Installieren im Systemkontext

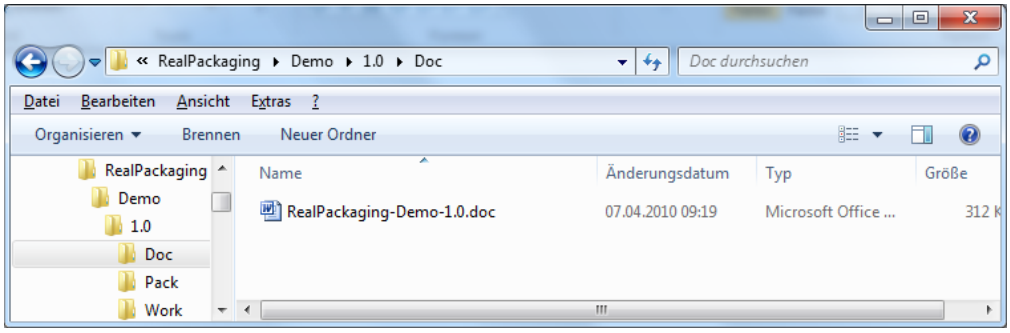
Das Softwarepaket ist unbedingt mittels einer Testinstallation mit *SCCM* auf einem produktionsnahen Testgerät zu prüfen. Treten Probleme auf, die im Ausführungskontext vermutet werden, kann eine weitere Analyse und ein Schnelltest mit dem *Remote Package Installer* oder mit den *PSTools* von Microsoft (ehemals SysInternals) hilfreich sein. Mit dem folgenden Aufruf startet man mit *psexec* eine Kommandozeile mit lokalen Systemrechten:

```
psexec -i -s cmd.exe
```



6.15 Paketdokumentation erstellen

Aus Sicht der späteren Reproduzierbarkeit (beispielsweise bei einer neuen Version) sollten alle Veränderungen, die der Softwarepaketierer implementiert hat, bzw. den Ablauf der bei der Paketierungsarbeit zielführend war, dokumentieren! Das Verzeichnis „Doc“ ist für die Dokumentation vorgesehen.



6.16 Signierung und Einbindung von Treibern

Zur Signierung von Treibern mit einem entsprechenden Zertifikat ist gem. Anleitung von Microsoft vorzugehen: *http://technet.microsoft.com/en-us/library/cc754052.aspx*.

Das Einbinden eines Treibers in ein Softwarepaket kann am einfachsten mit *DPInst.EXE* erledigt werden, welches in einem *Pre-* oder *PostInstall* oder einem Script zur Ausführung kommt.

6.17 Software verteilen mit dem Package-Launcher App-Installer

Der richtige Zeitpunkt zum Installieren und Verteilen von Software ist bis heute eines der herausforderndsten Themen in der Softwarebereitstellung in Unternehmen. Mit *SCCM* lassen sich zwar Wartungsfenster konfigurieren, um die Produktivität des einzelnen Mitarbeiters und die des Unternehmens bei Softwareinstallationen möglichst nicht zu beeinträchtigen, bzw. eine fehlerfreie Installation dadurch zu gewährleisten, dass diese zu Zeiten durchgeführt wird, wo der Benutzer nicht mit Programmen am Arbeiten ist. Auch die Installation per *Wake On LAN* zielt darauf, die Softwareinstallation zu Zeiten auszuführen, wo der Benutzer nicht angemeldet ist und es somit während der Installationstransaktion zu keiner Beeinträchtigung durch geöffnete Programme kommt.

Es gibt auch Verteilszenarien, wo die Installation im abgemeldeten Clientzustand durchzuführt wird oder wo man während der Installation selbst den Mechanismen des *Windows Installer Restart Managers* oder denen der *Pending Files Rename queue (PFR)* zum Ersetzen von in-use-Dateien vertraut.

Oft reichen die letztgenannten Varianten aber nicht aus und es kommt durch Installations- und Deinstallationsprozesse, zu Zeiten, wo die Benutzer Programme am Computer geöffnet haben, zu Seiteneffekten. So, dass…

1. die Installation oder Deinstallation mit Fehler und/oder Rollback abgebrochen wird oder…
2. die Software nach der Installation mit Fehler im Funktionsumfang reagiert

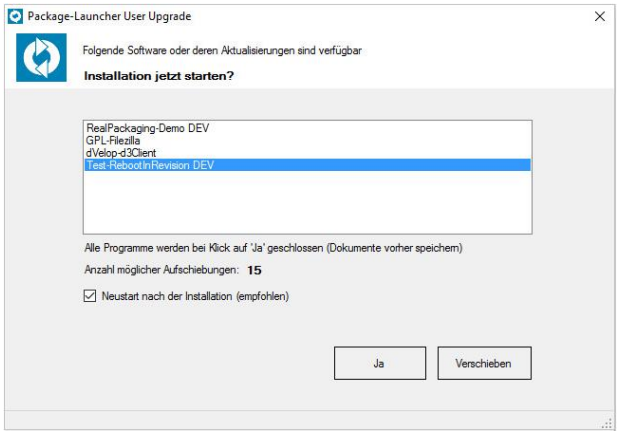
Mit dem *Package-Launcher App-Installer* lassen sich solche Störungen auf elegante Weise verhindern. Denn er ermöglicht das sichere Schliessen von Anwendungen kurz vor der geplanten Installation. Er zielt auf Neuinstallationen und *Upgrades* von **unversionierten** und per Device angebotenen Applications. Also solchen Applications, die dem Upgradepfad und dem Lifecycle der Anwendung folgen.

Der *Package-Launcher App-Installer* bietet folgende Vorteile:

1. Den Einbezug des Benutzers, so dass er für angebotene Erstinstallationen und *Upgrades* ein zeitlich begrenztes Veto einbringen kann. Heisst, wenn er gerade beschäftigt ist, kann er das Installations- und/oder Upgradeangebot temporär verschieben.
2. Der an sich heiklere Prozess für *Upgrades* kann in einer sicheren Umgebung durchgeführt werden, weil Benutzeranwendungen vorher geschlossen werden.

Der *Package-Launcher App-Installer* präsentiert sich nach dem Zuweisen von unversionierten *Applications* wiefolgt (startet 15 Minuten nach dem Anmelden und später jede Stunde (xx:15) nach Verfügbarkeit des Deployments von selbst):

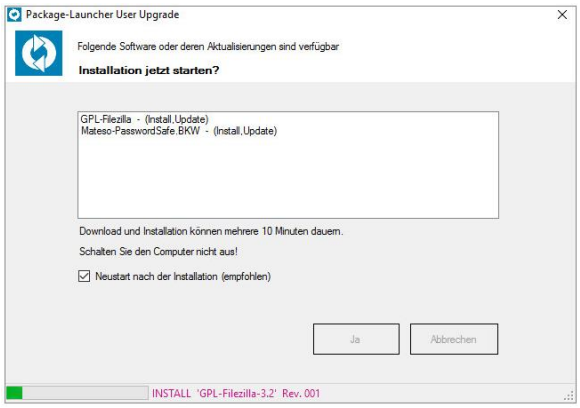
App-Installer Fenster, welches in regelmässigen Abständen erscheint und die zugewiesene Software zur Installation anbietet:



Dieses Fenster erscheint in regelmässigen Abständen (max. alle 3h, nachdem einmal auf «Verschieben» geklickt wurde). Je nach Anzeigesprache erscheint der Dialog in Deutsch, französisch oder Englisch. Der Text "Alle Programme werden geschlossen…" wird für die ersten fünf Installationstransaktionen mit roter Farbe dargestellt, so dass dieser nicht übersehen werden kann.

Das Fenster kann auch durch den Benutzer im Startmenü gestartet werden. Sind alle Aufschiebungen aufgebraucht (Zähler auf 0), wird die Installation nach Rückfrage automatisch ausgeführt.

