PBE

Parallel Batch Engine

Stand: Framework Studio 4.2

Inhalt

Einleitung

Die **P**arallel **B**atch **E**ngine ist eine Routine, die es ermöglicht eine Batch-Verarbeitung zu automatisieren. Neben der Ausführung normaler Batch-Verarbeitung ist sie dafür optimiert, die FSConsole.exe von Framework Studio zu steuern und so Compile-Läufe, Package-Exporte, Package-Exporte und Publish-Aktionen auszuführen. Weitere Kern-Funktionen sind die parallele Verarbeitung und das erzeugen und Archivieren einer Log-Datei.

Ausgeliefert wird diese Routine zusammen mit dem Framework Studio AddIn-Paket. Dieses kann vom bekannten FTP-Server heruntergeladen werden.

Arbeitsweise

Das Programm besteht aus folgenden Dateien:

Datei	Bedeutung
PBE.exe	Das eigentliche Programm
PBE.xml	hier wird die Konfiguration vorgenommen
PBE.dtd	beinhaltet die Schema-Definition für die PBE.xml
LogTemplate.htm	Vorlage für die Protokoll-Datei

Wenn die Konfiguration in der Datei PBE.xml vorgenommen wurde, dann kann die Routine durch ausführen der Datei PBE.exe gestartet werden. Es wird die XML-Datei ausgewertet und die darin konfigurierten Schritte werden alle abgearbeitet. Dabei wird ein Protokoll in Form einer HTML-Datei erzeugt. Dieses beinhaltet neben den Informationen, wann welche Aktion gestartet wurde und wie lange sie gedauert hat, auch das während der jeweiligen Aktion erzeugte Protokoll. Das Protokoll wird während der Ausführung regelmäßig aktualisiert. So kann man sich auch ein Bild vom Fortschritt der Aktion machen.

PBE ist dafür konzipiert in einem Lauf mit mehreren Versionen von Framework Studio parallel zu arbeiten. Unterstützt werden alle Versionen ab Framework Studio 3.2.

Konfiguration

Die Konfiguration erfolgt mithilfe der Datei PBE.xml. Diese wird am besten mit Visual Studio bearbeitet, weil es dort eine IntelliSense-Unterstützung und eine Validierung der Eingaben gibt.

Programm-Verzeichnisse

Für die unterschiedlichen FS-Versionen werden die Programm-Verzeichnisse ermittelt. Dabei geht die Routine für jede FS-Version mit den folgenden Prioritäten vor und verwendet das Verzeichnis, welches zuerst gefunden wird:

Prio 1.) Konfiguration in der XML-Datei

Beispiel:

```
<FSVersions>
  <FSVersion FS="3.7.0.0" Dir="C:\Programme\Framework Systems\FrameworkStudio 3.7.3"/>
  <FSVersion FS="3.8.0.0" Dir="C:\Programme\Framework Systems\FrameworkStudio 3.8"/>
</FSVersions>
```

Prio 2.) Order „C:\FS\Framework Studio X.Y.0.0\“

das ist der Standard im Haus von Nissen & Velten.

Prio 3.) Standard Installations-Verzeichnis

„%ProgramFiles%\Framework Systems\Framwork Studio X.Y“

Dabei werden auch Bugfix/ServiceRelease- und Beta-Versionen erkannt.

Parameter

Für eine einfachere Konfiguration wird mit Parametern gearbeitet. Diese werden am Anfang der Datei angegeben. Unterhalb des XML-Knoten <Params> werden die Parameter mit <Param>-Knoten aufgelistet. Diese Parameter können bei den Aktionen verwendet werden. Dazu müssen sie in geschweiften Klammern geschrieben werden: {parameter}

```
  <Params>
    <Param Name="ExportDir" Value="E:\temp\export123\"/>
    <Param Name="rep" Value="\ConnectionType SqlServer \Server NV261 \Database FSDemo37
\DBUser sa \DBPassword sql2005"/>
  </Params>

  <Sequence>
    <CompileRun FS="3.7.0.0" Rep="{rep}" Run="1"/>
  </Sequence>
```

