SEPA Documentation

SEPA Processor Manual

PAYMENTCOMPONENTS *Accessible Expertise, Empowering Great Solutions*

SEPA CT processor User Manual • A java-bean library for SEPA messages SEPACTprocessor 6.0 Web http://www.paymentcomponents.com e-mail info@paymentcomponents.com Phone +30 210 61 45 050 • Fax +30 210 61 45 055

Overview

Introduction

SEPA processor is a java library. It contains several JavaBeans that provide assistance in building, validating and editing SEPA messages or extracting information from them.

SEPA is built on a subset of the **UNIFI** "Payments Clearing and Settlement" and "Payments Initiation" messages. **UNIFI**, which stands for UNIversal Financial Industry message scheme is an International Standard (ISO 20022) which aims to provide the financial industry with a common platform for the development of messages in a standardized XML syntax. Using a modeling methodology (based on UML) to capture in a syntax-independent way financial business areas, business transactions and associated message flows. It also provides a set of XML design rules to convert the messages described in UML into XML schemas.

SEPA processor includes all of the "Payments Clearing and Settlement" messages defined in the EPC SEPA CT and DD Rulebook, as well some customized flavours like EBA. See appendix A for the list of included messages.

The following document is organized in a number of sections.

In Section 2 we describe the procedure a user must follow in order o build a new SEPA message using SEPA processor JavaBeans.

In Section 3 we describe the procedure a user must follow in order to edit a preexistent SEPA message. By "edit" we mean adding new information to the message, deleting information from the message or altering information of the message. By "preexistent" we mean a message that is stored in a File as XML, or in a character String, or in a SEPA processor JavaBean.

In Section 4 we describe the procedure a user must follow in order to extract information from an existent SEPA message that has been converted to a SEPA processor JavaBean.

In Section 5 we describe the procedure a user must follow in order to create the reply message of an existing SEPA message.

Section 6 covers the installation and usage of the evaluation version.

Finally in Section 7 we summarize the most important methods and code fragments, for a quick reference.

Usefulness of SEPA processor

SEPA was born from the need for more structured and well formed communication messages for financial institutions and has been built on a subset of the XML based UNIFI (ISO 20022) messages. Unlike the previous standard, ISO 15022, where the messages were formed in a difficult and non standard format, SEPA (ISO 20022), brings the advantages of XML to the financial institution focused software developer.

This new XML based standard, enables software developers to choose among a plethora of free or commercial tools and runtime libraries for handling the messages. Therefore, one could think that SEPA processor is nothing more than just another XML handling tool. However, this is not the case.

It is true that SEPA processor performs all of the tasks of the XML tools, i.e. it can parse an XML file, edit some elements in it, delete or add elements, validate it against an XML schema and finally export the message as XML file or character String. In addition there are many more actions that a complete SEPA message handling tool must be capable of doing apart from the common XML actions listed above. The most important being the message validation beyond the XML schema. Each SEPA message enforces many "Rules" as they are called in the SEPA literature. These Rules are described in detail in the definition of each SEPA message. Hard coding in some programming language is needed.

Another advantage of SEPA processor is its simplicity of usage. It is easily integrated with your development environment and every action is carried out by simple calls.

The following list summarizes the key advantages of the SEPACTprocessor.

- Simple integration with your development environment.
- Simplicity in the use of the actions applied to the messages.
- Validation of "Rules" defined for each SEPA message.
- Explanatory error reporting.

Changelog

What's new

- SEPA Processor has been rebuilt from scratch. It is now independent of the 3rd party XML Beans library by using the functionality of JAXB. You only need this jar. No other dependencies/3rd party jars needed anymore.
- New version still has the 2 of 3 ways of building/altering a SEPA message:
- Using generated Java classes
- Using method `setElement("PATH_OF_THE_ELEMENT`, VALUE)

We don't support anymore the third approach: `set_X_Y_Z` method where `X_Y_Z` is the path of the element. i.e. `set_GrpHdr_MsgId`.

- You can now read a message with the following methods:

```
parseAndValidate(String xmlFilePath)
parseAndValidateString(String contentOfMessage)
parseAndValidate(File file)
```

- Packaging has also changed. Due to the fact that our library supports variations of SEPA messages, different versions are built so to organize them, classes representing SEPA messages are located in package `gr.datamation.sepa.core.messages.XXX`. This version includes the EBA-related ones - `gr.datamation.sepa.core.messages.eba`.
- You can use SEPA CT 6.0 with Java 6, 7 or 8.

Backwards compatibility

For anyone using the old version of SEPA Processor and trying to upgrade to version 6.0, some functionality has changed. This is due to the fact that the main XML processing library has changed (XML Beans has been replaced by JAXB). A very common example is when having multiple instances accepted via schema definition. XML Beans handles these using arrays whereas JAXB is using `Lists`. You need to convert your old programs to be compatible with this new version where applicable.

Building a new SEPA message

All SEPA messages are identified by a code id and a name. The code id looks like this `FIToFICstmrCdtTrf` and is located in the `.xsd` file describing the XML schema of the current message. The name of the message is more descriptive and looks like `FIToFICustomerCreditTransfer`. Both the name and the code id of the message are available in the SEPA messages catalogue.

