IT-SERVICES DER SOZIALVERSICHERUNG GMBH

DVS Toolkit

User Guide

12.05.2015

V.0.9.4.2

Erstellt durch: Jörg Friede ITSV GmbH

Dokumentverwaltung

Verteiler

Name (alphab.)	Träger	Abteilung	Ort

Dokumentenhistorie

Version	Datum	Ersteller	Beschreibung der Änderungen
0.1	04.07.2013	Jörg Friede	Initiale Erstellung
0.2	26.08.2013	Jörg Friede	Erweiterungen
0.9.2	06.05.2014	Jörg Friede	Erweiterungen
0.9.3	17.06.2014	Jörg Friede	Erweiterungen
0.9.4	12.01.2015	Jörg Friede	Erweiterungen
0.9.4.1	12.05.2015	Jörg Friede	Dokumentation berichtigt
0.9.4.2	02.06.2015	Jörg Friede	Dokumentation berichtigt

QS-Prüfkreis

Zur Prüfung vorgelegt am	Prüfkreis	Zum Review zurück am	Freigegeben am

Inhalt

1 Einleitung

1.1 Zweck des Dokuments

1.2 Gültigkeit des Dokuments

1.3 Begriffsbestimmungen und Abkürzungen

Akronym	Erklärung

1.4 Zusammenhang mit anderen Dokumenten

- Kein Zusammenhang

2 Vorbedingungen

2.1 Betriebssystem

Das DVS-Toolkit unterstützt die folgenden Betriebssysteme/Versionen.

2.1.1 Windows

Windows XP/Vista/7.

2.1.2 Unix/Linux

Unter Unix/Linux wird als Laufzeitumgebung (Shell) „bash2“ benötigt.

2.2 Java Laufzeitumgebung

Java 7 JRE oder JDK (1.7.0_17 oder neuer) von Sun/Oracle oder unter AIX das IBM JDK/JRE für Version 7. Ältere Java Versionen (JRE/JDK 5/6) werden nicht unterstützt.

3 Installation

Entpacken Sie das ZIP- oder tar.gz Archiv in ein frei wählbares Verzeichnis.

4 Konfiguration

Damit die bereitgestellten Skripte funktionieren, muss die Umgebungsvariable JAVA_HOME auf das Wurzelverzeichnis Ihrer Java 7 Installation gesetzt sein.

4.1 Setzen von JAVA_HOME

Das hier beschriebene manuelle Setzen von JAVA_HOME ist nur notwendig, falls Sie keine Möglichkeit haben dies systemweit durchzuführen.

Öffnen der Datei „settings.cmd“ (Windows) oder „settings.sh“ (Unix/Linux) mit einem Texteditor. Zuweisen des Wurzelverzeichnisses an die Variable JAVA_HOME.

4.1.1 Windows

set JAVA_HOME=<DIR>

z.B.
set JAVA_HOME=C:\Program Files\Java\jdk1.7.0_17

oder
set JAVA_HOME=C:\Program Files\Java\jdk1.7.0_17\jre

4.1.2 Unix/Linux

export JAVA_HOME=

z.B.
export JAVA_HOME="/usr/java7_64"

4.2 Setzen des eigenen Versicherungsträgercode (VSTR)

Öffnen der Datei „settings.cmd“ (Windows) oder „settings.sh“ (Unix/Linux) mit einem Texteditor. Zuweisen des eigenen Versicherungsträgercodes an die Variable VSTR.

4.2.1 Windows

set VSTR=<VSTR>

z.B.
set VSTR=99

4.2.2 Unix/Linux

VSTR=<VSTR>

z.B.
VSTR=99

4.3 Java Speicher Optionen

Öffnen der Datei „settings.cmd“ (Windows) oder „settings.sh“ (Unix/Linux) mit einem Texteditor.

In der Umgebungsvariablen JAVA_MEM können spezielle Speichersettings für die Java-Laufzeitumgebung (JVM) angegeben werden,

z.B. die maximale Speichergröße JAVA_MEM="-Xmx256m“

4.4 Java User Optionen

Öffnen der Datei „settings.cmd“ (Windows) oder „settings.sh“ (Unix/Linux) mit einem Texteditor.

In der Umgebungsvariablen JAVA_USER_OPTIONS können spezielle Java System Properties für die Java-Laufzeitumgebung (JVM) angegeben werden.

4.4.1 Temporäres Verzeichnis

Per default wird das dem Benutzer vom Betriebssystem zugewiesene temporäre Verzeichnis verwendet. Dies kann mit dem System Property „at.itsv.dvs.tmp.dir“ überschreiben werden.

z.B.
JAVA_USER_OPTIONS=-Dat.itsv.dvs.tmp.dir=C:\TEMP (Windows)
JAVA_USER_OPTIONS=“-Dat.itsv.dvs.tmp.dir=/home/userXYZ/tmp“ (Unix/Linux)

4.4.2 Referenz IDs

Bei der Erzeugung von XML Beständen müssen auch sogenannte eindeutige „Referenz IDs“ erzeugt werden. Diese sollen eine Doppelverarbeitung von Beständen verhindern.

Das DVS-Toolkit bietet bereits einen eingebauten Defaultgenerator (siehe 4.4.2.1). Falls Dieser nicht ausreicht kann ein eigener Generator als Java Klasse angegeben werden (siehe 4.4.2.2).

4.4.2.1 Verzeichnis für Referenz IDs (Defaultmechanismus)

Die für die Erzeugung von XML Beständen benötigten Referenz IDs werden in einer speziellen Datei hochgezählt und gespeichert. Per default wird die Datei im Verzeichnis des aktuellen Benutzers im Unterverzeichnis „<USER HOME>/.dvs/ref“ angelegt. Über das System Property „at.itsv.dvs.ref.dir“ kann ein anderes Basisverzeichnis angegeben werden . Das Unterverzeichnis „.dvs/ref“ wird dabei jedoch weiterverwendet.

z.B.
JAVA_USER_OPTIONS=-Dat.itsv.dvs.ref.dir =C:\refdir (Windows)
JAVA_USER_OPTIONS=“-Dat.itsv.dvs.ref.dir =/home/userXYZ/refdir“ (Unix/Linux)

4.4.2.2 Java Klasse für Referenz IDs Generator

Die angegebene Java Klasse muss das Interface „at.itsv.dvs.util.refno. DVSReferenzNummerGenerator“ implementieren. Für ein komplettes Beispiel siehe 8.38.1.

z.B.
JAVA_USER_OPTIONS=-Dat.itsv.dvs.RefNoGeneratorClass=refgen.RandomRefNoGenerator (Windows)
JAVA_USER_OPTIONS=“-Dat.itsv.dvs.RefNoGeneratorClass=refgen.RandomRefNoGenerator“ (Unix/Linux)

4.4.3 Import/Export Listener

Das DVS-Toolkit bietet die Möglichkeit jeweils einen selbst implementierten Import bzw. Exportlistener über spezielle Systemproperties in das Laufzeitsystem einzubinden. Für komplette Beispiele siehe 8.1 bzw. 8.2.

z.B.
JAVA_USER_OPTIONS=-Dat.itsv.dvs.ImportListener=test.impexp.SampleImportListener
JAVA_USER_OPTIONS=-Dat.itsv.dvs.ExportListener=test.impexp.SampleExportListener
(Windows)
JAVA_USER_OPTIONS="-Dat.itsv.dvs.ImportListener=test.impexp.SampleImportListener"
JAVA_USER_OPTIONS="-Dat.itsv.dvs.ExportListener=test.impexp.SampleExportListener"
(Unix/Linux)