Bei Klick auf «Ja» werden alle Anwendungen geschlossen, der Desktop wird schwarz und die Installation wird gestartet. Ein Doppelklick auf die Statuszeile öffnet den *History.LOG Viewer*, um detailliertere Informationen zu erhalten.



Verlangt eine Anwendung einen Neustart, wird die Checkbox «Neustart nach der Installation» ausgegraut und im Anschluss wird automatisch ein Neustart ausgelöst.

Bedenken Sie, dass *Revisionsupdates* nachwievor im Hintergrund ohne Benutzerinteraktion aktualisiert werden und dieser Dialog nur bei Aktualisierungen in Form von unversioniert zugewiesenen *Upgrades* und Erstinstallationen erscheint.

6.17.1 Wie funktioniert der App-Installer im Detail?

Nach der Zuweisung eines Members in einer unversionierten Collection werden heikle Installationen und Upgrades (Entfernen alter Version – Installation neuer Version) mit dem *App-Installer* realisiert, um diese in einem sicheren Installationskontext ohne laufende Benutzeranwendungen durchzuführen. Dadurch sollen letztlich Störungsfälle reduziert werden. Die Praxis hat ganz klar gezeigt, dass dieses Ziel mit diesen Massnahmen erreicht wird. Zudem erlaubt der *App-Installer* den Einbezug des Benutzers, damit dieser die anstehenden Upgrades zu einem für ihn passenden Zeitpunkt durchführen kann.

Im Rahmen der Upgrades mit dem *App-Installer* muss auch immer ein Spagat zwischen den terminlichen Anforderungen einer Softwareausrollung und dem Nutzererlebnis gemacht werden. Jedes Upgrade hat einen Grund der Aktualisierung: In manchen Fällen werden damit wichtige Security-Issues der alten Version behoben, in anderen Fällen muss eine neue Version koordiniert mit einer Serveranpassung ausgerollt werden. Der *App-Installer* bietet hier einen Kompromiss dieser diametral entgegengesetzten Bedürfnisse der terminlichen Softwareausrollung und dem Nutzererlebnis.

App-Installer Verschiebungen (Phase 1):

Die Zyklen der Anzeige des *App-Installers* sind standardmässig auf 3h festgelegt. Das heisst, an einem durchschnittlichen Arbeitstag erscheint die Anzeige während dem Anwesenheitsfenster max. 3 x pro Tag. Insgesamt stehen dem Benutzer 15 (offizielle) Verschiebungen zur Verfügung. Wird das Gerät nach Arbeitsende heruntergefahren, überdauert ein anstehendes Deployment 4 ganze Arbeitstage, bis das Deployment den Benutzer zur sofortigen Auslösung auffordert. Ohne Abmelden nach Arbeitsende kann es sein, dass sich diese Zeit auf 3 Arbeitstage verkürzt:

Zielzeitenschema (**Beispiel**) vom App-Installer:

Abmelden nach Arbeitsende	Verschieben	Uhrzeit	Verschieben	Uhrzeit	Verschieben	Uhrzeit	Verschieben	Uhrzeit
Tag 1	15	08:15	14	11:15	13	14:15	12	17:15
Tag 2	11	08:15	10	11:15	9	14:15	8	17:15
Tag 3	7	08:15	6	11:15	5	14:15	4	17:15
Tag 4	3	08:15	2	11:15	1	14:15	**0**	17:15

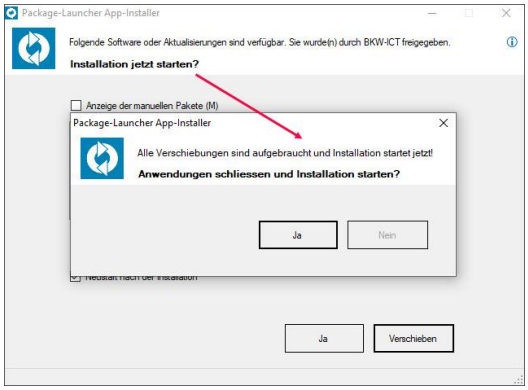
Ohne Abmelden nach Arbeitsende mit Wochenende	Verschieben	Uhrzeit	Verschieben	Uhrzeit	Verschieben	Uhrzeit	Verschieben	Uhrzeit	Verschieben	Uhrzeit	Verschieben	Uhrzeit
Tag 1	15	08:15	14	11:15	13	14:15	12	17:15	11	20:15	10	23:15
Samstag	9	08:15										
Sonntag	8	08:15										
Tag 2	7	08:15	6	11:15	5	14:15	4	17:15	3	20:15	2	23:15
Tag 3	1	08:15	**0**	14:15								

Im Hinblick des weiter oben beschriebenen Spagats sollte für Anwender mit 3 bis 4 Arbeitstagen eigentlich genug Zeit verbleiben, um sich kurz der Aufgabe der Softwareaktualisierung widmen zu können. Bei korrekter Planung führt der Benutzer die Aktualisierungsaufforderung kurz vor Arbeitsende aus, womit auch keine Arbeitszeit verloren geht.

App-Installer Enforcements (Phase 2):

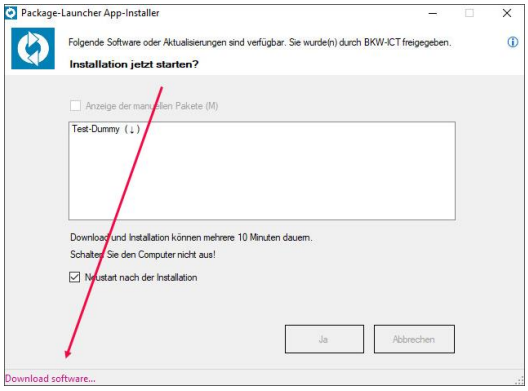
Wird den Aufforderungen nicht Folge geleistet und die Softwarepakete verbleiben in der Warteschlage, triggert der *App-Installer* die Installation ab 0 Verschiebungen automatisch. Dadurch wird ein Fenster angezeigt, das auf diesen Vorgang hinweist. Hier gibt es noch zweimal (bis Verschiebungen -1) eine Möglichkeit, den Dialog mit "Nein" abzubrechen:

Nach dem nächsten Erscheinen (Verschiebungen <= -2) kann das Vordergrundfenster zwar noch verschoben, aber nicht mehr geschlossen werden. Das Hintergrundfenster bleibt zentriert angezeigt und lässt sich auch nicht mehr schliessen. Es bleibt aber die Möglichkeit im Hintergrund geöffnete Dokumente abzuspeichern.



App-Installer Enforcements (Phase 3):

Werden die Aufforderungen weiterhin nicht beachtet, erfolgt die Softwareinstallation der anstehenden Pakete ab -5 Verschiebungen automatisch ohne Schliessen der Vordergrundanwendungen und ohne direkt initiierten Neustart.

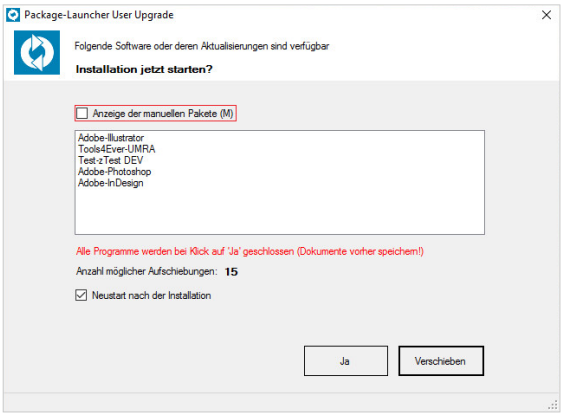


Diese Phase 3 ist als Notlösung gedacht. Die Art der automatischen Installation ohne Schliessen der Anwendungen sollte nicht die primäre Ausführungsmethode sein, da sie generell anfälliger für Störungsprobleme ist. Im flächendeckenden Deployment ist sie aber vertretbar, da über das vorgängige Verhalten davon ausgegangen werden kann, dass nur noch ein kleiner Teil der Installationen so getriggert würde.