For every SEPA message there is one `SEPA processor` JavaBean. The name of the JavaBean is identical to the name of the message. For example the name of the JavaBean for the message named `FIToFICustomerCreditTransfer` is `FIToFICustomerCreditTransfer`. There are three steps the user must follow in order to build a new SEPA message:

1. Initialize the JavaBean corresponding to the message.
2. Add data to the JavaBean.
3. Validate the message.

In the following three paragraphs we describe in detail those three steps.

Initialize the class corresponding to the message

Initialization of the class is as simple as initializing any class in Java. For the example message we are using here (`FIToFICustomerCreditTransfer`) the initialization would be

```
FIToFICustomerCreditTransfer message = new FIToFICustomerCreditTransfer();
```

The above command will initialize a class for this message named `FIToFICustomerCreditTransfer` which is initially empty.

Add data to the class

The next step is to add data to the new message. In order to add some data to the message the user must know which element in the message tree he wants to add. The element he wants to add is identified by an XML path. The value of the element the user wants to add may be a `String`, a `Boolean` or a complex type that is described in the SEPA type's catalogue.

So to enter some data into the message, the user must call the following method

```
message.setElement(path, value);
```

Where the path argument is a `String` identifying the element to add and the value argument is an `Object`. Or by using the Java generated classes.

Validate the message

After building a SEPA message using the appropriate SEPA `processor` class, the user may want to validate this message. Of course validation is not mandatory but is the only way to prove that the message is correct. Validation is performed by calling the validate() method of the class and internally is a two step process.

1. Validation against the schema of the message in order to ensure that the message is a well formed instance of it.
2. Validation against any Validation Rule as described for that message by the SEPA rules.

To perform those two levels of validation the use must call the following method

```
message.validate();
```

The `validate()` method performs two level validation on the message and fills a list that contains the validation errors that may occur. Each error is contained in the `ArrayList` as a `ValidationEvent` class describing the error. User can use `hasValidationErrors()` method to check for validity of message, or directly by checking the errors list with `getErrors()` method.

A more complex example

We will assume that the SEPA message named `FIToFICustomerCreditTransfer` with code id `FIToFICstmrCdtTrf` has an XML schema described by the following code. This code is just a fragment of the original schema description for the message `FIToFICustomerCreditTransfer` and of course the following description is not the description for the real `FIToFICustomerCreditTransfer` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="urn:iso:std:iso:20022:tech:xsd:pacs.008.001.02"
                  elementFormDefault="qualified" targetNamespace="urn:iso:std:iso:20022:tech:xsd:pacs.
008.001.02">
        <xs:element name="Document" type="Document"/>
        <xs:complexType name="Document">
                <xs:sequence>
                        <xs:element name="FIToFICstmrCdtTrf" type="FIToFICustomerCreditTransferV02"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="FIToFICustomerCreditTransferV02">
                <xs:sequence>
                        <xs:element name="GrpHdr" type="GroupHeader19"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="GroupHeader19">
                <xs:sequence>
                        <xs:element name="MsgId" type="Max35Text"/>
                        <xs:element name="CreDtTm" type="ISODateTime"/>
                        <xs:element name="NbOfTxs" type="Max15NumericText"/>
                </xs:sequence>
        </xs:complexType>
        <xs:simpleType name="Max35Text">
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="35"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="ISODateTime">
                <xs:restriction base="xs:dateTime"/>
        </xs:simpleType>

        <xs:simpleType name="Max15NumericText">
                <xs:restriction base="xs:string">
                        <xs:pattern value="[0-9]{1,15}"/>
                </xs:restriction>
        </xs:simpleType>
</xs:schema>
```

If we represent this schema description as a tree it would look like this

- `FIToFICstmrCdtTrf` (1, 1)

- GrpHdr (1, 1)
- MsgId (1, 1)
- CreDtTm (1, 1)
- NbOfTxs (1, 1)
- Document (1, 1)

The (1, 1) next to each element name means that this element must occur at least 1 time and max 1 time. In other words this element must occur once and only once. In the same way that (0, 2) would mean that the element can occur 0, 1 or 2 times. Note that the tree leave nodes are the simpleType elements of the schema.

One XML path for the previous schema would be /Document/FIToFICstmrCdtTrf/GrpHdr

another one would be /Document/FIToFICstmrCdtTrf/GrpHdr/MsgId

another one would be /Document/FIToFICstmrCdtTrf/GrpHdr[1]/CreDtTm[1].

Notice that the path is free to contain location predicates. This means that we can refer to the first GrpHdr element as /Document/pacs.008.002.02 /GrpHdr[1]/ and to the second GrpHdr element as /Document/pacs.008.002.02/GrpHdr[2]/. If the location predicate is missing then it is implied that the path refers to the first such element.

A sample XML describing a message of our FIToFICustomerCreditTransfer would be