4.5 Java Classpath Optionen

Über die Umgebungsvariablen PREPEND_CLASSPATH und APPEND_CLASSPATH kann der Klassenpfad der DVS-Toolkit Utilities erweitert werden. Dabei wird PREPEND_CLASSPATH vor und APPEND_CLASSPATH nach dem Systemklassenpfad eingefügt.

z.B.
set APPEND_CLASSPATH=<PATH>\commons-lang3-3.3.2.jar (Windows)
set APPEND_CLASSPATH="<PATH>/commons-lang3-3.3.2.jar" (Unix/Linux)

4.6 Temporäre Dateien (nicht) löschen

Bei der Verarbeitung von satzartbasierten Dateien (SART) werden die einzelnen Bestände in temporäre Dateien zerlegt. Per default werden diese Dateien nach Abschluss der Verarbeitung gelöscht. Falls Sie diese Dateien über die Verarbeitung hinaus behalten wollen, können Sie dies über die Umgebungsvariable KEEP_TMP_FILES einstellen, z.B.

set KEEP_TMP_FILES=true (Windows)
export KEEP_TMP_FILES="true" (Unix/Linux)

5 Verzeichnisstruktur

5.1 bin

Dieses Verzeichnis enthält die DVS-Applikationen in Form von Kommandozeilenskripten.

Skripte mit der Erweiterung „cmd" sind unter Windows-, mit der Erweiterung „sh" unter Linux/Unix lauffähig.

5.2 conf

Dieses Verzeichnis enthält Konfigurationsdateien.

5.3 docs

Dieses Verzeichnis enthält die Dokumentation für das dvs-toolkit.

5.4 lib

Dieses Verzeichnis enthält alle benötigten JAR-Dateien.

5.5 samples

Dieses Verzeichnis enthält Beispiele für das dvs-toolkit.

5.5.1 data

Dieses Verzeichnis enthält Beispieldaten für die Applikationen dvsConverter/dvsTester.

5.5.2 export

Exportverzeichnis für die Beispiele der Applikationen dvsConverter/dvsTester.

5.5.3 profiles

Dieses Verzeichnis enthält Beispielprofile für die Applikationen dvsConverter/dvsTester.

6 Applikationen

6.1 dvsConverter.{cmd | sh }

Diese Applikation dient zum Verarbeiten von DVS-konformen Datenbeständen und der Umwandlung von Datenbeständen in verschiedene Formate.

6.1.1 Synopsis

```
usage: dvsConverter.cmd -p <path> [-i <type>] [-ip <path>] [-ic <charset>] [-ia <args>]
[-ipt <pattern>] [-e <type>] [-ep <path>] [-ec <charset>] [-ea <args>] [-ed <direction>]
[-ez <zip-type>] [-edt <date>] [-v | -ve]  [-d]

 -p,--profile <path>          Pfad zu Profildatei

 -i,--import <type>           Typ des Imports. Erlaubte Werte sind XML4, XML5, SART,
                              SAMMEL,EZBINARY,EZSART

 -ip,--ipath <path>           Pfad der/des zu importierenden Datei/Verzeichnis

 -ic,--icharset <charset>     Character Set der Importdatei(en). Default ist UTF-8

 -ia,--iargs <args>           Erweiterte Import Argumente (kommasepariert). Abhaengig vom
                              verwendeten Import-Typ. Siehe weiter unten

 -ipt,--ipattern <pattern>    Muster zur Einschraenkung der zu importierenden Datei(en)

 -e,--export <type>           Typ des Exports. Erlaubte Werte sind XML4, XML5, EBXML4,
                              EBXML5,SART,EBSART,SAMMEL,EZBINARY,EZSART

 -ep,--epath <path>           Pfad zu Datei/Verzeichnis in die/das exportiert werden soll

 -ec,--echarset <charset>     Character Set der exportierten Datei(en). Default ist UTF-8

 -ea,--eargs <args>           Erweiterte Export Argumente (kommasepariert). Abhaengig vom
                              verwendeten Export-Typ. Siehe weiter unten

 -ed,--edir <direction>       Richtung des Exports. Erlaubte Werte sind IN,OUT. Default
                              ist OUT

 -ez,--ezip <zip-type>        Kompressionstyp des Exports. Erlaubte Werte sind
                              ZIP,GZIP,NONE. Fuer SART/SAMMEL/EZBINARY/EZSART ist der
                              Defaultwert NONE. Fuer XML5/XML4 entweder abgeleitet aus der
                              Dateierweiterung oder ZIP
```

```
 -edt,--edate <date>          Datum des Exports. Default ist das aktuelles Datum

 -etf,--etemplate <template> Pfad zu Templatedatei

 -v,--verbose                 Zeigt detaillierte Import/Export Informationen an

 -ve,--verbose-error          Zeigt detaillierte Import/Export Informationen nur fuer
                              fehlerhafte Pakete/Bestaende an

 -d,--debug                   Zeigt detaillierte Fehlermeldungen an


Importer erweiterte Argumente (-ia):

-------------------------------------------------------------------------------------------
[SART]

splitData=true|false             SART-basierte Dateien waehrend Import in temporaere
                                  Einzeldateien aufsplitten. Default ist "true"


[SAMMEL]

ZVPA=<2*DIGIT>                   Code des Zielversicherungstraeger (fuer Paket).
                                  Obligatorisch

UVPA=<2*DIGIT>                   Code des Ursprungversicherungstraeger (fuer Paket)

splitData=true|false             SART-basierte Dateien waehrend Import in temporaere
                                  Einzeldateien aufsplitten. Default ist "true"


[EZBINARY]

ZVPA=<2*DIGIT>                    Code des Zielversicherungstraeger (fuer Paket).
                                  Obligatorisch

UVPA=<2*DIGIT>                    Code des Ursprungversicherungstraeger (fuer Paket)

useDirStruct=true|false           Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS
                                  1.0. Default ist "false"


[EZSART]

ZVPA=<2*DIGIT>                    Code des Zielversicherungstraeger (fuer Paket).
                                  Obligatorisch

UVPA=<2*DIGIT>                    Code des Ursprungversicherungstraeger (fuer Paket)

useDirStruct=true|false           Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS
                                  1.0. Default ist "false"

lineFlags=true|false              Zeilen des Bestands beginnen mit DA-ORG Kennzeichen.
                                  Default ist "false"


Exporter erweiterte Argumente (-ea):
```

```
-------------------------------------------------------------------------------
[SART]

oneFilePerPaket=true|false      Erzeuge eine Datei pro DVS-Paket. Default ist "false"

addCR=true|false                Carriage return an jede Zeile anhaengen. Default ist
                                 "false"

minLineLength=<number>          Mindestlaenge einzelner Zeilen. Kuerzere Zeilen werden
                                 auf den angegebenen Wert mit Leerzeichen aufgefuellt

fileNameOldStyle=true|false      "Legacy" Dateinamenformat verwenden, i.e. rmt<VSTR>...


[SAMMEL]

oneFilePerPaket=true|false      Erzeuge eine Datei pro DVS-Paket. Default ist "false"

addCR=true|false                Carriage return an jede Zeile anhaengen. Default ist
                                 "false"

minLineLength=<number>          Mindestlaenge einzelner Zeilen. Kuerzere Zeilen werden
                                 auf den angegebenen Wert mit Leerzeichen aufgefuellt

[EZBINARY]

useDirStruct=true|false          Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS
                                 1.0. Default ist "false"

testFlag=true|false              Setzt das Testkennzeichen in DVS 1.0 konformem
                                 Dateinamen

zvob=<value>                     ZVOB fuer DVS 1.0 konformen Dateinamen


[EZSART]

addCR=true|false                 Carriage return an jede Zeile anhaengen. Default ist
                                 "false"

minLineLength=<number>           Mindestlaenge einzelner Zeilen. Kuerzere Zeilen werden
                                 auf den angegebenen Wert mit Leerzeichen aufgefuellt

addLineFlags=true|false          Fuegt DA-ORG Datenkennzeichen (3) an den Beginn jeder
                                 Zeile hinzu. Default ist "false"

useDirStruct=true|false          Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS
                                 1.0. Default ist "false"

testFlag=true|false              Setzt das Testkennzeichen in DVS 1.0 konformem
                                 Dateinamen

zvob=<value>                     ZVOB fuer DVS 1.0 konformen Dateinamen
```

