PiSA sales Dokumentation:

Erweiterter CSV-Import

Stand: 08.05.2012

Erstellt durch: PiSA sales GmbH
D-14169 Berlin, Fredericiastraße 17-19
Tel.: + 49 (0)30/81 07 00 –00
Fax.: + 49 (0)30/81 07 00 – 99
E-Mail: info@pisasales.de

Verwendung einer Klasse als Userexit für einen CSV-Import

Hiermit wird die Möglichkeit bereitgestellt, alle durch CSV-Import erzeugte oder geänderte Datensätze automatisch in ein Replikationspaket aufzunehmen.

Um das zu realisieren, wurde der CSV-Import erweitert. Es ist jetzt möglich eine Klasse im Server (Jar) oder im Repositiory (Dynamisches Java) als Userexit für einen CSV-Import zu verwenden. Die Klasse wird als Parameter in der CSV-Datei angegeben. Der neue Parameter ist **U**.

Danach folgt die Klasse als Verweis in den Server (z.B. als de.pisa.psa.csv.CSVusx - geht nicht wirklich, weil CSVusx nur als Basisklasse verwendet werden kann) oder mit $ für ein dynamisches Java (z.B. $CSV_USX_TST).

Die Klasse muss von CSVusx abgeleitet werden.

Beispiel:

```
public class CsvUsxTst extends de.pisa.psa.csv.CSVusx {
	PscSsn ssn = null;

	/**
	 * Constructor
	 * @param ssn
	 * @param mta
	 */
	public CsvUsxTst (PscSsn ssn, CSVmeta mta) {
		super(ssn, mta);
		this.ssn = ssn;
		this.ssn.wriTxt("TEST: Constructor");
	}

	/**
	 * @see de.pisa.psa.csv.CSVusx#cls()
	 */
	@Override
	public void cls() {
		super.cls();
		this.ssn.wriTxt("TEST: cls");
	}

	/**
	 * @see de.pisa.psa.csv.CSVusx#preImp(de.pisa.psa.csv.CSVimp)
	 */
	@Override
	public void preImp(CSVimp imp) throws Exception {
		this.ssn.wriTxt("TEST: preImp");
	}

	/**
	 * @see de.pisa.psa.csv.CSVusx#pstImp(de.pisa.psa.csv.CSVimp)
	 */
	@Override
	public void pstImp(CSVimp imp) throws Exception {
```

```
            this.ssn.wriTxt("TEST: pstImp");
        }

        /**
         * @see de.pisa.psa.csv.CSVusx#preFet(de.pisa.psc.srv.dto.PscDto)
         */
        @Override
        public void preFet(PscDto dto) throws Exception {
            this.ssn.wriTxt("TEST: preFet");
        }

        /**
         * @see de.pisa.psa.csv.CSVusx#pstFet(de.pisa.psc.srv.dto.PscDto)
         */
        @Override
        public void pstFet(PscDto dto) throws Exception {
            this.ssn.wriTxt("TEST: pstFet");
        }

        /**
         * @see de.pisa.psa.csv.CSVusx#prePut(de.pisa.psc.srv.dto.PscDto)
         */
        @Override
        public void prePut(PscDto dto) throws Exception {
            this.ssn.wriTxt("TEST: prePut");
        }

        /**
         * @see de.pisa.psa.csv.CSVusx#pstPut(de.pisa.psc.srv.dto.PscDto)
         */
        @Override
        public void pstPut(PscDto dto) throws Exception {
            this.ssn.wriTxt("TEST: pstPut");
        }
}
```

Die einzelnen Funktionen in denen man eingreifen kann sind:

1. Der Konstruktor: Initialisierung der Klasse wenn benötigt

Bitte beachten! Es werden zwei Instanzen der Klasse erzeugt. Eine nur zum Zählen der Zeilen - dafür kommen keine weiteren Aufrufe. Eine wenn der Import der Daten erfolgt - mit Aufruf der Call-Back-Funktionen. Auch cls() wird nur für die zweite Instanz aufgerufen. Wichtige Ressourcen sollten daher erst im preImp() geöffnet werden.)

2. Bereinigung `cls()`: Aufräumen und freigeben von Ressourcen - wird während der Freigabe der Ressourcen des Imports aufgerufen, auch wenn eine Fehler während des Imports aufgetreten ist.

3. `preImp()` / `pstImp()`: Wird vor und nach dem Import aufgerufen

Bitte beachten! `pstImp()` wird, falls ein Fehler aufgetreten ist, möglicherweise nicht mehr vom Import aufgerufen. `cls()` hingegen schon!)

4. `preFet(PscDto dto) / pstFet(PscDto dto)`: Wird vor und nach Laden der Daten aus der DB aufgerufen. Kann verwendet werden, um die Suche oder das Ergebnis zu manipulieren.

5. `prePut(PscDto dto) / pstPut(PscDto dto)`: Wird vor und nach Speichern der Daten in die DB aufgerufen.