```
<?xml version="1.0" encoding="UTF-8"?>
<Document xmlns="urn:iso:std:iso:20022:tech:xsd:pacs.008.002.02">
        <FIToFICstmrCdtTrf>
                <GrpHdr>
                        <MsgId>messageNo555</MsgId>
                        <CreDtTm>2006-01-01T23:10:00</CreDtTm>
                        <NbOfTxs>4</NbOfTxs>
                </GrpHdr>
        </FIToFICstmrCdtTrf>
</Document>
```

Let's now try to build a FIToFICustomerCreditTransfer message containing the above data using the SEPACTprocessor class. We have seen how to initialize the appropriate class. After the initialization we can enter data. The paths needed are the following

- /Document/FIToFICstmrCdtTrf/GrpHdr/MsgId
- /Document/FIToFICstmrCdtTrf/GrpHdr/CreDtTm
- /Document/FIToFICstmrCdtTrf/GrpHdr/NbOfTxs

Every path of every SEPA message starts with /Document/xxxxxxxxxxx and so we integrated this prefix in every message in order to make the paths the user enters simpler. In other words the paths that the user must add is now

- /GrpHdr/MsgId
- /GrpHdr/CreDtTm
- /GrpHdr/NbOfTxs

The user now can add the appropriate data by calling the following methods

```
message.setElement("/GrpHdr/MsgId", anObject1);
message.setElement("/GrpHdr/CreDtTm", anObject2);
message.setElement("/GrpHdr/NbOfTxs", anObject3);
```

Notice that each call of the setElement method takes as arguments a String identifying the path of the element and an Object containing the value for the element. We mentioned before that this Object can be a String, a Boolear or a more complex type.

There are two ways for the user to know which type must use for a specific element. The first way is from the SEPA reference of the message. This reference is a comprehensive description of the elements of the message and the corresponding types. The second way is by calling the getElementTyp e() method of the message.

By using either method the user will find out that the elements and the corresponding types for the message he/she wants to build are:

- /GrpHdr/MsgId - Max35Text
- /GrpHdr/CreDtTm - ISODateTime
- /GrpHdr/NbOfTxs - Max15NumericText

So the user will make the following calls

```
message.setElement ("/GrpHdr/MsgId", "MESSAGEID");
message.setElement ("/GrpHdr/CreDtTm", Calendar.newInstance());
message.setElement ("/GrpHdr/NbOfTxs", "1");
```

Another way to build the message is to find the type of the `/GrpHdr` element and then constructing an `Object` of this type, filling it with the appropriate data and setting it as value of the `/GrpHdr` element by calling the method

```
message.setElement ("/GrpHdr", aGrpHdrTypeObject);
```

Note that if you set data to an element that has already been set then the element will be overwritten. You can also set the current object via the following method

```
message.getRootElement().setGrpHdr(aGrpHdrTypeObject);
```

The method `setElement` throws an `Exception` of type `Error` if something goes wrong during the operation of setting data on an element. There are two reasons for something going wrong at this stage. The first is when the path entered by the user as the first argument of the `setElement` method is invalid. The second is when the data set for an element is invalid. In either case the operation is aborted and an Error Exception is thrown. The user must catch this `Error` object and get the message from it in order to identify the error. For example a sample code would look like

```
try {
    message.setElement ("some_path", anObject);
} catch (Error e){
    System.err.println (e.getMessage());
    System.exit(1);
}
```

Editing a SEPA message

In the previous section we saw how you can create a new SEPA message from scratch using SEPACTprocessor JavaBeans. In this section we will see how the user is able to edit an existing SEPA message. By "editing" we mean either altering a piece of information in the message, or deleting some, or adding new.

Note that all this procedures apply also to the case where the user creates a new message. After the user creates the new message, following the procedures in Section 4, he is able to alter, delete, or add information to the message.

So in this section we assume that there is a SEPA message. The format of the message varies. It may have been just created by the user following the procedures of the previous section. It may also have been created in the past and saved in an XML file. Or it may have been created by another part of the user's software and it is available as a String. So a SEPA message may exist as a File, as a String, or as a SEPACTprocessor JavaBean.

The procedures we will describe in this section apply only to the SEPACTprocessor form of an existing SEPA message. The user must load the File or String version of a message into a SEPACTprocessor JavaBean in order to be able to alter, delete or add data to it.

Fortunately the SEPACTprocessor framework provides the tools to do this conversion. By "conversion" we mean loading a SEPA message from a File or a String into the equivalent SEPACTprocessor class.

Loading

We will assume that the user has a SEPA message saved as XML in an XML File called "message.xml". The procedures we will describe apply also to the case where the message is in a String form.

The loading of a message is done via the `parseAndValidate` method of SEPACTprocessor JavaBean of the corresponding message. This method takes a single argument. The argument may be a `File`, or a path `String`. Alternatively you can use method `parseAndValidateString` that takes a `String` representation of a message as argument. It is obvious which one to call when a user has a message in a `File` or a `String` format.

Continuing the example we will assume that the user has in the message.xml file a SEPA message of code `pacs.008.001.02`. Searching through the SEPA documentation he will find that the name of this message with code `pacs.008.001.02` is `FIToFICustomerCreditTransfer`.

Therefore to load the message in the message.xml file he must execute the following commands.