Die im Folgenden aufgelisteten Parameter sind immer gefüllt:

Parameter	Bedeutung
ExportDir	Beinhaltet einen Ordner, in dem die Dateien der Package-Exporte abgelegt werden. Die Package-Importe werden ebenfalls in diesen Ordner gesucht. Dieser Parameter wird bei den Aktionen <Export> und <Import> automatisch verwendet.
ApprovedExportDir	Beinhaltet einen Ordner, in dem die Dateien der Approved Package-Exporte abgelegt werden. Dieser Parameter wird bei der Aktion <ApprovedExport> automatisch verwendet.
ApprovedImportDir	Beinhaltet einen Ordner, in dem die Dateien abgelegt werden, die mit der Aktion <ApprovedImport> verbarbeitet werden sollen.
ApprovedHistoryDir	Beinhaltet einen Ordner, in den die Dateien verschoben werden, nachdem sie mit der Aktion <ApprovedImport> verbarbeitet wurden.
Date	Start-Datum im Format "yyyyMMdd" z.B. 20140526
DateTime	Start-Zeit im Format "yyyyMMdd-HHmmss" z.B. 20140526-142457
DateTimeText	Start-Zeit im Format "yyyy-MM-dd HH:mm (dddd)" z.B. 2014-05-26 14:24 (Montag)
ExportFilePrefix	Start-Datum mit dem Format "yyyy-MM-dd_" z.B. 2014-05-26_ Dieses Präfix wird vor den Dateinamen der Export-Dateien gestellt. So haben die Dateinamen dasselbe Format wie es auch der Package-Manager beim Export vorschlägt. z.B. 2014-05-26_eNVenta_3.4.db
Weekday	Aktueller Wochentag in Deutsch: Mo, Di, Mi, Do, Fr, Sa, So Kann z.B. in einer <Condition> verwendet werden um wöchentliche Aktionen zu definieren.
Title	Hat standardmäßig den Wert "Nachtlauf {DateTimeText}". Dieser Parameter kann bei Bedarf überdefiniert werden.
Machine	Name des Rechners (Environment.MachineName)

Darüber hinaus können auch eigene Parameter definiert werden. So können z.B. die verwendeten Repository-Connections an zentraler Stelle definiert werden.

Bei Parametern kann auch auf vorher definierte Parameter verwiesen werden – wie z.B. bei dem vordefinierten Parameter „Title“.

Organisation

Die Aktionen werden in der XML-Datei als Sequenzen oder Parallel-Verarbeitungen organisiert. Die unterschiedlichen Knoten können beliebig ineinander verschachtelt werden. Einzige Ausnahme ist der oberste Knoten – dieser muss immer <Sequence> sein.

Jeder Knoten kann Optional ein Attribut „Name“ erhalten. Dieser Name wird in der Protokoll-Datei ausgegeben.

Knoten	Beschreibung
<Sequence>	Alle darunter aufgeführten Aktionen werden nacheinander verarbeitet. In der Protokoll-Datei werden die Einträge untereinander ausgegeben. Attribute • **Name**: optional
<Parallel>	Alle darunter definierten Aktionen werden parallel verarbeitet. In der Protokoll-Datei werden die Einträge nebeneinander ausgegeben. Attribute • **Name**: optional • **MaxTasks**: optional - gibt die Anzahl der maximal parallel ausgeführten Aktionen an. Bei einer sehr langen Liste an Aktionen macht es Sinn, die Parallelität z.B. auf 4 zu begrenzen. ACHTUNG!! Wenn dieses Attribut nicht angegeben ist, erfolgt die Verarbeitung komplett parallel – egal wie viele Aktionen definiert wurden.

<Condition>	Arbeitet wie eine <Sequence> - die darunter aufgeführten Aktionen werden nacheinander ausgeführt. Die Ausführung erfolgt aber nur dann, wenn die beiden Attribute „Value“ und „Equals“ denselben Wert haben.
	Attribute
	• **Name**: optional
	• **Value**: erforderlich – gibt den linken Wert für den Vergleich an. Üblicherweise wird hier ein Parameter – z.B. "{Weekday}" – angegeben.
	• **Equals**: erforderlich – gibt den Wert an, mit dem verglichen werden soll – z.B. "So".
	