6.1.2 Begriffsklärungen

Die nachfolgenden Abschnitte beschreiben die zur Verfügung stehenden Import/Export Typen.

6.1.2.1 SART

Die Daten sind in einem DDS-Paket enthalten. In einer Datei können mehrere Bestände enthalten sein. Der Aufbau entspricht also exakt dem Aufbau eines DDS-Paketes laut Organisationsbeschreibung: Datenaustausch mit dem Hauptverband (DA).

6.1.2.2 XML5

Die Daten sind in einem XML-DDS-Paket enthalten. In einer Datei können mehrere Bestände enthalten sein. Der Aufbau entspricht also exakt dem Aufbau eines XML-DDS-Paketes laut DA-ORG (inkl. XML-Nachricht „DatendrehscheibePaket“ VERSION 5).

6.1.2.3 XML4

Die Daten sind in einem XML-DDS-Paket enthalten. In einer Datei können mehrere Bestände enthalten sein. Der Aufbau entspricht also exakt dem Aufbau eines XML-DDS-Paketes laut DA-ORG (inkl. XML-Nachricht „DatendrehscheibePaket“ VERSION 4).

6.1.2.4 SAMMEL

Die Dateien enthalten satzartbasierte Bestände laut DA-ORG. In einer Datei können mehrere Bestände enthalten sein. Jeder Bestand beginnt mit einem Vorsatz und endet mit einem Nachsatz. Die einzelnen Datensätze beginnen mit dem Satzkennzeichen „3“. Zwischen Vorsatz und dem ersten Datensatz kann ein Vorlaufsatz (Satzkennzeichen „2“) stehen.

Der Paketkopf und Paketendesatz ist nicht enthalten. Der Aufbau entspricht also dem Aufbau eines DDS-Paketes laut DA-ORG ohne Paketkopf- und Paketendesatz. Sinn dieses Formats ist es, einem STP weiterhin die Möglichkeit zu geben alle seine Daten als Bestände in eine Datei zu schreiben.

6.1.2.5 EZSART

Jeder Bestand ist in einer Datei enthalten. Die Daten enthalten keinen Vorsatz und Nachsatz laut DDS.

Jede Zeile enthält einen Datensatz. Optional können die Datensätze bereits das Satzkennzeichen enthalten. Dies ist in den Konfigurationsdaten anzugeben, per default ist das Satzkennzeichen nicht enthalten.

Projekt- und Listkennzeichen und ZVST sind in diesem Fall nicht in den Daten enthalten und müssen beim Import aufgrund der Dateinamen ermittelt werden können. Folgende Struktur ist dabei vorgegeben:

```
<xxx>_<zvstr>_<uvstr>_<proj>_<list>_[t]_[zvob].XXXXXXXXXX

<xxx> ... beliebiger Prefix bei Import. Bei Export Defaultwert „dvs“.
<zvstr> ... Zielversicherungsträgercode (2 Alphanumerisch)
<uvstr> ... Ursprungsversicherungsträger (2 Alphanumerisch)
<proj> ... Projektkennzeichen (2 Alphanumerisch)
<list> ... Listkennzeichen (2 Alphanumerisch)
[t]    ... Testkennzeichen ('t' oder nichts)
[ZVOB] ... Zusätzlicher Ordnungsbegriff (kein „.“, „/“ oder „_“ erlaubt)

XXXXXXXXXX … in Verantwortung des Erzeugers (kein „.“, „/“ oder „_“ erlaubt)
```

Bei Export wird per default der Prefix „dvs“ verwendet werden. Dies kann über das erweiterte Exportargument „fileNamePrefix“ verändert werden (siehe 6.1.3.4.4.,6.1.3.4.5).

Wird ein ZVOB (zusätzlicher Ordnungsbegriff) angegeben so wird dieser beim Import/Export übernommen. Wird Dieser nicht angegeben so generiert DVS einen eigenen Ordnungsbegriff.

Sonderzeichen im ZVOB müssen in Ihrer URL-Encoded Form angegeben werden, bzw. werden in Ihrer URL-Encoded Form in den Dateinamen aufgenommen. Folgende Ersetzungen müssen dabei dürchgeführt werden bzw. werden bei der Erstellung einer Datei durchgeführt:

Zeichen	Encoded
/	%2F
\	%5C
<SPACE>	%20
:	%3A
*	%2A
?	%3F
"	%22
,	%27
<	%3C
>	%3E
|	%7C
+	%2B
ö	%C3%B6
ä	%C3%A4
ü	%C3%BC
Ö	%C3%96
Ä	%C3%84
Ü	%C3%9C
ß	%C3%9F

Wird das Testkennzeichen 'T' angegeben so wird dies beim Export des Bestandes in den Dateinamen übernommen.

Alternativ können die Dateien auch in eine Unterverzeichnisstruktur gespeichert werden. Dazu muss „useDirStruct=true“ in den erweiterten Argumenten angegeben werden. Projekt- und Listkennzeichen und ZVST/UVST sind in diesem Fall nicht in den Daten enthalten und müssen daher aufgrund der Verzeichnisstruktur nach folgendem Schema ermittelt werden können:

```
|– <… lokales Wurzelverzeichnis …>
      ├– <proj>
            ├– <list>
                  ├– <vstr>


<proj>      ... Projektkennzeichen (2 Alphanumerisch)

<list>      ... Listkennzeichen (2 Alphanumerisch)

<vstr>      ... Ziel-/Ursprungsversicherungsträgercode (2 Alphanumerisch)
```

Beim Export in dieses Format bezeichnet VSTR den Ursprungsversicherungsträgercode. Beim Import den Zielversicherungsträgercode.

6.1.2.6 EZBINARY

Stellt beliebige binäre Datenbestände, gespeichert in einer Datei, dar. Aufbau des Dateinamens und sonstige Besonderheiten siehe 6.1.2.5.

6.1.2.7 EBSART

Der Aufbau entspricht exakt dem Aufbau einer satzartbasierten Empfangsbestätigung laut Organisationsbeschreibung: Datenaustausch mit dem Hauptverband (DA).