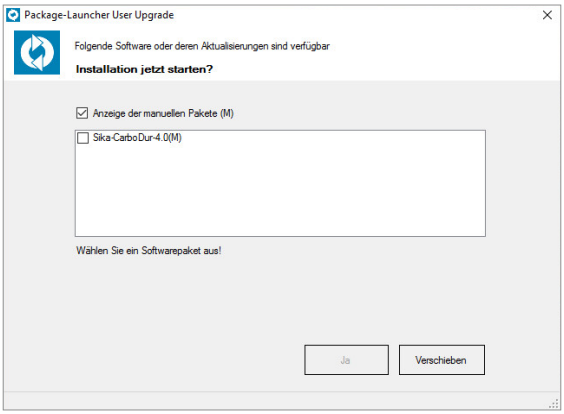
Phase	Verschiebungen	Massnahme
1	15-0 Verschiebungen	Während den Arbeitszeiten wird in regelmässigen Abständen der App-Installer zur Anzeige gebracht
2	0 bis -5 Verschiebungen	Die **Installation** wird im App-Installer **getriggert** und der **Dialog** zum Bestätigen des Schliessens der Anwendungen steht an. Das Fenster kann ab -2 Verschiebungen nicht mehr geschlossen werden.
3	-5 bis -999 Verschiebungen	Die Installation wird vom App-Installer ohne Schliessen der Anwendungen und ohne Neustart getriggert (silent) und im Hintergrund ausgeführt.

6.17.2 Manuelle Pakete

Auch manuelle Pakete, die es nur versioniert gibt, können mit dem *Package-Launcher App-Installer* Assistenten installiert werden. Sind solche verfügbar und dem Client zugewiesen, wird eine Checkbox im oberen Teil der Anzeige sichtbar:



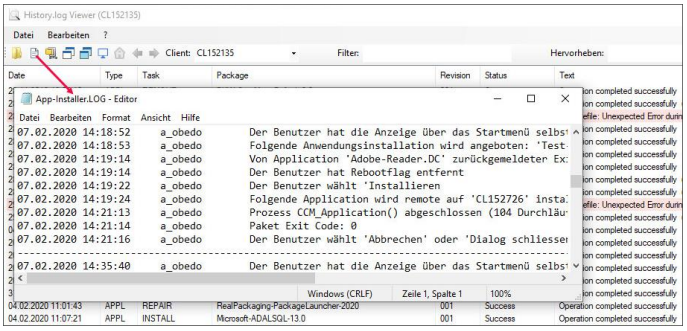
Klickt man auf die Checkbox, zeigen sich die verfügbaren manuellen Pakete, die nun einzeln selektiert und mit Benutzeranzeige installiert werden können:



Zu beachten ist, dass hier vor der Installation Programme nicht automatisch beendet werden, während der Installation der Desktop nicht schwarz wird und nach der manuellen Installation kein Neustart ausgeführt wird.

6.17.3 Spezialitäten

- Mit der Taste F5 wird die Ansicht aktualisiert.
- Mittels Shift-F5 werden SCCM Policies (Computerrichtlinien & Evalusationszyklus für Anwendungsbereitstellung) ausgelöst und zyklisch die Ansicht aktualisiert, bis in der Listbox Applications angezeigt werden. Die laufende Funktion wird in der Statuszeile angezeigt:
  Refresh Policies=True
- Im %PL%\..\Logs-Verzeichnis wird eine Protokolldatei "App-Installer.LOG" erstellt, woraus Informationen für Support bezogen werden können (einsehbar per History.LOG Viewer).



- Läuft die Netzverbindung über Direct Access, erscheint die Anzeige der verfügbaren Applikationen nur nach manuellem Start der Anwendung.
- Treten Fehler während der Installation auf, wird dies am Ende der Transaktion angezeigt. Ein Neustart wird nur ausgelöst, wenn die Application einen Neustart verlangt hat (Bspl. Reboot=True).

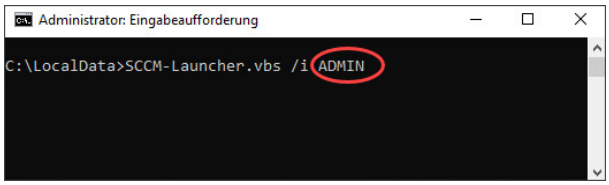
7 Spezialfälle und besondere Eigenschaften

7.1 Varianten

Es gibt drei verschiedene Typen, aus einem Paket verschiedene Ausprägungen zu gestalten: *Paketvarianten*, *Instanzvarianten* und *regionenspezifische Varianten*:

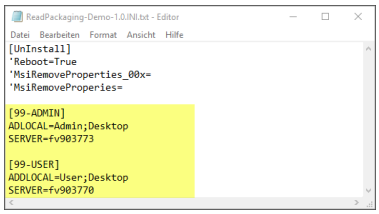
7.1.1 Paketvarianten

Paketvarianten dienen dazu, aus ein und demselben Softwarepaket verschiedene Ausprägungen installieren zu können. So sind aus der selben Paketsource von Office beispielsweise das Office, aber auch ein Sharepointdesigner und auch ein Visio realisierbar. Die Paketvariante wird hierbei einfach dem Aufrufescript als zusätzliche Kommandozeilenerweiterung übergeben.



Der Softwarepaketierer kann über die übergebene Variante Installationsanweisungen implementieren, die ausschliesslich für die übergebene Variante gelten.

Ausprägungen von *Windows Installer Properties* können sich je nach vorgesehener *Variante* unterscheiden. Hier platziert der Softwarepaketierer die *Properties* in der *INI*-Datei des Softwarepakets.



Bei der Ausführung von *SCCM-Launcher.vbs /i ADMIN* werden im obigen Beispiel die Windows Installer Properties auf *ADDLOCAL=Admin;Desktop SERVER=fv903773* gesetzt. Zudem steht dem *Pre-* und *PostInstall* neben der Variablen *%VARIANT%* auch die Variable *%VARIANTSECTION%* zur Verfügung. Diese Variable enthält den Sectionbezeichner *99-ADMIN,* der vom *Package-Launcher* ausgelesen wurde. Somit kann auch aus einem *Pre-* und *PostInstall* und aus Scripts auf die Inhalte der angewendeten *Properties* zugegriffen werden. Weiter stehen die in der *VARIANTSECTION* platzierten Properties als Environmentvariablen zur Verfügung. Diese Variablen werden mit "PL_" eingeleitet:

Aus der Ausprägung in der INI…

```
[99-STANDARD]
SERVER=FV903770
```

…erstellt der *Package-Launcher* die folgende Environmentvariable:

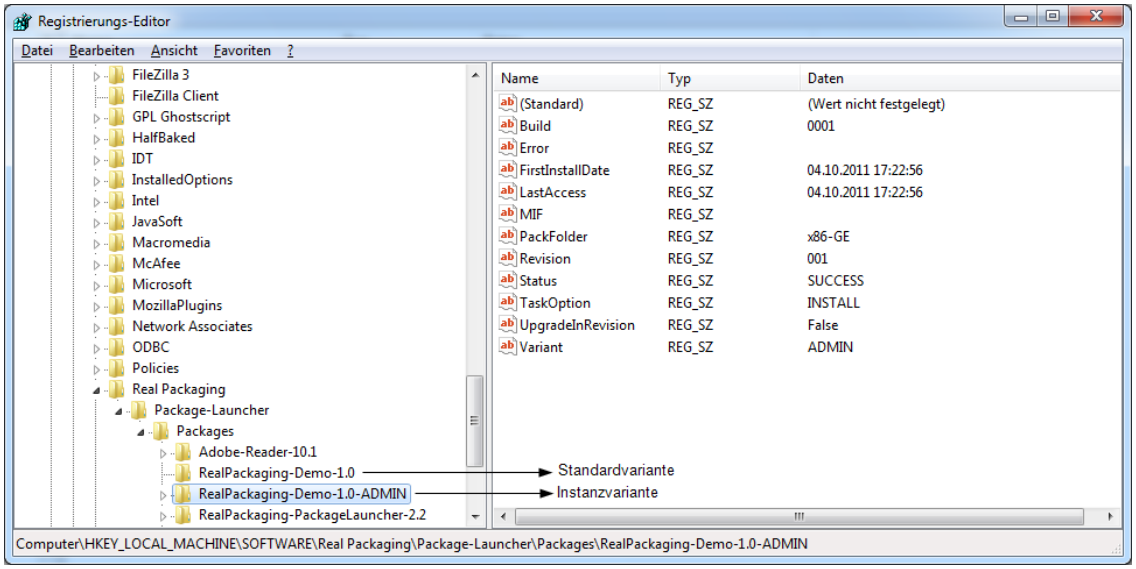
```
PL_SERVER=FV903770
```

Die Überführungsanpassungen für die *Varianten* müssen in *SCCM* manuell getätigt werden. Mit dem Überführungsscript *SCCMCreateApp_Link.vbs* werden zunächst nur die Standardobjekte erstellt. Diese können im Nachgang angepasst werden (Ändern der Kommandozeilenoption). **Alle zusätzlichen Varianten müssen komplett manuell erstellt werden!**