```
FIToFICustomerCreditTransfer message = new FIToFICustomerCreditTransfer();
message.parseAndValidate(new File("/path/to/message.xml"));
```

These commands must be surrounded with try-catch expression, so if something goes wrong during the loading, the user will be informed. The procedure is identical in the case where the message is in a String format. The user must just replace the `parseAndValidate(File f)` method with the `parseAndValidateString(String s)`.

Editing

After loading a SEPA message in a SEPA processor JavaBean, a user is able to apply any editing action, like deleting elements, adding new, or altering the existing ones.

Adding

Addition to a loaded message is done just like adding new information while building a new SEPACTprocessor message, as we have seen in the Section 4 describing the creation of a new SEPACTprocessor message.

Altering

Altering a piece of information in a SEPACTprocessor message is done just like adding new ones. Let's assume that a user wants to alter the MsgId of the first GroupHeader element of a FIToFICustomerCreditTransfer message and set the value "newId". This is done as follows:

```
FIToFICustomerCreditTransfer message = new FIToFICustomerCreditTransfer();
message.parseAndValidate(new File("message.xml"));
message.setElement("/GrpHdr/MsgId", "NEWMESSAGEID");
```

Extracting data

Extracting data from a SEPA message is as simple as calling one line of code. It is assumed that previously the message is loaded to a SEPACTprocessor JavaBean object or it has been built. The following code fragment shows how easy it is to get a piece of information from a message.

```
FIToFICustomerCreditTransfer message = new FIToFICustomerCreditTransfer();
message.parseAndValidate(new File("/path/to/message.xml"));
Object data = message.getElement("/GrpHdr");
```

`Of course the user is responsible to cast this Object, the data object, to the appropriate structure. The alternative version of getElement can be used as follows:`

```
FIToFICustomerCreditTransfer message = new FIToFICustomerCreditTransfer();
message.parseAndValidate(new File("/path/to/message.xml"));
S2SCTGroupHeader33 data = message.getRootMessage().getGrpHdr();
```

In addition to the getElement methods, the `resolveTransaction() or resolveTransactions()` methods can be used to greatly simplify the parsing of messages. The following uses a FIToFICustomerCreditTransfer (`pacs.008.001.02`) message as an example. These methods return vectors of vectors of `LeafInfo` objects.

`LeafInfo` objects have 3 fields:

- `fieldPath`
- `fieldcd`
- `value`

So for the following path in XML `/GrpHdr/InitgPty/Nm` with value `"ABC Corporation"` the fields would be populated as follows:

fieldPath	/GrpHdr/InitgPty
fieldcd	Nm
value	ABC Corporation

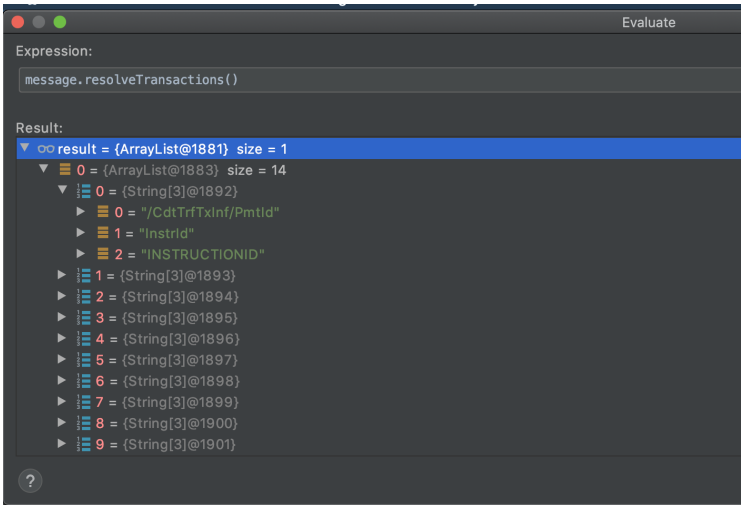
The following demonstrates the 14 `String` array objects which will be stored in a list. The `resolveTransactions` method searches for CdtTrfTxInf elements in the file, and returns a list of lists of LeafInfo for each CdtTrfTxInf element A sample output is as follows:

FIELD PATH	FIELD NAME	VALUE
/CdtTrfTxInf/PmtId	InstrId	INSTRUCTIONID
/CdtTrfTxInf/PmtId	TxId	TXID
/CdtTrfTxInf/PmtId	EndToEndId	NOTPROVIDED
/CdtTrfTxInf/PmtTpInf/SvcLvl	Cd	SEPA
/CdtTrfTxInf	IntrBkSttlmAmt	20
/CdtTrfTxInf	ChrgBr	SLEV
/CdtTrfTxInf/Dbtr	Nm	DEBTOR NAME
/CdtTrfTxInf/Dbtr/Id/OrgId	BICOrBEI	AAAADEFFXXX
/CdtTrfTxInf/DbtrAcct/Id	IBAN	DE36500400000586060600
/CdtTrfTxInf/DbtrAgt/FinInstnId	BIC	COBADEFFXXX
/CdtTrfTxInf/CdtrAgt/FinInstnId	BIC	CYDBCY2N
/CdtTrfTxInf/Cdtr	Nm	CREDITOR NAME
/CdtTrfTxInf/Cdtr/Id/OrgId	BICOrBEI	CYDBCY2NXXX
/CdtTrfTxInf/CdtrAcct/Id	IBAN	CY21003000700000007011014888

Examples of the resolve method include:

```
List<List<String[]>> listGH = message. resolveTransactions();
```

Example code demonstrating the use of these methods can be found in the 'working with the evaluation' section.



Building the Reply Message

The Reply Message (R-Transactions, RT) is a SEPA message that is used to provide an automatic response to a message. An RT can be constructed and modified according to the following instructions. This feature is designed to simplify the creation of an RT since it can be automatically constructed by following the specification of the initiating message. The construction of the RT is defined by an internal mechanism whose prime purpose is to:

1. Transfer "common" data blocks (or reusable information) from the original message to the RT.
2. Create the required message blocks depending on whether the RT message applies to all the transactions of the original message or not.

The resulting RT contains all the "reusable" data extracted from the initial message and only requires the addition of any further necessary data to complete the creation of the RT. *Note that: until all mandatory data has been added, the RT object will not be able to produce a valid message.* The construction of an RT is implemented in the class of the respective SEPA message, via the autoReply(String, boolean) method. The first parameter specifies the unique code of the RT (since an existing message could have more than one possible RT) whereas the second designates whether the RT applies to all transactions or not. For example, assuming that:

1. the variable named "fiToFi" holds a valid FIToFICustomerCreditTransfer message (pacs.008.001.02).
2. the PaymentReturn message (pacs.004.001.02) is one of the possible replies of the FIToFICustomerCreditTransfer message and finally
3. the RT does not refer to all transactions declared in the FIToFICustomerCreditTransfer. Then, the calling of the autoReply method using the following code fiTofi.autoReply("pacs.004.001.02", false) will return a PaymentReturn object.

```
        FIToFICustomerCreditTransfer fiToFi = new FIToFICustomerCreditTransfer();

        try {
            fiToFi.setElement("/GrpHdr/MsgId", "MESSAGEID");
            fiToFi.setElement("/GrpHdr/IntrBkSttlmDt", Calendar.getInstance());
            fiToFi.setElement("/GrpHdr/NbOfTxs", "1");

            S2SCTCurrencyAndAmount amt = new S2SCTCurrencyAndAmount();
            amt.setValue(new BigDecimal(20));
            amt.setCcy(S2SCTCurrencyCode.EUR);
            fiToFi.setElement("/GrpHdr/TtlIntrBkSttlmAmt", amt);

            fiToFi.setElement("/GrpHdr/SttlmInf/SttlmMtd", "CLRG");
            fiToFi.setElement("/GrpHdr/SttlmInf/ClrSys/Prtry", "ST2");
            fiToFi.setElement("/GrpHdr/InstdAgt/FinInstnId/BIC", "MILBGRAA");

            //message.setElement("/CdtTrfTxInf/XchgRate",                   BigDecimal.ONE);
            fiToFi.setElement("/CdtTrfTxInf/PmtId/InstrId", "INSTRUCTIONID");
            fiToFi.setElement("/CdtTrfTxInf/PmtId/EndToEndId", "NOTPROVIDED");
            fiToFi.setElement("/CdtTrfTxInf/PmtId/TxId", "TXID");
            fiToFi.setElement("/CdtTrfTxInf/PmtTpInf/SvcLvl/Cd", "SEPA");

            CreditTransferTransactionInformation11.IntrBkSttlmAmt intrBkSttlmAmt = new
CreditTransferTransactionInformation11.IntrBkSttlmAmt();
            intrBkSttlmAmt.setValue(new BigDecimal(20));
            intrBkSttlmAmt.setCcy(S2SCTCurrencyCode.EUR);

            fiToFi.setElement("/CdtTrfTxInf/IntrBkSttlmAmt", intrBkSttlmAmt);
            fiToFi.setElement("/CdtTrfTxInf/ChrgBr", "SLEV");

            fiToFi.setElement("/CdtTrfTxInf/Dbtr/Nm", "DEBTOR NAME");
            fiToFi.setElement("/CdtTrfTxInf/Dbtr/Id/OrgId/BICOrBEI", "AAAADEFFXXX");
            fiToFi.setElement("/CdtTrfTxInf/DbtrAcct/Id/IBAN", "DE36500400000586060600");
            fiToFi.setElement("/CdtTrfTxInf/DbtrAgt/FinInstnId/BIC", "COBADEFFXXX");

            fiToFi.setElement("/CdtTrfTxInf/Cdtr/Nm", "CREDITOR NAME");
            fiToFi.setElement("/CdtTrfTxInf/Cdtr/Id/OrgId/BICOrBEI", "CYDBCY2NXXX");
            fiToFi.setElement("/CdtTrfTxInf/CdtrAcct/Id/IBAN", "CY21003000700000007011014888");
            fiToFi.setElement("/CdtTrfTxInf/CdtrAgt/FinInstnId/BIC", "CYDBCY2N");

            fiToFi.resolveTransactions();
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println(e.getMessage());
            System.exit(1);
        }

        fiToFi.validate();
        System.err.println("Message is " + (!fiToFi.hasValidationErrors() ? "valid." : "invalid."));

        if (fiToFi.hasValidationErrors()) {
            fiToFi.printValidationErrors();
        }
```