6.1.2.8 EBXML5

Der Aufbau entspricht exakt dem Aufbau einer XML-Empfangsbestätigung laut DA-ORG (inkl. XML-Nachricht „EmpfangsbestaetigungsFile“ VERSION 5).

6.1.2.9 EBXML4

Der Aufbau entspricht exakt dem Aufbau einer XML-Empfangsbestätigung laut DA-ORG (inkl. XML-Nachricht „EmpfangsbestaetigungsFile“ VERSION 4).

6.1.3 Beschreibung

Die Applikation wird über Einträge in speziellen Textdateien – sogenannten Profilen – gesteuert. Diese Dateien müssen über den Kommandozeilenparameter („-p/ --profile“) angegeben werden. Wahlweise können die einzelnen Einstellungen in den Profilen noch über weitere Kommandozeilenparameter überschrieben werden.

6.1.3.1 Einträge Profildatei / Kommandozeilenparameter

Die Einträge in den Profildateien sind zeilenorientiert in folgender Form zu halten:

<eintrag>=<wert>↵

Zeilen die mit dem Zeichen „#“ beginnen, werden als Kommentar behandelt.

Diese Einträge können auf der Kommandozeile überschrieben werden.

6.1.3.1.1 import.type (obligatorisch)

Typ des Imports. Erlaubte Werte sind XML4,XML5,SART,SAMMEL,EZBINARY,EZSART. Entspricht dem Kommandozeilenparameter „-i/ --import“.

6.1.3.1.2 import.path (obligatorisch)

Pfad der/des zu importierenden Datei/Verzeichnisses. Unter Windows müssen Backslahes („\“) entweder durch doppelte Backslahes („\\“) oder einen einfachen Slash („/“) ersetzt werden.

Entspricht dem Kommandozeilenparameter „-ip/ --ipath“. Hier ist eine Ersetzung der Backslahes unter Windows nicht notwendig.

6.1.3.1.3 import.charset

Character Set der Importdatei(en). Default ist UTF-8. Entspricht dem Kommandozeilenparameter „-ic/--icharset“.

Typische Werte sind UTF-8 (Zeichensatz UTF-8 lt. DA-ORG), ISO-8859-1 (Codepage ISO 8859-1 lt. DA-ORG) oder IBM850 (Codepage 850 lt. DA-ORG).

6.1.3.1.4 import.extargs

Erweiterte Import Argumente (kommasepariert). Abhaengig vom verwendeten Import-Typ. Entspricht dem Kommandozeilenparameter „-ia/ --iargs“.

z.B.
import.extargs=ZVPA=11,lineFlags=true oder - *ia ZVPA=11,lineFlags=true*

6.1.3.1.5 import.pattern

Muster zur Einschränkung der zu importierenden Datei(en). Entspricht dem Kommandozeilenparameter „-ipt/-- ipattern“.

Für eine Beschreibung des Aufbaus der Suchmuster(Pattern) siehe 7.1.

6.1.3.1.6 export.type (obligatorisch)

Typ des Exports. Erlaubte Werte sind XML4, XML5, EBXML4, EBXML5, SART, EBSART, SAMMEL,EZBINARY,EZSART. Entspricht dem Kommandozeilenparameter „-e/ --export“.

6.1.3.1.7 export.path

Pfad zu Datei/Verzeichnis in die/das exportiert werden soll. Unter Windows müssen Backslahes („\“) entweder durch doppelte Backslahes („\\“) oder einen einfachen Slash („/“) ersetzt werden.

Entspricht dem Kommandozeilenparameter „-ep/ --epath“. Hier ist eine Ersetzung der Backslahes unter Windows nicht notwendig.

6.1.3.1.8 export.charset

Character Set der exportierten Datei(en). Default ist UTF-8. Entspricht dem Kommandozeilenparameter „-ec/ --echarset“.

Typische Werte sind UTF-8 (Zeichensatz UTF-8 lt. DA-ORG), ISO-8859-1 (Codepage ISO 8859-1 lt. DA-ORG) oder IBM850 (Codepage 850 lt. DA-ORG).

6.1.3.1.9 export.extargs

Erweiterte Export Argumente (kommasepariert). Abhaengig vom verwendeten Export-Typ. Entspricht dem Kommandozeilenparameter „-ea/ --eargs“.

z.B.
export.extargs=addCR=true,minLineLength=750 oder *-ea addCR=true,minLineLength=750*

6.1.3.1.10 export.direction

„Richtung“ des Exports. Erlaubte Werte sind IN, OUT. Default ist OUT.

Die „Richtung“ wird als „ein“ (IN) oder „aus“ (OUT) zu den exportierten Dateien hinzugefügt, z.B. ddein<VSTR>… oder ddaus<VSTR>…

6.1.3.1.11 export.ziptype

Kompressionstyp des Exports. Erlaubte Werte sind ZIP, GZIP, NONE. Für SART/SAMMEL/EINZEL ist der Defaultwert NONE. Fuer XML5/XML4 entweder abgeleitet aus der Dateierweiterung oder ZIP.

6.1.3.1.12 export.date

Datum des Exports. Default ist das aktuelles Datum. Format yyyyMMdd.

6.1.3.1.13 export.templatefile

Pfad zu Templatedatei. Für eine detaillierte Beschreibung des Template-Mechanismus siehe 6.1.3.2.

6.1.3.2 Template Mechanismus

Der Template Mechanismus bietet die Möglichkeit, vor dem Export, nicht befüllte Werte von Paketen (nur Bezeichnung) und Beständen mit vorgegebenen Werten zu befüllen.

Diese Werte können in einer XML-Datei vorbereitet werden. Diese Datei kann entweder über den Kommandozeilenparameter „-etf,--etemplate“ oder das Profilproperty „export.templatefile“ angegeben werden.

Als Vorlage kann die Datei „<installdir>/conf/default.template“ verwendet werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<dds:DatendrehscheibePaket
xmlns:dds="http://www.sozvers.at/ns/if/dds/draft/1/5/DDS_Schema_V1_5.xsd">
      <dds:Paketkopf>
        <dds:Paketbezeichnung></dds:Paketbezeichnung>
    </dds:Paketkopf>
    <dds:BestandsListe>
        <dds:Bestand>
            <dds:BestandsInfo>
                <dds:ProjektKennzeichen></dds:ProjektKennzeichen>
                <dds:ListKennzeichen></dds:ListKennzeichen>
                <dds:AenderungsdienstNummer></dds:AenderungsdienstNummer>
                <dds:EingabeBestandsNummer></dds:EingabeBestandsNummer>
                <dds:EingabeArt></dds:EingabeArt>
                <dds:TestKennzeichen></dds:TestKennzeichen>
                <dds:ZusaetzlOrdnungsbegriff></dds:ZusaetzlOrdnungsbegriff>
                <dds:ZusaetzlInfo></dds:ZusaetzlInfo>
            </dds:BestandsInfo>
            <dds:FileInfo>
                <dds:LinkAufBestandsFile></dds:LinkAufBestandsFile>
```