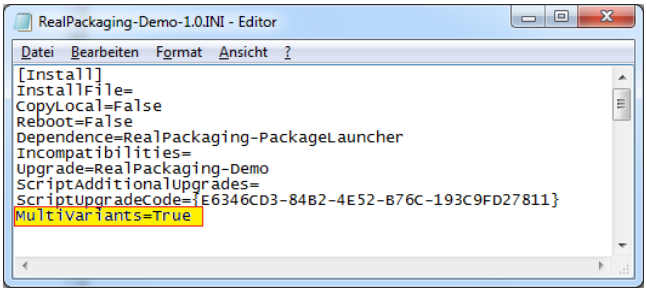
7.1.2 Instanzvarianten

Die *Varianten*gestaltung ist vermutlich eher die Ausnahme als die Regel in der Softwarepaketierung. Dennoch ergeben sich so zum Teil elegantere Realisierungsmöglichkeiten, als über mehrere, einzelne Softwarepakete. Die soeben beschriebene *Standardvarianten*-implementation geht von einer Entweder-Oder-Installation aus. Das bedeutet, dass auf einem Client entweder die eine oder die andere Ausprägung (*Variante*) zum Einsatz kommen soll.

Ist es erforderlich, dass zwei *Varianten* parallel installierbar sein sollen, so kann dies über *Instanzvarianten* realisiert werden. Diese werden im Gegensatz zu den regulären *Varianten* durch den *Package-Launcher* separat und eigenständig registriert.

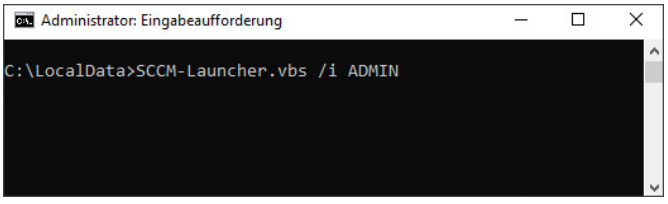


1. Implementieren Sie für diesen Zweck einfach den Bezeichner *MultiVariants=True* in der *Install*-Sektion der *INI*-Datei:

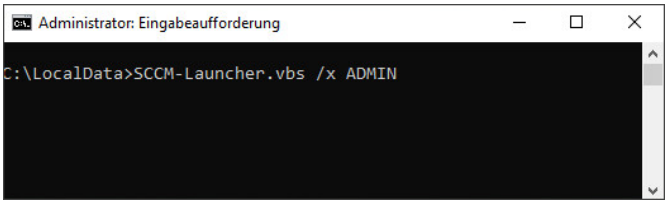


2. Ändern Sie die *Properties* über die *Paket-INI*-Datei oder verwenden Sie eine *Pre/PostInstall* für die *Variante*nabbildung gemäss dem Vorgehen aus letztem Kapitel.

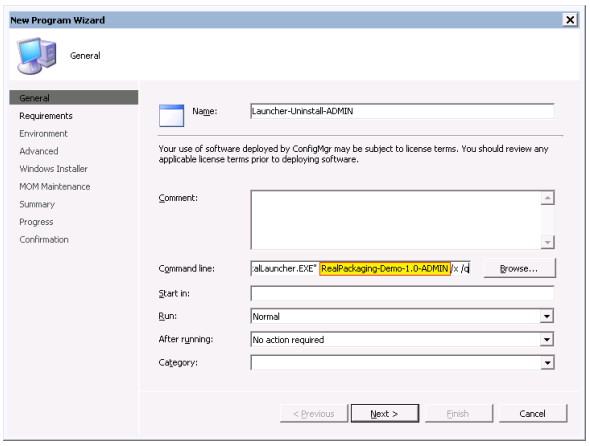
3. Installieren Sie die *Instanzvariante* wie gewohnt mit einer Kommandozeilenerweiterung:



4. Für die Deinstallation verwenden Sie für eigene Tests die folgende Möglichkeit:



5. Für die *SCCM*-Integration ist aber bei der Deinstallation der *Instanzvariante* auf folgende Kommandozeile zurückzugreifen: *"%WINDIR%\Package-Launcher\Bin\LocalLauncher.EXE" RealPackaging-Demo-1.0-**ADMIN** /x /q*



7.1.3 Regionsspezifische Varianten

Für Anpassungen und Varianten, die sich auf Regionen beziehen, sind drei Umsetzungsmöglichkeiten vorgesehen:

1. variable *MST*-Dateien
2. unterschiedliche *Windows Installer Properties* über *Sectionproperties* in der INI-Datei
3. variable *Gestaltung über Scripts* oder *Pre-* und *PostInst_00x.wse* nach Auswertung der zum Installationszeitpunkt verfügbaren Variablen *%VARIANTSECTION%*

7.1.3.1 Regionsspezifische Varianten über MST Dateien

Damit dieser Mechanismus funktioniert, ist die Einhaltung der Regeln für die Namensgebung der *MST*-Dateien zwingend erforderlich.

Regionsspezifische *MST*-Dateien müssen mit dem Schlüsselwort „DEP." beginnen, gefolgt von einem zweistelligen Ziffern- oder Buchstabencode:

D	E	P	.	x	x	.	M	S	T

Für den zweistelligen Ziffern- oder Buchstabencode ist folgendes Vorgehen zu wählen.

1. Ist die *MST*-Datei für eine **spezifische Organisationseinheit** bestimmt, dann ist der Buchstabencode der entsprechenden Organisationseinheit zu verwenden (Bspl. „EE" für Generalsekretariat in der Erziehungsdirektion)
2. Ist die *MST*-Datei für eine **übergeordnete Organisationsgruppe** bestimmt, dann ist als erster Buchstabe der Code der Organisationsgruppe zu verwenden, als zweiter Buchstabe folgt dann die Ziffer „9" (Bspl. „E9" für die gesamte Erziehungsdirektion)
3. Soll die *MST* Datei überall dort angewendet werden, wo keine spezifisch Organisations-Transformation vorliegt, dann ist der Zifferncode „99" zu verwenden.

Eine Liste der Organisationskürzel wird in einem separaten Dokument dokumentiert. Der *Package-Launcher* liest den Referenzcode, also die Zugehörigkeit des Clients, aus *dem* Registrykey *HKLM\SOFTWARE\Real Packaging\Package-Launcher**DepAgency** ein.

Achtung: Die Anwendung der regionsspezifischen *MST*-Datei erfolgt in der oben aufgezählten Reihenfolge (*Organisations**einheits**-MST* gewinnt bei Übereinstimmung vor der *Organisations-**gruppen**-MST* und diese wiederum vor der allgemeinen *MST* „99“).

Es werden alle im Paketverzeichnis vorliegenden nichtregionenspezifischen *MST*-Dateien angewendet und - sofern vorhanden - maximal eine regionenspezifische *MST*-Datei.