```
        //Initialize a payment return (pacs.004)
        PaymentReturn paymentReturn = new PaymentReturn();
        //message id is commonly auto-generated by each system
        paymentReturn.setElement("/GrpHdr/MsgId", "OURMESSAGEID");
        //The settlement date for our return
        paymentReturn.setElement("/GrpHdr/IntrBkSttlmDt", Calendar.getInstance());
        //In most cases this is set to CLRG...
        paymentReturn.setElement("/GrpHdr/SttlmInf/SttlmMtd", "CLRG");
        //and Clearing system is set to ST2
        paymentReturn.setElement("/GrpHdr/SttlmInf/ClrSys/Prtry", "ST2");
        //Construct the message reply information
        //Here we are returning a message with reason code AC01 and with no charges
        //Charges have effect only for reason code FOCR (acceptance of Request for Recall)
        Vector<MsgReplyInfo> msgReplyInfo = new Vector<MsgReplyInfo>();
        ReasonCode rc = new ReasonCode(ReasonCode.CD, "AC01", null, null);
        //The message reply info has the transaction Id of the payment we want to return, the reason code info,
a return Id we generate,
        //an instruction id and the settlement date of the payment that we return
        MsgReplyInfo mri = new MsgReplyInfo("TXID", rc, MsgReplyInfo.MSGID_BASED, "RETURNID", "", new Date());
        msgReplyInfo.add(mri);
        paymentReturn = (PaymentReturn) fiToFi.autoReply(paymentReturn, msgReplyInfo);

        System.out.println(paymentReturn.toString());
```

Working with the evaluation

The evaluation version of SEPA processor is fully functional but is limited in the number of different ISO20022 messages it can handle. The evaluation version is capable of processing SEPA messages of `pain.002.001.03` with the type `FIToFIPmtStsRptS2` and name `FIToFIPaymentStatusReport`. The user can find the full description of this message, as with all other ISO20022 based messages, in the official web site of the ISO20022 organization available at the following URL. www.iso20022.org. Additional SEPA specific information is available on the EPC's website at the following URL. http://www.europeanpaymentscouncil.eu/ Using the evaluation, a developer can execute all the actions described in this manual. Namely parse, build, edit and extract, but only when the message is of the type `FIToFIPmtStsRptS2` with a schema designation of `pacs.002.001.03S2`. The evaluation includes the following files that need to be copied to your projects lib directory and then imported into the project.

Main evaluation library:

- sepa-ct-x.x.x-demo.jar

The following sample code will provide a quick start to test that the environment is configured correctly.

```
import gr.datamation.sepa.core.messages.eba.pacs.FIToFIPaymentStatusReport;
import org.junit.Assert;
import org.junit.Test;