```
            <dds:Inhalt></dds:Inhalt>
            <dds:Signiert>N</dds:Signiert>
            <dds:Verschluesselt>N</dds:Verschluesselt>
            <dds:Komprimiert>N</dds:Komprimiert>
            <dds:HashWert></dds:HashWert>
        </dds:FileInfo>
        <dds:AboInfo>
            <dds:Abo>N</dds:Abo>
            <dds:AboKey></dds:AboKey>
        </dds:AboInfo>
        <dds:DDSOLInfo>
            <dds:PartnerCode></dds:PartnerCode>
            <dds:Hinweis1></dds:Hinweis1>
            <dds:Hinweis2></dds:Hinweis2>
        </dds:DDSOLInfo>
    </dds:Bestand>
</dds:BestandsListe>
</dds:DatendrehscheibePaket>
```

Wie oben zu sehen ist kann für das Paket nur die „Paketbezeichnung“ angeben werden. Für Bestände können jedoch mehrere Bestandsvorlagen angegeben werden. Dabei wird die Kombination Projekt/Listkennzeichen als Unterscheidungsmerkmal verwendet. Die Kombination aus leerem Projekt- und leerem Listkennzeichen gilt für alle Bestände für deren Kombination aus Projekt/Listkennzeichen keine eigene Bestandsvorlage definiert wurde.

Die Tags <dds:AboInfo> und <dds:DDSOLInfo> sind optional und können daher weggelassen werden.

Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<dds:DatendrehscheibePaket
xmlns:dds="http://www.sozvers.at/ns/if/dds/draft/1/5/DDS_Schema_V1_5.xsd">
    <dds:Paketkopf>
      <dds:Paketbezeichnung>PAKET</dds:Paketbezeichnung>
    </dds:Paketkopf>
    <dds:BestandsListe>
        <dds:Bestand>
            <dds:BestandsInfo>
                <dds:ProjektKennzeichen></dds:ProjektKennzeichen>
                <dds:ListKennzeichen></dds:ListKennzeichen>
                <dds:AenderungsdienstNummer>KLAE</dds:AenderungsdienstNummer>
                <dds:EingabeBestandsNummer>88</dds:EingabeBestandsNummer>
                <dds:EingabeArt>WO</dds:EingabeArt>
                <dds:TestKennzeichen></dds:TestKennzeichen>
                <dds:ZusaetzlOrdnungsbegriff>
                   ${UVST}-${REFNR}
                </dds:ZusaetzlOrdnungsbegriff>
                <dds:ZusaetzlInfo>INFO</dds:ZusaetzlInfo>
            </dds:BestandsInfo>
            <dds:FileInfo>
                <dds:LinkAufBestandsFile></dds:LinkAufBestandsFile>
                <dds:Inhalt>pdf</dds:Inhalt>
                <dds:Signiert>N</dds:Signiert>
                <dds:Verschluesselt>J</dds:Verschluesselt>
                <dds:Komprimiert>J</dds:Komprimiert>
                <dds:HashWert>SHA-512</dds:HashWert>
            </dds:FileInfo>
```

```
            <dds:AboInfo>
                <dds:Abo>J</dds:Abo>
                <dds:AboKey>ABO</dds:AboKey>
            </dds:AboInfo>
            <dds:DDSOLInfo>
                <dds:PartnerCode>HH</dds:PartnerCode>
                <dds:Hinweis1>HINTI 1</dds:Hinweis1>
                <dds:Hinweis2>HINTI 2</dds:Hinweis2>
            </dds:DDSOLInfo>
        </dds:Bestand>
        <dds:Bestand>
            <dds:BestandsInfo>
                <dds:ProjektKennzeichen>IP</dds:ProjektKennzeichen>
                <dds:ListKennzeichen>KV</dds:ListKennzeichen>
                <dds:AenderungsdienstNummer>346X</dds:AenderungsdienstNummer>
                <dds:EingabeBestandsNummer>30</dds:EingabeBestandsNummer>
                <dds:EingabeArt>IN</dds:EingabeArt>
                <dds:TestKennzeichen>T</dds:TestKennzeichen>
                <dds:ZusaetzlOrdnungsbegriff></dds:ZusaetzlOrdnungsbegriff>
                <dds:ZusaetzlInfo>TESTINFO</dds:ZusaetzlInfo>
            </dds:BestandsInfo>
            <dds:FileInfo>
                <dds:LinkAufBestandsFile></dds:LinkAufBestandsFile>
                <dds:Inhalt>zip</dds:Inhalt>
                <dds:Signiert>N</dds:Signiert>
                <dds:Verschluesselt>N</dds:Verschluesselt>
                <dds:Komprimiert>N</dds:Komprimiert>
                <dds:HashWert></dds:HashWert>
            </dds:FileInfo>
        </dds:Bestand>
    </dds:BestandsListe>
</dds:DatendrehscheibePaket>
```

In diesem Beispiel wird die Paketbezeichnung auf „PAKET“ gesetzt falls Diese in dem zu exportierenden Paket leer ist.

Weiters werden zwei Bestandsvorlagen definiert:

- für Bestände deren Projektkennzeichen „IP“ und das Listkennzeichen „KV“ enthält
- für alle anderen zu exportierenden Bestände

Spezielle Werte

Der Wert des Feldes „<dds:ZusaetzlOrdnungsbegriff>“ kann über folgende Variablen dynamisch erzeugt werden:

Variable	Beschreibung
${ZVST}	Wird durch den ZVST des aktuellen Bestandes ersetzt
${UVST}	Wird durch den UVST des aktuellen Bestandes ersetzt
${PROJ}	Wird durch das Projektkennzeichen des aktuellen Bestandes ersetzt

${LIST}	Wird durch das Listkennzeichen des aktuellen Bestandes ersetzt
${REFNR}	Wird durch die Referenznummer des aktuellen Bestandes ersetzt

Im Feld „<dds:HashWert>“ kann ein Hashalgorithmus angegeben werden. Dieser wird verwendet um die Hashwerte für die Bestandsdateien eines XML-Paketes zu generieren. Der Hashalgorithmus muss der Java Virtual Machine zur Laufzeit bekannt sein.

Algorithmen der Java Virtual Machine von Oracle/Sun:

- MD2
- MD5
- SHA
- SHA-256
- SHA-384
- SHA-512

6.1.3.3 Erweiterte Argumente Import

Die erweiterten Argumente sind abhängig vom jeweils gewählten Importtyp. Die Parameternamen sind Case-sensitiv.

6.1.3.3.1 SART

Parameter	Werte	Beschreibung	Muss	Defaultwert
splitData	true / false	SART-basierte Dateien waehrend Import in temporaere Einzeldateien aufsplitten		true

6.1.3.3.2 SAMMEL

Parameter	Werte	Beschreibung	Muss	Defaultwert
ZVPA	<2*DIGIT>	Code des Zielversicherungstraeger (fuer Paket)	X	
UVPA	<2*DIGIT>	Code des Ursprungversicherungstraeger (fuer Paket)		
splitData	true / false	SART-basierte Dateien waehrend Import in temporaere Einzeldateien aufsplitten		true

6.1.3.3.3 EZBINARY

Parameter	Werte	Beschreibung	Muss	Defaultwert
ZVPA	<2*DIGIT>	Code des Zielversicherungstraeger (fuer Paket)	X	
UVPA	<2*DIGIT>	Code des Ursprungversicherungstraeger (fuer Paket)		
useDirStruct	true / false	Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS 1.0		false

6.1.3.3.4 EZSART

Parameter	Werte	Beschreibung	Muss	Defaultwert
ZVPA	<2*DIGIT>	Code des Zielversicherungstraeger (fuer Paket)	X	
UVPA	<2*DIGIT>	Code des Ursprungversicherungstraeger (fuer Paket)		
useDirStruct	true / false	Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS 1.0		false
lineFlags	true / false	Zeilen des Bestands beginnen mit DA-ORG Kennzeichen		false

6.1.3.4 Erweiterte Argumente Export

Die erweiterten Argumente sind abhängig vom jeweils gewählten Exporttyp. Die Parameternamen sind Case-sensitiv.