	Im Value kann eine Funktion "#EXISTS(<dateipfad>)" verwendet werden. Diese liefert den Wert "True" oder "False". Der Dateipfad kann auch Parameter beinhalten.
	Beispiel
	`<Condition Name="Prüfen ob Publish2Go existiert"`
	`Value="#Exists({InputDir}\sqlitedb.p2go)"`
	`Equals="True">`

<ImportQueue>	Damit können die Package-Importe so organisiert werden, dass diese parallel zu anderen Aktionen durchgeführt werden können. Die Export-Aktionen packen, wenn sie fertig sind, entsprechende Import-Aktionen in diese Queue. Diese wird dann sofort mit der Abarbeitung beginnen. Dabei werden aber alle Importe nacheinander verarbeitet, weil parallele Importe auf einem Repository nicht möglich sind. Attribute • **Name**: optional • **QueueName**: erforderlich – gibt den Namen der Queue an. Diese kann bei einer <Export>-Aktion verwendet werden. • **Rep**: erforderlich – gibt das Repository an, in dem die Packages importiert werden sollen. Beispiel

	Es laufen parallel 2 Compile-Läufe und Exporte auf Repository1. Daneben werden zeitgleich die Exporte in Repository2 importiert.
	`<Parallel>`
	`  <ImportQueue QueueName="ImportQueue" Rep="{rep2}"/>`
	
	`  <Sequence Name="Compile-Lauf FS 3.7">`
	`  <CompileRun FS="3.7.0.0" Rep="{rep1}" Run="1"/>`
	`  <Export FS="3.7.0.0" Rep="{rep1}" Package="Cust1" Version="3.2"`
	`  Queue="ImportQueue"/>`
	`  <Export FS="3.7.0.0" Rep="{rep1}" Package="Cust2" Version="3.2"`
	`  Queue="ImportQueue"/>`
	`  </Sequence>`
	
	`  <Sequence Name="Compile-Lauf FS 3.8">`
	`  <CompileRun FS="3.8.0.0" Rep="{rep1}" Run="1"/>`
	`  <Export FS="3.8.0.0" Rep="{rep1}" Package="Cust1" Version="3.3"`
	`  Queue="ImportQueue"/>`
	`  <Export FS="3.8.0.0" Rep="{rep1}" Package="Cust2" Version="3.3"`
	`  Queue="ImportQueue"/>`
	`  </Sequence>`
	`</Parallel>`

Allgemein Aktionen

Die eigentliche Arbeit wird in den Aktionen ausgeführt. Neben der FSConsole.exe können auch allgemeine Aktionen definiert werden. So können z.B. entsprechende Ordner oder Dienste vorbereitet werden.

Knoten	Beschreibung

<Batch>	Damit kann eine beliebige Batch-Routine ausgeführt werden.
	Attribute
	• **Name**: optional
	• **Cmd**: erforderlich - gibt den Pfad für eine Exe oder eine Batch-Datei an. Die Angabe sollte mit komplettem Pfad erfolgen.
	• **Args**: optional – hier können Kommandozeilen-Argumente übergeben werden.
	
	Beispiele
	`<Batch Name="IIS neu starten" Cmd="iisreset"/>`
	`<Batch Name="IIS beenden" Cmd="NET" Args="STOP W3SVC"/>`
	`<Batch Name="IIS starten" Cmd="NET" Args="START W3SVC"/>`

<MD>	Legt ein neues Verzeichnis an. Attribute • **Name**: optional • **Dir**: Der anzulegende Ordner Beispiel `<MD Dir="{ExportDir}"/>`
<RD>	Löscht ein Verzeichnis samt ihrem Inhalt. Attribute • **Name**: optional • **Dir**: Der zu löschende Ordner Beispiel `<RD Dir="{ExportDir}"/>`