7.1.3.2 Hierarchien

Insgesamt gibt es drei Hierarchien in denen solche *MST*-Dateien appliziert werden:

1. Unternehmensübergreifende *MST*-Dateien: DEP.99.MST
2. Organisationsgruppe: DEP.#9.MST (bspl. DEP.E9.MST)
3. Organisationseinheits-*MST*-Dateien: DEP.##.MST (bspl. DEP.EE.MST)

Das Zeichen ‚#‘ steht für ein beliebiges Zeichen gemäss Liste aus dem Anhang der Paketierungs-dokumentation.

7.1.3.3 Eindeutigkeit

Es hat sich bewährt, dass jede *MST*-Datei das komplette Set an Einstellungen enthält, inkl. die durch *AddProperties.vbs* erstellten Standardproperties. Zum Erstellen der *MST*-Datei wird das Umbenennen der Datei *STANDARD.MST* zu *DEP.99.MST* empfohlen. Anschliessend wird das Kopieren und anschliessende Umbenennen der Datei *"DEP.99.MST – Kopie"* zu einer Datei in der Form *DEP.#9.MST* oder *DEP.##.MST* empfohlen.

Die sich unterscheidenden Einstellungen können dann in der neuen *MST*-Datei abgeändert werden. Während des Lebenszyklus eines Softwarepakets ist auch das folgende Vorgehen denkbar (Beispiel):

1. Die Initialversion eines Softwarepakets, wo eine Lizenzproperty zum Einsatz kommt, wird in der *Rev. 001* in der Datei *STANDARD.MST* implementiert.

2. Nach Verbreitung des Softwarepakets melden sich weitere Organisationsgruppen mit abweichendem Lizenzschlüssel. Jetzt kann der Softwarepaketierer die Datei *STANDARD.MST* auf *DEP.99.MST* umbenennen. Für die in dieser *MST*-Datei platzierten Lizenzschlüssel abweichenden Direktionen werden Kopien der Datei *DEP.99.MST* erstellt, diese mit den richtigen Namen benannt (bspl. *DEP.EE.MST*, *DEP.E9.MST*) und dort die Lizenzproperty abgeändert. → Dieses Verfahren hat keinen negativen Einfluss auf bestehende Installationen, sondern wird sich nur für Neuinstallationen auswirken.

3. Die wichtigste Vorkehrung ist das Erhöhen der *Build-Datei* um einen Zähler, um die in dieser *KeepRevision* vorgenommenen Änderungen auf dem System nachvollziehen zu können!

4. Das Paket neu überführen.

Wichtig

Der Package-Launcher **wendet** auch bei Vorliegen mehrerer *DEP.*.MST* Dateien **immer nur eine einzelne *DEP.*.MST* an!** Er sucht sich die für ihn in der lokalen Umgebung (*DepAgency*) am besten passende *MST*-Datei aus.

7.1.3.4 Wahl der passenden Hierarchie

Ausschlaggebend für die Gestaltung des Region-Bezeichners bei der *MST*-Datei (bspl. ‚EE' oder ‚99') und für die Wahl der *[Section]* (bspl. '[99-STANDARD]') in der Paket-*INI*-Datei ist **die höchste Hierarchie, in der die gelieferten Einstellungen Gültigkeit besitzen kann** und aus Sicht des Softwarepaketierers dort **technisch anwendbar** ist. Dabei gibt es im Handling keinen Unterschied zwischen der Art der Einstellungen, also, ob es sich bei den Einstellungen um Lizenzschlüssel, um Einstellungsoptionen, Serverangaben, DFS-Informationen oder Featureselektionen handelt.

Die im Auftrag angegebene Organisationseinheit muss also nicht zwingend seine eigene *Organisationseinheits-MST* erhalten! Auch an Lizenzkeys sieht man äusserlich selten an, ob diese nur für eine bestimmte Organisationseinheit gelten, oder nicht.

Aus diesen Überlegungen ergibt sich, dass beim Erstauftrag vielfach auch Einstellungen, die für die Umsetzung eines Auftrages einer Organisationseinheit vorgesehen sind, in der *STANDARD.MST* oder *DEP.99.MST* appliziert werden – zumindest sofern die Einstellungen allgemein gültigen Charakter aufweisen. Kommen später weitere Organisationseinheiten dazu, welche abgeänderte Einstellungen erfordern, können dort entsprechende *Organisationseinheits-MST's* oder -Bezeichner in der INI angewendet werden.

7.1.3.5 Beispiele

Ausgangslage (Organisationsgruppen-MST)

Der Softwarepaketierer erhält einen Auftrag von einer Organisationsgruppe mit Lizenz-informationen, die einen lokalen Flexnet-Server verwenden. Der Flexnet-Server ist aus einer anderen Umgebung nicht erreichbar, sondern nur aus der beauftragenden Organisationsgruppe. Es existiert noch kein gleiches oder ähnliches Paket. Und es sind keine weiteren Informationen vorhanden, dass noch eine weitere Organisationseinheit diese Software benötigen wird.

Realisierung

Der Softwarepaketierer erstellt eine Datei ***STANDARD.MST*** mit allen Standardproperties (*AddProperties*.vbs) und benennt diese zu ***DEP.99.MST*** um. Diese Datei kopiert er und erstellt

eine *Organisationsgruppen-MST*-Datei ***DEP.E9.MST***[1)], wo er alle speziellen Direktionseinstellungen appliziert. Die Property des Servers wird in der INI-Datei unter *[E9-STANDARD]* angegeben.

Ausprägung

DEP.99.MST; DEP.E9.MST; INI

Ausgangslage (unternehmensübergreifende *MST*)

Der Softwarepaketierer erhält einen neuen Paketierungsauftrag. In den vorzunehmenden Einstellungen wird auf keine Shares und Server verwiesen. Technisch lässt sich das Paket überall installieren. Es sind keine Informationen vorhanden, die den Gebrauch oder Installation irgendwie einschränken würden. Das Paket existiert noch nicht.

Realisierung

Der Softwarepaketierer erstellt mit *AddProperties.vbs* eine Datei ***STANDARD.MST*** mit allen Standardproperties. In diese Datei appliziert er zusätzlich alle vorzunehmenden Einstellungen.

Ausprägung

STANDARD.MST

Ausgangslage (Organisationseinheits-*MST*)

Ein bestehendes Paket, welches bereits eine ***STANDARD.MST***-Datei enthält benötigt Anpassungen für eine spezielle Organisationseinheit. Diese Einstellungen sollen nicht für den Rest der Organisationseinheiten und auch nicht für andere Organisationsgruppen gelten.

Realisierung

Die bestehende Datei ***STANDARD.MST*** wird auf ***DEP.99.MST*** umbenannt. Zusätzlich wird eine Kopie derselben erstellt, welche zu ***DEP.EE.MST***[1)] umbenannt wird. In dieser neuen ***DEP.EE.MST*** können die verlangten Modifikationen vorgenommen werden. Zusätzlich platziert der Softwarepaketierer eine *Build-Datei* in demselben Verzeichnis, wo er die *Build*nummer um 001 erhöht.

Ausprägung

DEP.99.MST, DEP.EE.MST

1) Die Bezeichner EE und E9 sind nur Beispiele und hängen massgeblich von der Organisationsstruktur der Unternehmung ab, sowie deren Zuweisung zu entsprechenden Kürzeln

7.1.3.6 Einstellungen in der INI-Datei, anstatt in der MST-Datei (Section Properties)

Können Einstellungsunterschiede in einigen wenigen Properties abgebildet werden und sind neben der unterschiedlichen Ausprägung dieser Properties keine anderweitigen Einstellungsunterschiede zu applizieren, so ist der Verwendung von Properties in der *INI*-Datei des Softwarepakets den Vorzug zu geben. Auch schnell wechselnde Einstellungen, wie Pfadverweise,

Serverangaben, IP-Adressen, etc. sind nach Möglichkeit in der entsprechenden *[Section]* in der *INI*-Datei zu applizieren. Dies auch, wenn schon eine entsprechende *MST*-Datei vorliegend ist.