6.1.3.4.1 SART

Parameter	Werte	Beschreibung	Muss	Defaultwert
oneFilePerPaket	true / false	Erzeuge eine Datei pro DVS-Paket		false
addCR	true / false	Carriage return an jede Zeile anhaengen		false
minLineLength	<number>	Mindestlänge einzelner Zeilen. Kürzere Zeilen werden auf den angegebenen Wert mit Leerzeichen aufgefüllt		
fileNameOldStyle	true / false	"Legacy" Dateinamenformat verwenden, i.e. rmt<VSTR>...		false
fileNamePrefix		Erweiterung in Dateinamen, i.e. ddein<VSTR>.<**PREFIX**>.<CCYYMMDD>.<nnnnnn>[.xml][.<gz/zip>]		

6.1.3.4.2 SAMMEL

Parameter	Werte	Beschreibung	Muss	Defaultwert
oneFilePerPaket	true / false	Erzeuge eine Datei pro DVS-Paket		false
addCR	true / false	Carriage return an jede Zeile anhaengen		false
minLineLength	<number>	Mindestlänge einzelner Zeilen. Kürzere Zeilen werden auf den angegebenen Wert mit Leerzeichen aufgefüllt		
fileNamePrefix		Erweiterung in Dateinamen, i.e. ddein<VSTR>.<**PREFIX**>.<CCYYMMDD>.<nnnnnn>[.xml][.<gz/zip>]		

6.1.3.4.3 XML4/XML5

Parameter	Werte	Beschreibung	Muss	Defaultwert
fileNamePrefix		Erweiterung in Dateinamen, i.e. ddein<VSTR>.<**PREFIX**>.<CCYYMMDD>.<nnnnnn>[.xml][.<gz/zip>]		

6.1.3.4.4 EZBINARY

Parameter	Werte	Beschreibung	Muss	Defaultwert
useDirStruct	true / false	Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS 1.0		false
testFlag	true / false	Setzt das Testkennzeichen in DVS 1.0 konformem Dateinamen		false
zvob	<alpha>	ZVOB fuer DVS 1.0 konformen Dateinamen		
fileNamePrefix		Setzt Dateinamenprefix, i.e. <**PREFIX**>_<zvstr>_<uvstr>_<proj>_<list>_[t]_[zvob].XXXXXXXXXX		

6.1.3.4.5 EZSART

Parameter	Werte	Beschreibung	Muss	Defaultwert
addCR	true / false	Carriage return an jede Zeile anhaengen		false

minLineLength	<number>	Mindestlänge einzelner Zeilen. Kürzere Zeilen werden auf den angegebenen Wert mit Leerzeichen aufgefüllt	
addLineFlags	true / false	Fuegt DA-ORG Datenkennzeichen (3) an den Beginn jeder Zeile hinzu	false
useDirStruct	true / false	Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS 1.0	false
testFlag	true / false	Setzt das Testkennzeichen in DVS 1.0 konformem Dateinamen	false
zvob	<alpha>	ZVOB fuer DVS 1.0 konformen Dateinamen	
fileNamePrefix		Setzt Dateinamenprefix, i.e. <**PREFIX**>_<zvstr>_<uvstr>_<proj>_<list>_[t]_[zvob].XXXXXXXXXX	

6.1.4 Beispiele

Die in den folgenden Profildateien verwendeten Pfaden gelten relativ zu dem „bin“ Verzeichnis des dvs-toolkit, d.h. die Beispiele müssen aus dem „bin“ Verzeichnis heraus gestartet werden. Dazu muss auf Kommandozeilenebene in das „bin“ Verzeichnis gewechselt werden.

6.1.4.1 EINZEL_DIRSTRUCT_SART_LF_to_SART.profile

Dieses Beispiel importiert Einzelbestände die in einer Unterverzeichnisstruktur gespeichert sind. Die Zeilen der satzartbasierenden Einzelbestände weisen bereits das Satzkennzeichen 3 auf.

Die Einzelbestände werden als DDS-Paketdatei lt. DA-ORG exportiert. Die Export-Richtung ist „ein“.

6.1.4.2 EINZEL_DIRSTRUCT_SART_to_SART_GZIP.profile

Dieses Beispiel importiert Einzelbestände die in einer Unterverzeichnisstruktur gespeichert sind. Die Zeilen der satzartbasierenden Einzelbestände weisen kein Satzkennzeichen 3 auf.

Die Einzelbestände werden als DDS-Paketdatei lt. DA-ORG exportiert und mittels GZIP komprimiert.

6.1.4.3 EINZEL_DIRSTRUCT_to_XML4_GZIP.profile

Dieses Beispiel importiert Einzelbestände die in einer Unterverzeichnisstruktur gespeichert sind. Die Einzelbestände sind von beliebigem Format und werden als binäre Daten ohne Struktur behandelt.

Die Einzelbestände werden als XML-DDS-Paket Version 4 lt. DA-ORG exportiert und mittels GZIP komprimiert.

6.1.4.4 EINZEL_DIRSTRUCT_to_XML5.profile

Dieses Beispiel importiert Einzelbestände die in einer Unterverzeichnisstruktur gespeichert sind. Die Einzelbestände sind von beliebigem Format und werden als binäre Daten ohne Struktur behandelt.

Die Einzelbestände werden als XML-DDS-Paket Version 5 lt. DA-ORG exportiert und per default mittels ZIP komprimiert.

6.1.4.5 EINZEL_FILE_LF_to_SART_GZIP.profile

Dieses Beispiel importiert Einzelbestände deren Metadaten sich aus den Dateinamen ergeben. Die Zeilen der satzartbasierenden Einzelbestände weisen bereits das Satzkennzeichen 3 auf. Es werden nur Dateien mit den Erweiterungen „lf“ berücksichtigt.

Die Einzelbestände werden als DDS-Paketdatei lt. DA-ORG exportiert und mittels GZIP komprimiert. Die Export-Richtung ist „ein“.

6.1.4.6 EINZEL_FILE_to_SART.profile

Dieses Beispiel importiert Einzelbestände deren Metadaten sich aus den Dateinamen ergeben. Die Zeilen der satzartbasierenden Einzelbestände weisen kein Satzkennzeichen 3 auf. Es werden nur Dateien mit den Erweiterungen „ok“ und „test“ berücksichtigt.

Die Einzelbestände werden als DDS-Paketdatei lt. DA-ORG exportiert.

6.1.4.7 EINZEL_FILE_to_XML5.profile

Dieses Beispiel importiert Einzelbestände deren Metadaten sich aus den Dateinamen ergeben. Die Einzelbestände sind von beliebigem Format und werden als binäre Daten ohne Struktur behandelt. Es werden nur Dateien mit den Erweiterungen „pdf“ berücksichtigt.

Die Einzelbestände werden als XML-DDS-Paketdatei Version 5 lt. DA-ORG exportiert.