FSConsole-Aktionen

Kern dieser Routine ist die Arbeit mit der FSConsole.exe. Dazu werden folgende Aktionen angeboten

Knoten	Beschreibung
<FSConsole>	Startet eine FSConsole.exe Attribute

	• **Name**: optional • **FS**: erforderlich - gibt die Version von Framework-Studio an. "3.8.0.0", "3.7.0.0", … • **Rep**: erforderlich – gibt das Repository an, mit dem gearbeitet werden soll • **Args**: erforderlich – gibt die weiteren Kommandozeilen-Parameter an.
<CompileRun>	Führt mithilfe der FSConsole.exe einen Compile-Run durch Attribute • **Name**: optional • **FS**: erforderlich (siehe oben) • **Rep**: erforderlich (siehe oben) • **Run**: erforderlich – Gibt den Compile-Run an. Dieser kann im Package-Manager an der Package-Version gepflegt werden. • **MaxParallel**: optional – (ab FS 3.6) gibt an, wie viele Compiler innerhalb des CompileRuns parallel laufen dürfen. Beispiel Führt den CompileRun „1“ für Framework-Studio 3.8 aus. `<CompileRun FS="3.8.0.0" Rep="{rep1}" Run="1"/>`

<Export>	Exportiert ein Package. Dieses wird in dem Ordner abgelegt, der im Parameter „**ExportDir**“ angegeben wird. Der Name der Datei setzt sich aus dem Namen des Packages und der Version zusammen.
	Es werden automatisch die Debug-Informationen mit exportiert.
	Attribute
	• **Name**: optional
	• **FS**: erforderlich (siehe oben)
	• **Rep**: erforderlich (siehe oben)
	• **Package**: erforderlich – gibt den Namen des zu exportierenden Packages an
	• **Version**: erforderlich – gibt den Namen der zu exportierenden Package-Version an
	• **Mode**: optional – wenn ein Bugfix / ServiceRelease exportiert werden soll, muss **`Mode="Bugfix"`** angegeben werden
	• **Queue**: optional – Der Name der Import-Queue, mit der dieser Export wieder importiert werden soll. Siehe auch Knoten **<ImportQueue>**
	• **Dir:** optional – der Ordner, in dem die Export-Dateien abgelegt werden sollen, falls dieser vom Parameter „ExportDir“ abweicht.
	• **IncludeBasePackages**: optional – gibt an ob Basis-Package in den Export eingeschlossen werden sollen. Mögliche Werte:
	  ○ ServiceRelease - analog zum Package-Manager
	  ○ Unsealed - analog zum Package-Manager
	  ○ All - alle kompletten Base-Packages

<Import>	Importiert ein Package. Ordner und Dateinamen werden wie beim Export verwendet. Die Attribute müssen zum <Export>-Knoten passen. Bei der Verwendung der <ImportQueue> wird diese Aktion automatisch von der PBE.exe erzeugt. Es werden automatisch die Debug-Informationen mit importiert. Attribute • **Name**: optional • **FS**: erforderlich (siehe oben) • **Rep**: erforderlich (siehe oben)