public class Pacs002ParserValidator {

    @Test
    public void testParseValidate() {
        FIToFIPaymentStatusReport t = new FIToFIPaymentStatusReport();
        t.parseAndValidateString("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
                "<Document xmlns=\"urn:iso:std:iso:20022:tech:xsd:pacs.002.001.03S2\">\n" +
                "\t<FIToFIPmtStsRptS2>\n" +
                "\t\t<GrpHdr>\n" +
                "\t\t\t<MsgId>0006936999</MsgId>\n" +
                "\t\t\t<CreDtTm>2010-02-23T14:29:02.0Z</CreDtTm>\n" +
                "\t\t\t<InstgAgt>\n" +
                "\t\t\t\t<FinInstnId>\n" +
                "\t\t\t\t\t<BIC>BCYPCY2N</BIC>\n" +
                "\t\t\t\t</FinInstnId>\n" +
                "\t\t\t</InstgAgt>\n" +
                "\t\t</GrpHdr>\n" +
                "\t\t<OrgnlGrpInfAndSts>\n" +
                "\t\t\t<OrgnlMsgId>BCYPCY2N201002231526046860001</OrgnlMsgId>\n" +
                "\t\t\t<OrgnlMsgNmId>pacs.004</OrgnlMsgNmId>\n" +
                "\t\t\t<OrgnlNbOfTxs>4</OrgnlNbOfTxs>\n" +
                "\t\t\t<OrgnlCtrlSum>1200</OrgnlCtrlSum>\n" +
                "\t\t\t<GrpSts>ACCP</GrpSts>\n" +
                "\t\t\t<StsRsnInf>\n" +
                "\t\t\t\t<Orgtr>\n" +
                "\t\t\t\t\t<Id>\n" +
                "\t\t\t\t\t\t<OrgId>\n" +
                "\t\t\t\t\t\t\t<BICOrBEI>DEUTDEFF</BICOrBEI>\n" +
                "\t\t\t\t\t\t</OrgId>\n" +
                "\t\t\t\t\t</Id>\n" +
                "\t\t\t\t</Orgtr>\n" +
                "\t\t\t\t<Rsn>\n" +
                "\t\t\t\t\t<Prtry>B00</Prtry>\n" +
                "\t\t\t\t</Rsn>\n" +
                "\t\t\t</StsRsnInf>\n" +
                "\t\t</OrgnlGrpInfAndSts>\n" +
                "\t</FIToFIPmtStsRptS2>\n" +
                "</Document>");
        if (t.hasValidationErrors()) {
            t.printValidationErrors();
        } else {
            System.out.println(t.getMessageDocument().getFIToFIPmtStsRptS2().getGrpHdr().getMsgId());
        }
        t.getMessageDocument().getFIToFIPmtStsRptS2().getGrpHdr().setMsgId("TESTMSGIDPACS002");
        System.out.println("-------------------------------------------");
        System.out.println(t.toString());
        System.out.println("-------------------------------------------");
        Assert.assertTrue(t.toString(), !t.hasValidationErrors());
    }
}
```

Summary

Initialize SEPACTprocessor message.	Name_Of_SEPA_msg message = new Name_Of_SEPA_msg();
Add new data to a SEPACTprocessor message.	message.setElement("some_path", anObject); message.getRootMessage().set<Object>(new object)
Get data from a SEPACTprocessor message.	Object o = message.getElement("path"); message.getRootMessage().get<Object>()

Installation Instructions

The component is packaged as jar file and can be added to any Java IDE project as jar reference. This implementation was currently deployed and tested in Eclipse and other IDE's. It requires at least JDK 1.6.x.

SEPACTprocessor includes:

- *sepa-core-x.x.x-demo.jar* which is the SEPA runtime library

Contact and Support

The support team is available to answer questions on using the component, code assistance, error determination, ideas, examples etc. Please forward your questions to: EMAIL: info@paymentcomponents.com Also visit www.paymentcomponents.com for other components and services by **PaymentComponents**.

Appendix A

SEPACTprocessor supports: The following messages per implementation:

EBA - SEPA CREDIT TRANSFERS

Message Name	Msg ID	Schema ID
FIToFICustomerCreditTransfer	FIToFICstmrCdtTrf	pacs.008.001.02
PaymentStatusReport	FIToFIPmtStsRpt	pacs.002.001.03S2
PaymentReturn	PmtRtr	pacs.004.001.02
FIToFIPaymentCancellationRequest	FIToFIPmtCxlReq	camt.056.001.01
ResolutionOfInvestigation	RsltnOfInvstgtn	camt.029.001.03
CustomerCreditTransferInitiation	CstmrCdtTrfInitn	pain.001.001.03
CustomerPaymentStatusReport	CstmrPmtStsRpt	pain.002.001.03
ClaimNonReceipt	ClmNonRct	camt.027.001.06
RequestToMorifyPayment	ReqToModfyPmt	camt.087.001.05
ResolutionOfInvestigation08	RsltnOfInvstgtn	camt.029.001.08