6.1.4.8 EINZEL_FILE_to_XML5_WITH_TEMPLATE.profile

Wie 6.1.4.7. Fehlende Werte werde jedoch auch noch aus der Template-Datei „../templates/ sample1.template“ ergänzt.

6.1.4.9 SAMMEL_to_SART.profile

Dieses Beispiel importiert Sammelbestände.

Die Sammelbestände werden als DDS-Paketdatei lt. DA-ORG exportiert.

6.1.4.10 SAMMEL_to_SART_MULTIPLE_FILES.profile

Dieses Beispiel importiert Sammelbestände.

Die Sammelbestände werden als DDS-Paketdatei lt. DA-ORG exportiert. Dabei wird jedes DDS-Paket in eine eigene Datei gespeichert.

6.1.4.11 SAMMEL_to_SART_LEGACY_STYLE.profile

Dieses Beispiel importiert Sammelbestände.

Die Sammelbestände werden als DDS-Paketdatei lt. DA-ORG exportiert. Dabei wird das "Legacy" Dateinamenformat verwendet, i.e. rmt<VSTR>...

6.1.4.12 SART_850_to_SART_UTF-8.profile

Dieses Beispiel importiert DDS-Paketdateien mit dem Zeichensatz 850 und exportiert DDS-Paketdateien mit dem Zeichensatz UTF-8.

6.1.4.13 SART_8859-1_to_SART_850.profile

Dieses Beispiel importiert DDS-Paketdateien mit dem Zeichensatz ISO 8859-1 und exportiert DDS-Paketdateien mit dem Zeichensatz 850.

6.1.4.14 SART_to_EBSART.profile

Dieses Beispiel importiert die DDS-Paketdatei „…/sart.zip“ und generiert satzartbasierende Empfangsbestätigungen lt. DA-ORG. Die Export-Richtung ist „ein“.

6.1.4.15 SART_to_EBXML5.profile

Dieses Beispiel importiert DDS-Paketdateien und generiert XML-Empfangsbestätigungen Version 5 lt. DA-ORG. Die Export-Richtung ist „ein“.

6.1.4.16 SART_to_EZSART.profile

Dieses Beispiel importiert DDS-Paketdateien und exportiert SART-basierte Einzeldateien lt. DVS 1.0 siehe 6.1.2.5.

6.1.4.17 SART_UTF-8_to_SART_8859-1.profile

Dieses Beispiel importiert DDS-Paketdateien mit dem Zeichensatz UTF-8 und exportiert DDS-Paketdateien mit dem Zeichensatz ISO 8859-1.

6.1.4.18 SART_ZIP_to_SART_CR_750LL.profile

Dieses Beispiel importiert DDS-Paketdateien und exportiert DDS-Paketdateien. Die exportierten Zeilen werden auf Länge 750 erweitert und der Zeilenumbruch wird von Line Feed auf Carriage Return/ Line Feed umgestellt.

6.1.4.19 XML4_to_XML5.profile

Dieses Beispiel importiert XML-DDS-Paketdatei Version 4 lt. DA-ORG und exportiert XML-DDS-Paketdatei Version 5 lt. DA-ORG.

6.1.4.20 XML5_to_EZBINARY.profile

Dieses Beispiel importiert XML-DDS-Paketdatei Version 5 lt. DA-ORG und exportiert binäre Einzeldateien lt. DVS 1.0 siehe 6.1.2.5.

6.2 dvsTester.{cmd | sh }

Diese Applikation dient zum Testen von DVS-konformen Datenbeständen. Es stellt de facto nur die Importfunktionalität der Applikation „dvsConverter“ dar. Für detailliertere Beschreibungen siehe dvsConverter.{cmd | sh } (6.1).

6.2.1 Synopsis

```
usage: dvsTester.cmd -p <path> [-i <type>] [-ip <path>] [-ic <charset>] [-ia <args>] [-
ipt <pattern>] [-v | -ve]  [-d]

 -p,--profile <path>          Pfad zu Profildatei

 -i,--import <type>           Typ des Imports. Erlaubte Werte sind XML4, XML5, SART,
                              SAMMEL,EZBINARY,EZSART

 -ip,--ipath <path>           Pfad der/des zu importierenden Datei/Verzeichnis

 -ic,--icharset <charset>     Character Set der Importdatei(en). Default ist UTF-8

 -ia,--iargs <args>           Erweiterte Import Argumente (kommasepariert). Abhaengig vom
                              verwendeten Import-Typ. Siehe

                              weiter unten

 -ipt,--ipattern <pattern>    Muster zur Einschraenkung der zu importierenden Datei(en)

 -v,--verbose                 Zeigt detaillierte Import/Export Informationen an

 -ve,--verbose-error          Zeigt detaillierte Import/Export Informationen nur fuer
                              fehlerhafte Pakete/Bestaende an

 -d,--debug                   Zeigt detaillierte Fehlermeldungen an


Importer erweiterte Argumente (-ia):

-------------------------------------------------------------------------------

[SART]

splitData=true|false            SART-basierte Dateien waehrend Import in temporaere
                                 Einzeldateien aufsplitten. Default ist "true"



[SAMMEL]

ZVPA=<2*DIGIT>                  Code des Zielversicherungstraeger (fuer Paket).
                                 Obligatorisch

UVPA=<2*DIGIT>                  Code des Ursprungversicherungstraeger (fuer Paket)

splitData=true|false            SART-basierte Dateien waehrend Import in temporaere
                                 Einzeldateien aufsplitten. Default ist "true"



[EZBINARY]

ZVPA=<2*DIGIT>                   Code des Zielversicherungstraeger (fuer Paket).
                                 Obligatorisch

UVPA=<2*DIGIT>                   Code des Ursprungversicherungstraeger (fuer Paket)

useDirStruct=true|false          Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS
                                 1.0. Default ist "false"
```

```
[EZSART]

ZVPA=<2*DIGIT>                  Code des Zielversicherungstraeger (fuer Paket).
                                Obligatorisch

UVPA=<2*DIGIT>                  Code des Ursprungversicherungstraeger (fuer Paket)

useDirStruct=true|false         Metadaten ergeben sich aus Verzeichnisstruktur lt. DVS
                                1.0. Default ist "false"

lineFlags=true|false            Zeilen des Bestands beginnen mit DA-ORG Kennzeichen.
                                Default ist "false"
```

7 Verschiedenes

7.1 Suchmuster (Pattern)

Die folgenden Regeln werden bei der Auflösung von angegebenen Suchmustern angewandt:

- The * character matches zero or more characters of a name component without crossing directory boundaries.
- The ** characters matches zero or more characters crossing directory boundaries.
- The ? character matches exactly one character of a name component.
- The backslash character (\) is used to escape characters that would otherwise be interpreted as special characters. The expression \\ matches a single backslash and "\{" matches a left brace for example.
- The [ ] characters are a bracket expression that match a single character of a name component out of a set of characters. For example, [abc] matches "a", "b", or "c". The hyphen (-) may be used to specify a range so [a-z] specifies a range that matches from "a" to "z" (inclusive). These forms can be mixed so [abce-g] matches "a", "b", "c", "e", "f" or "g". If the character after the [is a ! then it is used for negation so [!a-c] matches any character except "a", "b", or "c". Within a bracket expression the *, ? and \ characters match themselves. The (-) character matches itself if it is the first character within the brackets, or the first character after the ! if negating.
- The { } characters are a group of subpatterns, where the group matches if any subpattern in the group matches. The "," character is used to separate the subpatterns. Groups cannot be nested.
- Leading period/dot characters in file name are treated as regular characters in match operations. For example, the "*" glob pattern matches file name ".login". The Files.isHidden(java.nio.file.Path) method may be used to test whether a file is considered hidden.
- All other characters match themselves in an implementation dependent manner. This includes characters representing any name-separators.
- The matching of root components is highly implementation-dependent and is not specified.