	• **Package**: erforderlich – gibt den Namen des zu exportierenden Packages an • **Version**: erforderlich – gibt den Namen der zu exportierenden Package-Version an • **Mode**: optional – wenn ein Bugfix / ServiceRelease exportiert werden soll, muss **`Mode="Bugfix"`** angegeben werden • **Dir**: optional – der Ordner, aus dem die Export-Dateien gelesen werden sollen, falls dieser vom Parameter „ExportDir“ abweicht.
<ApprovedExport>	Arbeitet wie der <Export>-Konten mit folgenden Unterschieden: • Wenn nicht über das Attribut Dir anders definiert, dann wird für den Export-Ordner der Parameter „**ApprovedExportDir**“ verwendet. • Das Attribut **Queue** steht nicht zur Verfügung.
<ApprovedImport>	Importiert alle im Ordner enthaltenen Dateien in das angegebene Repository. Nach erfolgreichem Import werden die Dateien in das „HistoryDir“ verschoben. Wenn dieser Ordner nicht angegeben ist – weder als Parameter noch als Attribut – dann werden die Dateien nach dem Import gelöscht, damit sie beim nächsten Lauf nicht noch einmal importiert werden. Bei der Verarbeitung wird für jeden Import ein separater „Import“-Vorgang protokolliert. Die FS-Version wird aus dem Dateinamen ermittelt. Dieser muss z.B. so aussehen: „2014-11-10_FSDemo_3.9 (FS 3.9).db“. Beim Export wird bereits ein passender Dateiname erzeugt. Dieser sollte nicht verändert werden. Attribute • **Name**: optional • **Rep**: erforderlich (siehe oben) • **Dir**: optional – der Ordner, aus dem die zu importierenden Dateien gelesen werden sollen, falls dieser vom Parameter „**ApprovedImportDir**“ abweicht. • **HistoryDir**: optional – der Ordner, in den die Dateien nach erfolgreichen Import verschoben werden sollen, falls dieser vom Parameter „**ApprovedHistoryDir**“ abweicht.
<Publish>	Führt einen Publish-Vorgang aus. Attribute • **Name**: optional • **FS**: erforderlich (siehe oben) • **Rep**: erforderlich (siehe oben) • **Package**: erforderlich – gibt den Namen des zu exportierenden Packages an • **Version**: erforderlich – gibt den Namen der zu exportierenden Package-Version an • **Setting**: erforderlich – Name des Settings. Dieses muss im Publish-Wizard vorbereitet und abgespeichert werden.
<Publish2Go>	Führt einen Publish2Go-Vorgang aus. Attribute • **Name**: optional • **FS**: erforderlich (siehe oben) *(ab FS 3.11.8)* • **Rep**: erforderlich (siehe oben) • **Package**: erforderlich – gibt den Namen des zu exportierenden Packages an

	• **Version**: erforderlich – gibt den Namen der zu exportierenden Package-Version an • **Setting**: erforderlich – Name des Settings. Dieses muss im Publish-Wizard vorbereitet und abgespeichert werden. Im Setting muss der Folder angegeben sein. Dieser darf für den Publish2Go keinen Inhalt haben.
<ExportDoc>	Exportiert die komplette Dokumentation der Package-Version im HTML-Format. Attribute • **Name**: optional • **FS**: erforderlich (siehe oben) *(ab FS 3.11.8)* • **Rep**: erforderlich (siehe oben) • **Package**: erforderlich – gibt den Namen des zu exportierenden Packages an • **Version**: erforderlich – gibt den Namen der zu exportierenden Package-Version an • **Iso**: erforderlich – der Iso-Code der zu exportierenden Sprache – z.B. „de“ oder „en“ • **Dir**: optional – der Ordner, in dem die Dokumentation abgelegt werden soll, falls dieser vom Parameter „ExportDir“ abweicht. Für den Export wird ein Unter-Ordner mit dem folgenden Format erzeugt: {ExportFilePrefix}_<Package>_<Version>_Help_<Iso> Beispiel: ExportDir\2016-09-17_eNVenta_3.7_Help_de\...

