PiSA sales Documentation:

Extended CSV import

State: 08.05.2012

PiSA sales GmbH
D-14050 Berlin, Fredericiastraße 17-19
phone.: + 49 (0)30/81 07 00 –00
fax.: + 49 (0)30/81 07 00 – 99
e-mail: info@pisasales.de

Usage of a class as an userexit for a CSV import

Herewith the possibility is provided to record all or modified records generated by CSV import automatically in a replication package.

To realize this, the CSV import was extended. Now it is possibly to use a class in the server (Jar) or in the Repositiory (Dynamic Java) as an userexit for a CSV import. The class is stated as parameter in the CSV file. The new parameter is U.

Then the class follows as a reference in the server (e.g., as de.pisa.psa.csv. CSVusx - does not operate really because CSVusx can be used only as a base class) or with $ for dynamic Java (e.g., $CSV_USX_TST).

The class must be derived from CSVusx.

Example:

```
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");
	}
}
```

The single functions in those one can intervene are:

1. The constructor: Initiaöizing oft he class if required

Please, notice! Two instances of the class are generated. Only one to count of the rows - for it there come no further callings. Another when the import of the data happens - with call of the call back functions. Also cls () is called on only for the second instance. Hence, important resources should be opened only in preImp ().)

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

Cleaning out and release of resources - it is called on during the release of the resources of the import, even if an error has appeared during the import.

3. `preImp() / pstImp()`: Is called before and after the import

Please notice! `pstImp()` is, if an error occurs, possibly no more caled on by the import. `cls()`however does!)

4. **`preFet(PscDto dto) / pstFet(PscDto dto)`**: It is called on before and after loading of the data from the database. It can be used to manipulate the query or the result.

5. **`prePut(PscDto dto) / pstPut(PscDto dto)`:** Is called on before and after storing the data into the database.