Beispiele:

- *.zip Matches a path that represents a file name ending in .zip
- *.* Matches file names containing a dot
- *.{zip,gz} Matches file names ending with .zip or .gz
- ddein.? Matches file names starting with ddein. and a single character extension
- /home/*/* Matches /home/gus/data on UNIX platforms
- /home/** Matches /home/gus and /home/gus/data on UNIX platforms
- C:* Matches C:\foo and C:\bar on the Windows platform (note that the backslash is escaped)

8 APPENDICES

8.1 Import Listener entwickeln

Um einen eigenen Import Listener zu entwickeln muss eine Java Klasse erstellt werden (JDK 7 compliant) die das Interface *at.itsv.dvs io.ImportListener* implementiert. Zur Kompilierzeit muss sich dabei das Jar-File *dvs-core-X.Y.Z.jar* (X.Y.Z bezeichnet die aktuelle Version) im Klassenpfad befinden.

Dieses Interface definiert die Funktionen:

public void importFile(String fileName, int fileNo, int fileTotal);

public void importPaket(DVSPaket paket);

public void importBestand(DVSBestand bestand);

Diese Funktionen werden nach dem Importieren (Lesen aus Datei) der Pakete/Bestände aufgerufen. Dabei können die Werte der einzelnen Pakete/Bestände überschrieben werden.

Beispiel:

```
package test.impexp;

import at.itsv.dvs.io.ImportListener;
import at.itsv.dvs.model.DVSBestand;
import at.itsv.dvs.model.DVSPaket;
import at.itsv.dvs.model.InvalidDataException;

public class SampleImportListener implements ImportListener
{

	@Override
	public void importBestand(DVSBestand bestand)
	{
		try
```

```
		{
			bestand.setZusaetzlOrdnungsbegriff("INP-XYZ");
		}
		catch (InvalidDataException ex)
		{
			ex.printStackTrace();
		}
	}

	@Override
	public void importFile(String fileName, int fileNo, int fileTotal)
	{
	}

	@Override
	public void importPaket(DVSPaket paket)
	{
	}
}
```

Diese Java Datei kann nun direkt als erzeugte Klasse oder besser in einer Jar-Datei verpackt in den Klassenpfad der DVS-Toolkit Utilities aufgenommen werden (siehe 4.4.3).

8.2 Export Listener entwickeln

Um einen eigenen Export Listener zu entwickeln muss eine Java Klasse erstellt werden (JDK 7 compliant) die das Interface *at.itsv.dvs io.ExportListener* implementiert. Zur Kompilierzeit muss sich dabei das Jar-File *dvs-core-X.Y.Z.jar* (X.Y.Z bezeichnet die aktuelle Version) im Klassenpfad befinden.

Dieses Interface definiert die Funktionen:

public void exportFile(String fileName) throws DVSExportException;

public void exportPaket(DVSPaket paket) throws DVSExportException;

public void exportBestand(DVSBestand bestand) throws DVSExportException;

Diese Funktionen werden vor dem Exportieren (Schreiben in Datei) der Pakete/Bestände aufgerufen. Dabei können die Werte der einzelnen Pakete/Bestände überschrieben werden. Im Falle einer schweren Ausnahme im Exporter kann durch die Exception *at.itsv.dvs.io. DVSExportException* die Verarbeitung beendet werden.

Beispiel:

```
package test.impexp;

import java.util.List;

import at.itsv.dvs.io.DVSExportException;
import at.itsv.dvs.io.ExportListener;
```

```
import at.itsv.dvs.model.DVSBestand;
import at.itsv.dvs.model.DVSPaket;
import at.itsv.dvs.model.InvalidDataException;

public class SampleExportListener implements ExportListener
{
	@Override
	public void exportBestand(DVSBestand bestand) throws DVSExportException
	{
	}

	@Override
	public void exportFile(String fileName) throws DVSExportException
	{
	}

	@Override
	public void exportPaket(DVSPaket paket) throws DVSExportException
	{
		Integer i=1;
		List<DVSBestand> lb = paket.getBestandListe();
		for(DVSBestand bestand:lb)
		{
			try
			{
				bestand.setZusaetzlInfo("ZUSINFO" + i++);
			}
			catch (InvalidDataException ex)
			{
				throw new DVSExportException(ex);
			}
		}

	}
}
```

Diese Java Datei kann nun direkt als erzeugte Klasse oder besser in einer Jar-Datei verpackt in den Klassenpfad der DVS-Toolkit Utilities aufgenommen werden (siehe 4.4.3).

8.3 Referenz ID Generator entwickeln

Um einen eigenen Generator für Referenz IDs zu entwickeln muss eine Java Klasse erstellt werden (JDK 7 compliant) die das Interface *at.itsv.dvs.util.refno.DVSReferenzNummerGenerator* implementiert. Zur Kompilierzeit muss sich dabei das Jar-File *dvs-core-X.Y.Z.jar* (X.Y.Z bezeichnet die aktuelle Version) im Klassenpfad befinden.

Dieses Interface definiert die Funktion:

public DVSReferenzNummer generateReferenzNummer(DVSBestand bestand)

Diese Funktion muss ein Objekt vom Typ *at.itsv.dvs.util.refno.DVSReferenzNummer* zurückgeben. Falls der übergebene String null ist oder länger als 12 Zeichen wird eine Exception vom Typ *at.itsv.dvs.util.refno.ReferenzNummerException* geworfen.

Der übergebene Parameter *DVSBestand bestand* verweist dabei auf eine Kopie des aktuell verarbeiteten Bestandsobjekts.

Beispiel:

Die folgende Klasse erzeugt eine randomisierte Referenz ID mithilfe der Klasse *org.apache.commons.lang3.RandomStringUtils* aus dem Paket Apache commons-lang (siehe http://commons.apache.org/proper/commons-lang)

```
package refgen;


import org.apache.commons.lang3.RandomStringUtils;

import at.itsv.dvs.model.DVSBestand;
import at.itsv.dvs.util.refno.DVSReferenzNummer;
import at.itsv.dvs.util.refno.DVSReferenzNummerGenerator;
import at.itsv.dvs.util.refno.ReferenzNummerException;

public class RandomRefNoGenerator implements DVSReferenzNummerGenerator

{
	@Override

	public DVSReferenzNummer generateReferenzNummer(DVSBestand bestand)

		throws ReferenzNummerException

	{

		return new
		DVSReferenzNummer(RandomStringUtils.randomAlphanumeric(12).toUpperCase());

	}

}
```

Diese Java Datei kann nun direkt als erzeugte Klasse oder besser in einer Jar-Datei verpackt in den Klassenpfad der DVS-Toolkit Utilities aufgenommen werden (siehe 4.4.3). Für dieses Beispiel muss zusätzlich auch die Jar-Datei „commons-lang3-X.Y.Z.“ (http://commons.apache.org/proper/commons-lang) in diesen Klassenpfad aufgenommen werden.