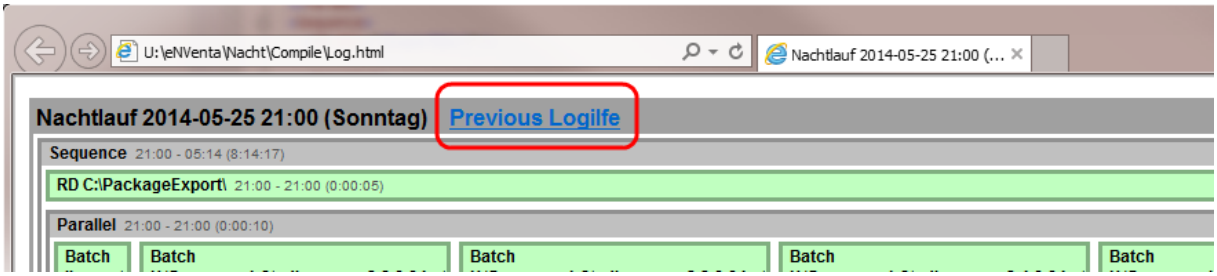
Protokoll

Die Routine erzeugt ein Protokoll. In dem Haupt-Knoten <PBE> kann eingestellt werden, wohin das Protokoll geschrieben werden soll. In jeder Protokoll-Datei wird ein Link „Previous Logfile“ erzeugt, in dem jeweils auf das vorherige „archivierte“ Protokoll verwiesen wird. So kann man sich einfach durch die Protokolle klicken.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE PBE SYSTEM "PBE.dtd">
<PBE Logfile="U:\eNVenta\Nacht\Compile\Log.html"
     Logarchive="U:\eNVenta\Nacht\Compile\Archiv\{DateTime}.html">
  <Params>
    <Param Name="ExportDir" Value="C:\PackageExport\"/>
```

Logfile: gibt den Pfad der Protokoll-Datei an. In Netzwerk-Umgebungen macht es Sinn, das Protokoll auf einem Netzlaufwerk abzulegen, so kann sie z.B. einfach über eine interne Web-Seite verlinkt werden.

Logarchive: gibt an, wo die Historischen Protokolle gespeichert werden sollen. Dabei ist es sinnvoll den Parameter {DateTime} in den Dateinamen einzubauen, um einen eindeutigen Namen zu erhalten.



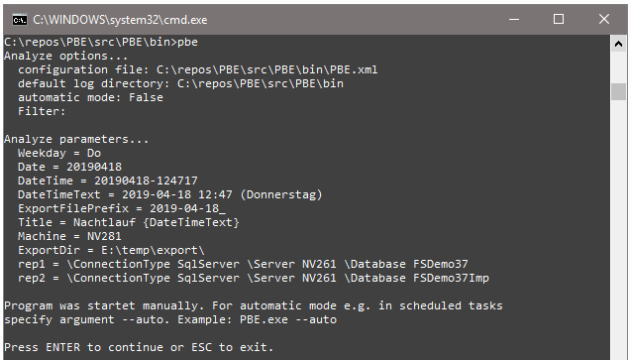
Beispiel-Konfiguration

Die folgende Konfiguration stellt beispielhaft einen etwas komplexeren Ablauf mit mehreren FS-Versionen und Repositories dar.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE PBE SYSTEM "PBE.dtd">
<PBE Logfile="C:\temp\Log.html" Logarchive="C:\temp\Archive\Log_{DateTime}.html">
  <FSVersions>
    <FSVersion FS="3.7.0.0" Dir="C:\Programme\Framework Systems\FrameworkStudio 3.7.3"/>
  </FSVersions>

  <Params>
    <Param Name="ExportDir" Value="E:\temp\export\"/>
    <Param Name="rep1" Value="\ConnectionType SqlServer \Server NV261 \Database FSDemo37"/>
    <Param Name="rep2" Value="\ConnectionType SqlServer \Server NV261 \Database FSDemo37Imp"/>
  </Params>

  <Sequence>
    <RD Name="letztes Export-Dir löschen" Dir="{ExportDir}"/>
    <MD Name="Export-Dir neu anlegen" Dir="{ExportDir}"/>

    <Parallel Name="Compile Repository 1">
      <ImportQueue QueueName="ImportQueue" Rep="{rep2}"/>

      <Sequence Name="Compile-Lauf FS 3.7">
        <CompileRun FS="3.7.0.0" Rep="{rep1}" Run="1"/>

        <Export FS="3.7.0.0" Rep="{rep1}" Package="Cust1" Version="3.2" Queue="ImportQueue"/>
        <Export FS="3.7.0.0" Rep="{rep1}" Package="Cust2" Version="3.2" Queue="ImportQueue"/>
      </Sequence>

      <Sequence Name="Compile-Lauf FS 3.8">
        <CompileRun FS="3.8.0.0" Rep="{rep1}" Run="1"/>

        <Export FS="3.8.0.0" Rep="{rep1}" Package="Cust1" Version="3.3" Queue="ImportQueue"/>
        <Export FS="3.8.0.0" Rep="{rep1}" Package="Cust2" Version="3.3" Queue="ImportQueue"/>
      </Sequence>
    </Parallel>

    <Parallel Name="Compile Repository 2">
      <CompileRun FS="3.7.0.0" Rep="{rep2}" Run="1"/>
      <CompileRun FS="3.8.0.0" Rep="{rep2}" Run="1"/>

      <Condition Name="Sonntags-Compile" Value="{Weekday}" Equals="So">
        <Parallel>
          <CompileRun FS="3.7.0.0" Rep="{rep2}" Run="Sonntag"/>
          <CompileRun FS="3.8.0.0" Rep="{rep2}" Run="Sonntag"/>
        </Parallel>
      </Condition>
    </Parallel>

    <Sequence Name="Publish">
      <Batch Name="IIS beenden" Cmd="NET" Args="STOP W3SVC"/>

      <Parallel Name="Publish" MaxTasks="2">
        <Publish FS="3.7.0.0" Rep="{rep1}" Package="Cust1" Version="3.2"
                 Setting="Cust1_Setting1"/>
        <Publish FS="3.7.0.0" Rep="{rep1}" Package="Cust1" Version="3.2"
                 Setting="Cust1_Setting2"/>
        <Publish FS="3.7.0.0" Rep="{rep1}" Package="Cust2" Version="3.2"
                 Setting="Cust2_Demo"/>
      </Parallel>

      <Batch Name="IIS wieder starten" Cmd="NET" Args="Start W3SVC"/>
    </Sequence>
  </Sequence>
</PBE>
```