FIToFIPaymentInstantStatusInquiryForInvestigation	FIToFIPmtStsReq	pacs.028.001.01
SCTCcfBulkCreditTransfer	SCTCcfBlkCredTrf	SCTCcfBlkCredTrf
SCTCvfBulkCreditTransfer	SCTCvfBlkCredTrf	SCTCvfBlkCredTrf
SCTIcfBulkCreditTransfer	SCTIcfBlkCredTrf	SCTIcfBlkCredTrf
SCTScfBulkCreditTransfer	SCTScfBlkCredTrf	SCTScfBlkCredTrf
SCTIqfBulkCreditTransfer	SCTIqfBlkCredTrf	SCTIqfBlkCredTrf
SCTOqfBulkCreditTransfer	SCTOqfBlkCredTrf	SCTOqfBlkCredTrf
SCTQvfBulkCreditTransfer	`SCTQvfBlkCredTrf`	SCTQvfBlkCredTrf
SCTPcfBulkCreditTransfer	SCTPcfBlkCredTrf	SCTPcfBlkCredTrf

EPC - SEPA CREDIT TRANSFERS

Message Name	Msg ID	Schema ID
FIToFICustomerCreditTransfer	FIToFICstmrCdtTrf	pacs.008.001.02
PaymentStatusReport	FIToFIPmtStsRpt	pacs.002.001.03

PaymentReturn	PmtRtr	pacs.004.001.02
FIToFIPaymentCancellationRequest	FIToFIPmtCxlReq	camt.056.001.01
ResolutionOfInvestigation	RsltnOfInvstgtn	camt.029.001.03
ClaimNonReceipt	ClmNonRct	camt.027.001.06
RequestToMorifyPayment	ReqToModfyPmt	camt.087.001.05
ResolutionOfInvestigation08	RsltnOfInvstgtn	camt.029.001.08
FIToFIPaymentInstantStatusInquiryForInvestigation	FIToFIPmtStsReq	pacs.028.001.01

DIAS - SEPA CREDIT TRANSFERS

Message Name	Msg ID	Schema ID
FIToFICustomerCreditTransfer	FIToFICstmrCdtTrf	pacs.008.001.02
PaymentStatusReport	FIToFIPmtStsRpt	pacs.002.001.03
PaymentReturn	PmtRtr	pacs.004.001.02
FIToFIPaymentCancellationRequest	FIToFIPmtCxlReq	camt.056.001.01
ResolutionOfInvestigation	RsltnOfInvstgtn	camt.029.001.03
FIToFIPaymentInstantStatusInquiryForInvestigation	FIToFIPmtStsReq	pacs.028.001.01
DCTBulkCreditTransfer	DIASFileHdr	DIASSCTFH

SIBS - SEPA CREDIT TRANSFERS

Message Name	Msg ID	Schema ID
FIToFICustomerCreditTransfer	FIToFICstmrCdtTrf	pacsx.008.001.02
PaymentStatusReport	FIToFIPmtStsRpt	pacsx.002.001.03
PaymentReturn	PmtRtr	pacsx.004.001.02
FIToFIPaymentCancellationRequest	FIToFIPmtCxlReq	camtx.056.001.01
ResolutionOfInvestigation	RsltnOfInvstgtn	camtx.029.001.03
ClaimNonReceipt	ClmNonRct	camtx.027.001.06
RequestToMorifyPayment	ReqToModfyPmt	camtx.087.001.05
ResolutionOfInvestigation08	RsltnOfInvstgtn	camtx.029.001.08
FIToFIPaymentInstantStatusInquiryForInvestigation	FIToFIPmtStsReq	pacsx.028.001.01
SCTCcxBulkCreditTransfer	SCTCcfBlkCredTrf	SCTCcxBlkCredTrf
SCTCvxBulkCreditTransfer	SCTCvfBlkCredTrf	SCTCvxBlkCredTrf
SCTIcxBulkCreditTransfer	SCTIcfBlkCredTrf	SCTIcxBlkCredTrf
SCTScxBulkCreditTransfer	SCTScfBlkCredTrf	SCTScxBlkCredTrf
SCTIqxBulkCreditTransfer	SCTIqfBlkCredTrf	SCTIqxBlkCredTrf
SCTOqxBulkCreditTransfer	SCTOqfBlkCredTrf	SCTOqxBlkCredTrf
SCTQvxBulkCreditTransfer	SCTQvfBlkCredTrf	SCTQvxBlkCredTrf

EBA - SEPA DIRECT DEBITS

Message Name	Msg ID	Schema ID
FIToFIPaymentCancellationRequest	FIToFIPmtCxlReq	camt.056.001.01

FIToFICustomerDirectDebit	FIToFICstmrDrctDbt	pacs.003.001.02
FIToFIPaymentReversal	FIToFIPmtRvsl	pacs.007.001.02
FIToFIPaymentStatusReport	FIToFIPmtStsRpt	pacs.002.001.03
PaymentReturn	PmtRtr	pacs.004.001.02
MPEDDIdfBlkDirDeb	MPEDDIdfBlkDirDeb	MPEDDIdfBlkDirDeb

EPC - SEPA DIRECT DEBITS

Message Name	Msg ID	Schema ID
FIToFICustomerDirectDebit	FIToFICstmrDrctDbt	pacs.003.001.02
FIToFIPaymentReversal	FIToFIPmtRvsl	pacs.007.001.02
FIToFIPaymentStatusReport	FIToFIPmtStsRpt	pacs.002.001.03
PaymentReturn	PmtRtr	pacs.004.001.02

SIBS - SEPA DIRECT DEBITS

Message Name	Msg ID	Schema ID
FIToFIPaymentCancellationRequest	FIToFIPmtCxlReq	camt.056.001.01
FIToFICustomerDirectDebit	FIToFICstmrDrctDbt	pacs.003.001.02
FIToFIPaymentReversal	FIToFIPmtRvsl	pacs.007.001.02
FIToFIPaymentStatusReport	FIToFIPmtStsRpt	pacs.002.001.03
FIToFIPaymentStatusReportS2	FIToFIPmtStsRpt	pacs.002.001.03S2
PaymentReturn	PmtRtr	pacs.004.001.02
MPEDDCdxBulkDirectDebit	MPEDDCdxBlkDirDeb	MPEDDCdxBlkDirDeb
MPEDDDnxBulkDirectDebit	MPEDDDnxBlkDirDeb	MPEDDDnxBlkDirDeb
MPEDDDrxBulkDirectDebit	MPEDDDrxBlkDirDeb	MPEDDDrxBlkDirDeb
MPEDDDvxBulkDirectDebit	MPEDDDvxBlkDirDeb	MPEDDDvxBlkDirDeb
MPEDDIdxBulkDirectDebit	MPEDDIdxBlkDirDeb	MPEDDIdxBlkDirDeb
MPEDDIrxBulkDirectDebit	MPEDDIrxBlkDirDeb	MPEDDIrxBlkDirDeb

MPEDDRsxBulkDirectDebit	MPEDDRsxBlkDirDeb	MPEDDRsxBlkDirDeb
MPEDDSdxBulkDirectDebit	MPEDDSdxBlkDirDeb	MPEDDSdxBlkDirDeb