In der INI-Datei sind die zu unterscheidenden Properties über unterschiedliche *Sectionproperties* zu implementieren. Sofern keine Durchmischung mit *Standardvarianten* oder *Instanzvarianten* vorgesehen ist, sieht die Namensbezeichnung folgendermassen aus (Beispiel):

[99-STANDARD]	Die darunter befindlichen Properties werden für die *Standardvariante* unternehmensweit appliziert, sofern keine *Organisationseinheits- oder Organisationsgruppen-Variante* mit abweichenden Einstellungen gefunden wird.
[E9-STANDARD]	Die darunter befindlichen Properties werden für die *Standardvariante* in der Erziehungsdirektion appliziert, sofern keine *Organisations-einheitsvariante* mit abweichenden Einstellungen gefunden wird.
[EE-STANDARD]	Die darunter befindlichen Properties werden für die *Standardvariante* im Generalsekretariat der Erziehungsdirektion appliziert.

7.1.3.7 Verwendung in Scripts

In allen Arten von im Paket verwendeten Scripts (cmd, ps1, vbs, etc.) kann auf den INI-Bezeichner der Variantsection und auf die Variante via %VARIANT% zugegriffen werden, da dieser zum Installationszeitpunkt als Environmentvariable zur Verfügung steht. So sind bedingte Anweisungen in Scripts durch die Ausprägung der Umgebungsvariablen %VARIANTSECTION% möglich.

Beispiel:

```
IF "%VARIANTSECTION%"=="E9-STANDARD" GOTO ERZ
```

Die in der *VARIANTSECTION* in der INI-Datei platzierten Properties lassen sich in Scripts direkt auswerten:

```
[99-STANDARD]
SERIALNUMBER=1111-2222-3333-4444
```

ergibt: eine Environmentvariable *PL_SERIALNUMBER=1111-2222-3333-4444*

Auf diese kann in Scripts direkt zugegriffen werden (Beispiel):

```
%SOURCE%\Activator.exe SERIAL=%PLSERIALNUMBER%
```

7.2 Package-Launcher Restart Manager

Der *Package-Launcher Restart Manager* verwaltet Neustarts, die von der Installation von Softwarepaketen oder nach Microsoft Updates gefordert werden. Im abgemeldeten Zustand übernimmt der *Restart Manager Service* die Kontrolle. In allen anderen Fällen ist der Benutzer via *Restart Manager UI* zum Auslösen des Neustarts verantwortlich. Der Benutzer ist auch dafür verantwortlich, dass er seine Dokumente ordentlich gespeichert hat, bevor er den Neustart auslöst.

Folgende Ereignisse lösen die Neustart- und Anmeldeverwaltung aus:

1. Nach der Installation eines Softwarepakets wird ein Neustart verlangt, damit dieses fehlerfrei funktioniert (Reboot=True in Softwarepaket, Logoff=True in Softwarepaket od. Rückgabewert 3010 (ERROR_SUCCESS_REBOOT_REQUIRED) aus der Installation)
2. Ausstehende Neustarts nach der Installation von Microsoft Updates verlangen einen Neustart
3. Ausstehende Neustarts nach der Installation und vor der Applizierung von securityrelevanten Patches verlangen einen Neustart
4. Es wurde über eine längere Zeit (grösser 14 Tage) am Computer kein Neustart durchgeführt. Achtung: In einigen Fällen ist Computer herunterfahren und Computer starten nicht äquivalent einem Neustart! Bspl. Fast Startup eingeschaltet)

7.2.1 Restart Manager Service

Der *Restart Manager Service* ist für die Verwaltung von Neustarts zuständig, die durch ein der oben dokumentierten Ereignisse angefordert wurden. Ein automatischer Neustart durch den *Restart Manager Service* erfolgt nur, wenn niemand am Computer arbeitet (abgemeldeter Client auf dem keine Remotesitzungen geöffnet sind) und es sich beim System nicht um einen Server handelt. Server werden durch den *Package-Launcher* nie neu gestartet!

Der *Restart Manager Service* prüft einmal pro Minute, ob der Registrykeys *HKLM\Software\Real Packaging\Package-Launcher\MakeReboot* einen Zeitstempel enthält oder ob die anderen 2 Ereignisse zutreffen (Ausstehende Neustarts, System lange nicht neu gestartet). Sollte eine oder mehere dieser Bedingungen zutreffen, wird nach 5 minütiger Wartezeit das Gerät (Server ausgeschlossen) automatisch neu gestartet, sofern im Hintergrund keine Installation ausgeführt wird. Sollte *LocalLauncher.EXE* oder *msiexec.exe* noch Installationstätigkeiten vornehmen, wird mit einer Frequenz von fünf Minuten erneut geprüft, ob die Installationstätigkeit abgeschlossen ist und der Neustart wird dann im Anschluss ausgeführt.

Der *Restart Manager Service* löscht die Registrykeys *MakeReboot* und *MakeLogoff* bei jedem Neustart des Geräts, d.h. wenn dieser vom Service selbst ausgelöst, aber auch, wenn dieser vom Benutzer ausgeführt wird.

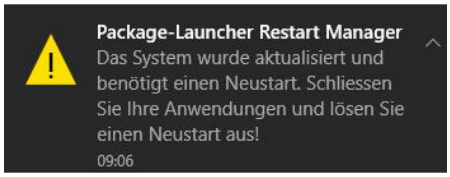
7.2.2 Restart Manager UI

Das *Restart Manager User Interface* übernimmt die Kommunikation von anstehenden Neustarts mit dem Benutzer. Diese Kommunikation läuft über *BallonTips* am rechten Seitenrand des Computerbildschirms. Der Benutzer ist dafür verantwortlich, dass er seine Dokumente gespeichert hat, bevor er den Neustart auslöst.

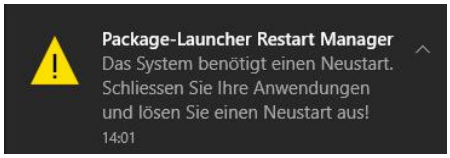
7.2.2.1 Neustartmeldung

Alle Ereignisse lösen in den ersten 48 Stunden (einstellbar) lediglich eine diskrete Benachrichtigung aus. Diese Benachrichtigung erscheint alle 2 Stunden. Folgende Meldung erscheint bei anstehendem Neustart:

Benachrichtigungsbeispiel nach Softwareinstallation:



Benachrichtigungsbeispiel bei Ereignissen ausstehender Neustart oder Pending Reboots:

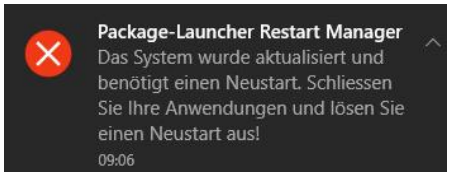


Zudem finden wir in der Taskleiste das folgende minimierte Icon:

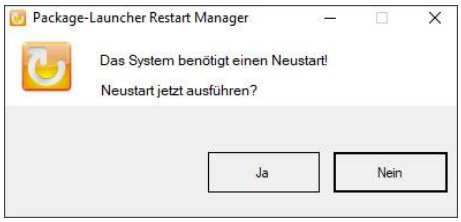
Darüber lässt sich vom Benutzer jederzeit der angeforderte Neustart ausführen. Es ist aber egal, ob letztlich der Neustart am System über das Startmenü ausgelöst wird oder die Funktion aus dem *Restart Manager* Dialog verwendet wird.