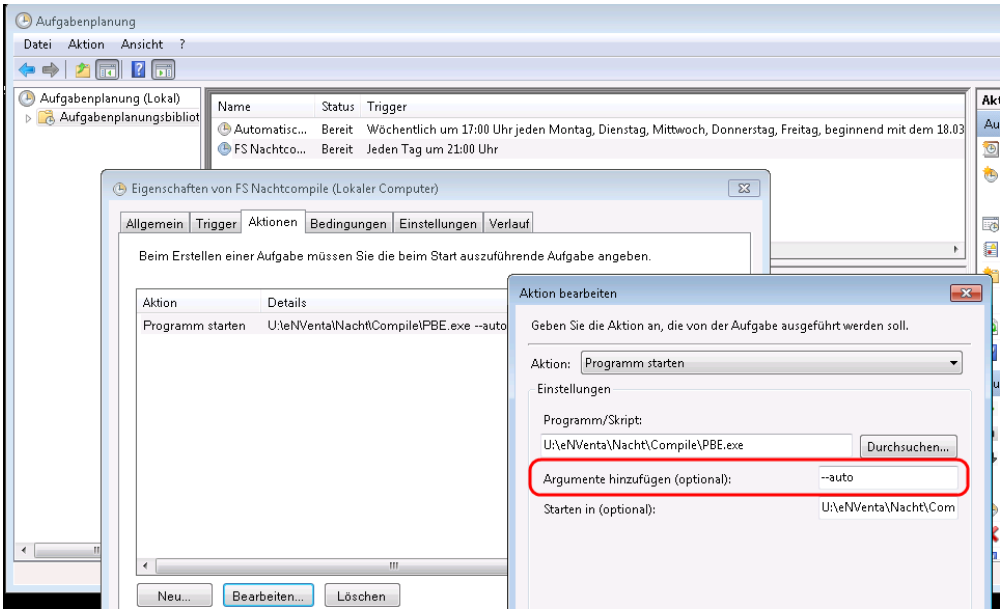
Starten der Routine

Die Routine kann manuell durch Ausführen der PBE.exe gestartet werden. Diese beinhaltet aber eine Absicherung und es muss zum Starten „ENTER“ gedrückt werden. Das verhindert, dass die Routine aus Versehen gestartet wird.



Am besten wird aber im Windows ein „Geplanter Task“ eingerichtet. Dabei muss darauf geachtet werden, dass der PBE.exe ein Kommandozeilen-Parameter --**auto** übergeben wird. Dieser hebelt die oben genannte Absicherung aus.

Es muss darauf geachtet werden, dass die Routine nur mit einem angemeldeten Benutzer arbeiten kann. Die FSConsole.exe benötigt ein Benutzer-Profil für Konfigurations-Informationen.



Kommandozeilen-Parameter

-a, --auto	Startet die Verarbeitung ohne eine weitere Rückfrage. Der alte Parameter **auto** (ohne --) wird auch unterstützt.
-c, --config	Gibt die XML-Datei an, die verarbeitet werden soll. Standardmäßig wird die Datei PBE.xml aus dem Programm-Verzeichnis gezogen.
-l, --logdir	Gibt den Ordner für die Log-Dateien an. Standardmäßig wird das Verzeichnis der XML-Datei verwendet. Ist in der XML-Datei die Log-Datei mit einem vollständigen Pfad angegeben, dann wird dieser Parameter ignoriert.
-f, --filter	Hier können die Task-Namen eingeschränkt werden, die ausgeführt werden sollen. Die Namen werden mit : getrennt. Die Groß-Klein-Schreibung ist relevant! Beispiel: --filter Taks1:Task2 Der alte Parameter **/filter** wird auch unterstützt, er kann aber nicht mit dem neuen Parameter kombiniert werden.
--help	Ausgabe der Hilfe

Installation

Für die Installation müssen alle Dateien in einen Ordner kopiert werden. Anschließend kann die Datei PBE.xml bearbeitet werden.

Es müssen immer alle Dateien zusammen in einem Ordner vorhanden sein. Wenn mehrere Routinen eingerichtet werden sollen, dann müssen mehrere Ordner mit jeweils allen Dateien angelegt werden.

Update

Das Programm PBE.exe wird mit jeder Version von Framework-Studio neu ausgeliefert. Dies ist nötig, weil sich die Möglichkeiten und damit die Parameter der FSConsole.exe regelmäßig ändern. Es wird empfohlen, dieses Update immer mit auszuführen, damit auch die jeweils richtige Unterstützung der FSConsole.exe gewährleistet ist.

Die Version der PBE.exe muss immer zur höchsten eingesetzten Version von Framework Studio passen.

Bei einem Update auf eine neuere Version der PBE.exe müssen folgende Dateien ersetzt werden:

- PBE.exe
- PBE.dtd
- CommandLine.dll
- LogTemplate.htm

Die Datei PBE.xml, welche die Konfiguration beinhaltet, muss stehen bleiben.

Neuheiten

Version 4.0.11

Überarbeitung der →

- Kommandozeilen-Parameter
  XML-Datei und Log-Verzeichnis können angegeben werden.

- Das <Condition>-Element kann im Value eine Funktion „#EXISTS“ verarbeiten.

Version 3.9

- Es kann ein „bestätigter“ Import abgebildet werden.

  Szenario:

  Aus einem Repository werden täglich Package-Versionen exportiert. Diese sollen aber nicht direkt in das Ziel-Repository importiert werden. Dazu wird der Export mithilfe des neuen Vorgangs <ApprovedExport> in einen separaten Ordner exportiert.

  Nachdem die Tests erfolgreich waren, verschiebt der Benutzer die Dateien aus dem ApprovedExportDir in das ApprovedImportDir.

  Beim nächsten Lauf werden dann mit dem Vorgang <ApprovedImport> alle im ApprovedImportDir befindlichen Dateien importiert und diese anschließend in das ApprovedHistoryDir verschoben.

- Der Name der Export-Dateien beinhaltet jetzt die FS-Version.
  Vorher: 2014-11-10_FSDemo_3.9.db
  Jetzt: 2014-11-10_FSDemo_3.9 (FS 3.9).db
  Dies ist nötig, damit bei der Verarbeitung der Dateien mit dem Vorgang <ApprovedImport> die richtige FS-Version erkannt wird.