Nach den 48 Stunden humaner Anzeige, wird die Anzeigewiederholung auf einmal pro Stunde (bei ausstehenden Neustarts oder Pending Reboots), bzw. alle 10 Minuten (Neustartanforderung nach Softwareinstallation) erhöht und das Neustartfenster wird dann jedes Mal zentriert angezeigt:

Benachrichtigungsbeispiel nach Softwareinstallation:

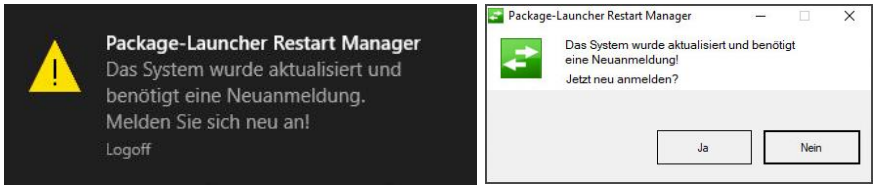


Anzeigebeispiel bei Ereignissen ausstehender Neustart oder Pending Reboots:



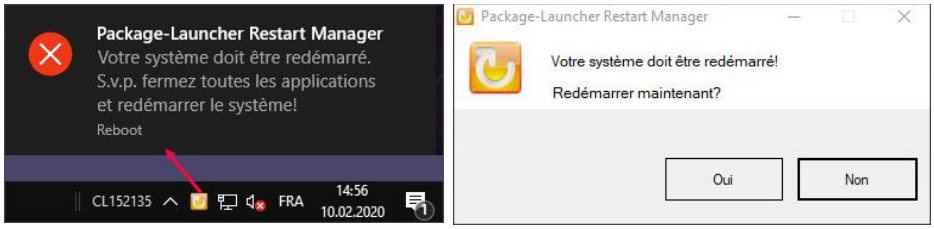
7.2.2.2 Neuanmeldemeldung

Folgende Meldung erscheint bei anstehender Neuanmeldung nach Softwareinstallation:

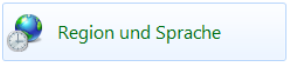


7.2.2.3 Sprachausprägungen

Der *Package-Launcher Restart Manager* unterstützt drei Sprachen – Deutsch, English und Französisch:

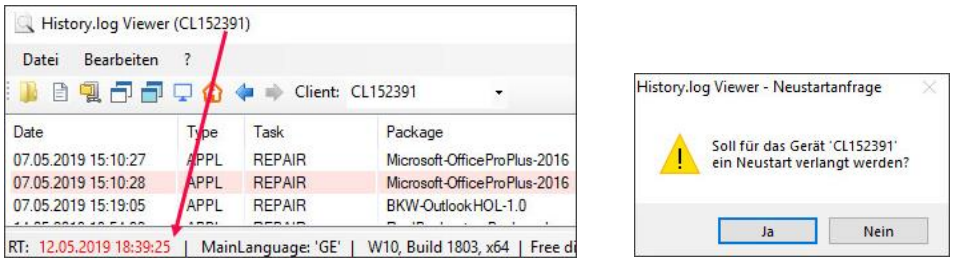


Die Sprache kann über die Systemsteuerung umgestellt werden und entspricht der Systemsprache die der aktuelle Benutzer gewählt hat.



7.2.2.4 Triggern über History.LOG Viewer

Ausstehende Neustarts sind zudem remote mit dem *History.LOG Viewer* einsehbar und es kann per Doppelklick darauf auf dem Remotecomputer das obige Fenster zur sofortigen Anzeige erzwungen werden (sofern der *Restart Manager* dort nicht schon gestartet ist).

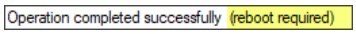


7.2.2.5 Automatische Neustarts

Ist kein Benutzer am System angemeldet, wird nach einer Wartezeit von 5 Minuten das System automatisch neu gestartet.

7.2.2.6 Aktionen auf Server

Auf einem Server wird kein automatischer Neustart durchgeführt und es wird dort auch nie ein Dialog durch den *Restart Manager* angezeigt. Der Serververantwortliche ist selbst für Neustarts verantwortlich. Neustanforderungen durch Paketinstallationen können aus dem *History.LOG* herausgelesen werden.

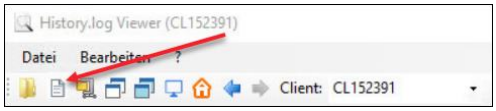


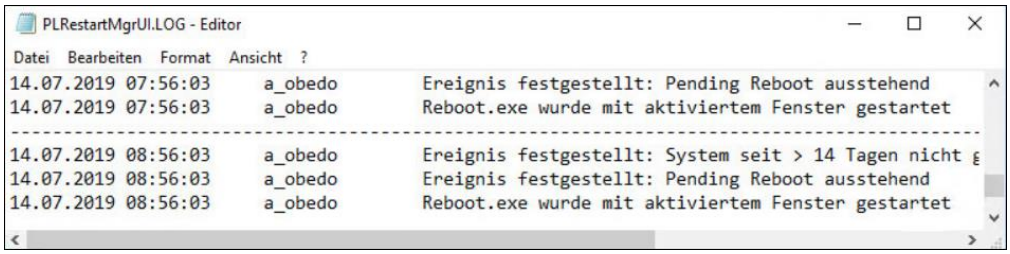
Ereignisse:

Ereignis	Darstellung erste 48h	rerun	Darstellung danach	Wiederholung danach
Pending Reboot (nach Systemupdates, etc.)	Fenster minimiert, BallonTip	alle 2h	Fenster wird zentriert angezeigt	Jede Stunde einmal
14 Tage System nicht neu gestartet	Fenster minimiert, BallonTip	alle 2h	Fenster wird zentriert angezeigt	Jede Stunde einmal
Anwendungsinstallation benötigt Neustart	Fenster minimiert, BallonTip	alle 2h	Fenster wird zentriert angezeigt	Alle 10 Minuten einmal

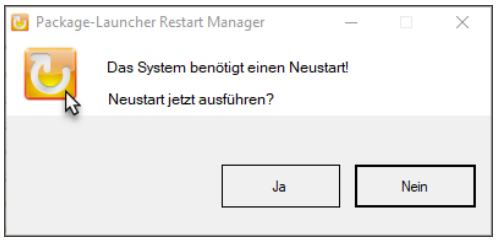
7.2.2.7 Protokollierung

Über den *History.LOG Viewer* ist eine Protokolldatei einsehbar, die weitere Informationen über die Ereignisse des *Restart Managers* aufzeigt:





Ein Klick auf das gelbe Icon im Fenster des *Restart Managers* öffnet ebenfalls die obige Protokolldatei..



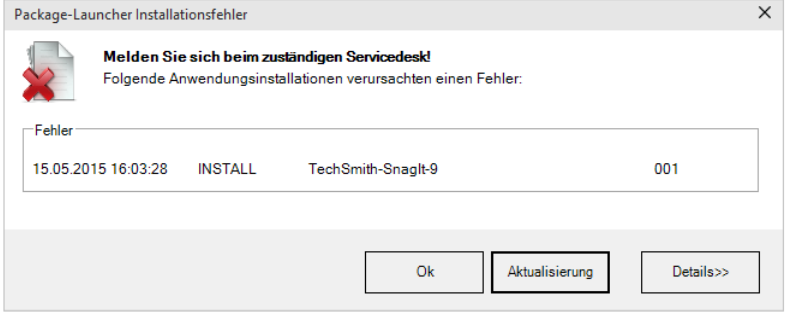
7.2.2.8 Registrykeys

Alle Registrykeys befinden sich unter *HKLM\Software\Real Packaging\Package-Launcher.* Diese Registrykeys können nach Bedarf über *GPO* geändert werden.

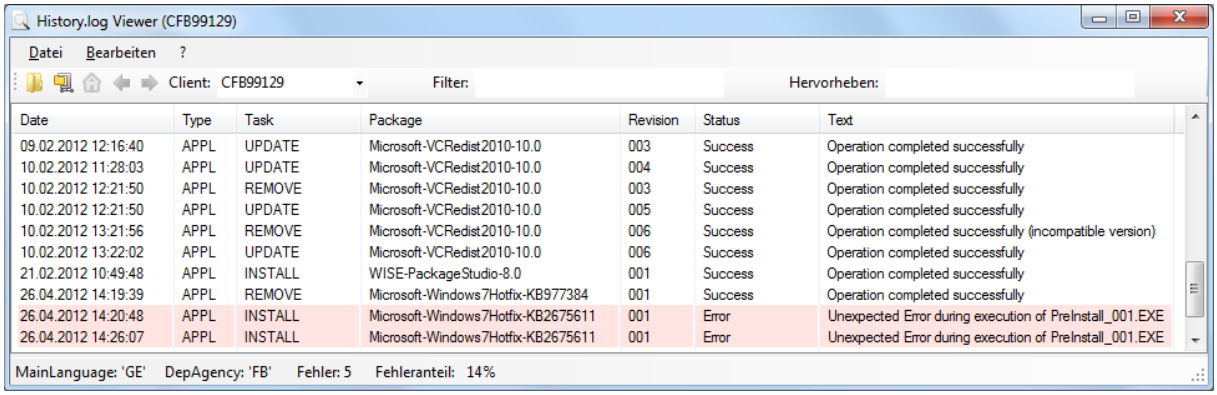
RestartManagerUIBehavior	**aggressive**¦extreme Mit diesem Key wird das Verhalten des zusätzlichen Benutzerdialogs gesteuert
RestartManagerUITimer	Zahl (**600000**) Frequenz in Millisekunden mit welcher die Prüfung auf die Registrykeys *MakeReboot* und *MakeLogoff* erfolgen soll und die *BalloonTips* angezeigt werden (*Soft-Warning* 6 x länger). Standardmässig sind 10 Minuten für die kleinsten Wiederholungseinheiten eingestellt (600'000). Siehe dazu die Tabelle *Ereignisse* aus dem letzten Kapitel.
RMSoftWarningInHour	Zahl (**48**) Wechsel in Stunden nachdem von *Soft-Warnings* zu *Strong-Warnings* gewechselt werden soll. Nach 12 Stunden erscheinen nur noch *Strong-Warnings* in kürzerer Frequenz.
ExtendedRebootSupport	**True**|False Mit ExtendedRebootSupport=False kann eine Neustart-aufforderung in Fällen «letzter Neustart > 14 Tage» und nach ausstehenden Neustarts nach Microsoft Updates und dergleichen, abgestellt werden (nicht empfohlen).
CountOfDayToWaitForReboot	Zahl (**14**) Anzahl Tage, nach der ohne Benutzerneustarts die Aufforderung zum Neustart über *PLRestartMgrUI* angezeigt wird.

7.3 Package-Launcher Error Wizzard

Der *Package-Launcher Error Wizzard* soll dem Benutzer oder der für das initiale Geräteaufsetzen verantwortlichen Person, allfällige Fehlermeldungen, die durch das Installieren entstanden sind, aufzeigen. Nach dem Anmelden erscheint **einmalig** pro Benutzer bei einem im *History.LOG* ausgewiesenen Fehler, die Anzeige des *Package-Launcher Error Wizzards* als *modalen Dialog* (möglicherweise ist in Ihrem Unternehmen diese Option ausgeschaltet):



Mit einem Klick auf *Ok* oder *Details>>* wird die Kenntnisnahme bestätigt und das Fenster verschwindet. Der Klick auf *Details>>* startet den lokalen *History.LOG Viewer,* um weitere Informationen einzuholen.



Die Anzeige im *Package-Launcher Error Wizzard* ist sprachabhängig (Deutsch, Französisch, Englisch). Die Sprache wird hierbei durch die aktuellen Benutzereinstellungen gesteuert.

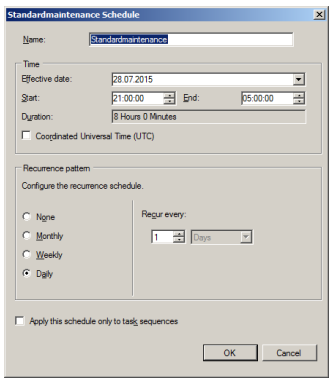
Damit der *Package-Launcher Error Wizzard* funktioniert, muss dieser über das Paket *RealPackaging-PackageLauncher-2020* installiert werden. Insbesondere ist es erforderlich, dass die Berechtigungen des Registrykeys *HKLM\SOFTWARE\Real Packaging\Package-Launcher**Display Error Wizzard** für Schreibzugriffe des Benutzers geöffnet sind (erledigt durch Paketinstallation).

7.3.1 Dauerhaftes Einschalten des Package-Launcher Error Wizzard

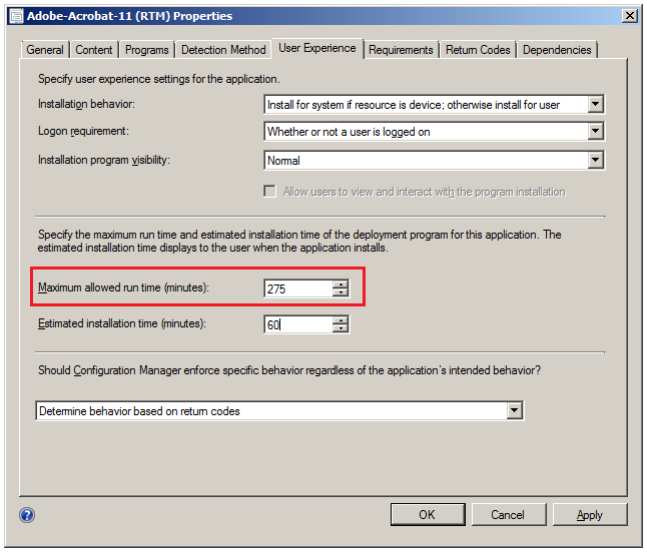
Um den *Package-Launcher Error Wizzard* anzuweisen, dass dieser immer erscheinen soll, wenn ein Fehler in der Installation entdeckt wird (also nicht nur bei der Erstanmeldung, sondern dauerhaft während Betriebszeiten), kann der Registrykeys *HKLM\SOFTWARE\Real Packaging\Package-Launcher\Display Error Wizzard**DisplayErrors** auf *True* gesetzt werden.

7.4 SCCM-Wartungsfenster

Im *Configuration Manager* können in *Collections* spezielle Wartungsfenster definiert werden, um Installationen in die Nacht oder in Randzeiten zu verlegen, so dass diese zu Zeiten durchgeführt werden, wo der Benutzer nicht arbeitet, bzw. keine Programme geöffnet sind.

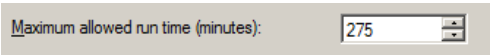


Hier kommt die maximal benötigte Zeit für die Installation der *Applications* ins Spiel. Im *Deployment Type* der *Application* kann die maximale Zeit definiert werden, die für die Installation benötigt wird. Hier handelt es sich um die Zeit für die betroffene *Application*, ohne deren Abhängigkeiten.



Wird eine Application nun über eine Collection mit *Wartungsfenster* zugewiesen, ist darauf zu achten, dass die Summe der maximalen Zeiten der Zielapplication, **inkl. deren erforderlichen Abhängigkeiten**, das über das Wartungsfenster zur Verfügung gestellte Zeitfenster nicht überschreitet. Wird der Wert überschritten, kann die Application nicht installiert/deinstalliert werden!

Aus dieser Ausgangslage ergibt sich beim Einsatz von *Wartungsfenster* schnell mal das Bedürfnis, möglichst kleine Werte im *Deployment Type* anzugeben.



Wollen Sie *SCCM Wartungsfenster* einsetzen, kann mit dem Paket-INI-Bezeichner *MaxExecuteTimeRTM* ein kleinerer Wert, als vorgesehen vorgegeben werden. Die Spannbreite bewegt sich zwischen 15 – 720 Minuten. Wählen Sie aber keinen zu kleinen Wert, da sonst das Deployment während der Installation abgebrochen würde. Folgendes Beispiel veranschaulicht den Einsatz:

```
[Install]
MaxExecuteTimeRTM=60
```