4.8.3 SEPA: Process logic

Language: Deutsch · **English**

4.8.3 SEPA: Process logic

> **Adapt the process logic (SEPALogic) for creditor identification, remittance information and the payment references to be considered, and initialize it in the export rules.**

HOW-TO | INTEGRATORS | UPDATED: JUL 2026 | APPLIES TO NUCLOS 4.2026.X

On this page

> **Machine-translated – under review**
>
> This page was machine-translated from German as a first draft and is currently under review. The authoritative source remains the **German original** (see the language switch above).

In the abstract class AbstractSEPALogic, the SEPA Nuclet provides the basic process logic for creating SEPA exports.

However, this process logic must be adapted in an application-specific way to your business practice in three places.

This concerns

- the generation of an application-specific SEPA creditor ID (based on the actual data of your company)
- the design of the remittance information to be transmitted for SEPA direct debits
- the determination of all payment references to be considered in a SEPA export

For this, the classes SEPALogic and SEPALogicFactory must be adjusted when connecting the SEPA Nuclet.

> It is recommended to use the SEPALogic class only as a template for your own implementation. You will find more on this in the next section 4.8.3.1.

In addition, the initialization of the actually used process logic must be inserted into two rules.

Class	Function	Java package	Methods to adjust
SEPALogic	template for application-specific process logic	org.nuclet.sepa.logic	• createCreditorIdentification(SEPAExport) • createDebitorIdentification(SEPAExport) • createRemittanceInformation(AbstractCreditorWrapper, AbstractReferenceWrapper, BigDecimal) • createRemittanceInformation(AbstractDebitorWrapper, AbstractReferenceWrapper, BigDecimal) • fetchCreditTansferReferences(SEPAExport) • fetchDirectDebitReferences(SEPAExport)
ExportSEPAMessage	creates a preview of the SEPA payments to be exported	org.nuclet.sepa.rule	custom(CustomContext)
ExportSEPAMessageAndProcessReferences	creates a SEPA export and marks the exported objects	org.nuclet.sepa.rule	changeState(StateChangeContext)

Table 4.8.3: Overview, adjustments in process logic

4.8.3.1 SEPALogic

Please use the SEPALogic class only as a template for your own implementation.

To do this, please carry out the following steps:

1. Clone the SEPALogic class.
2. When cloning, please specify your own package name in the first line of code (e.g. *com.yourcompany.sepa.logic*) an.
3. Optionally, you can give the cloned class your own class name. Please also take the constructors into account when doing so.

The SEPALogic class serves as a template for supplementing the basic sequences in the process logic.

You must supplement this logic in an application-specific way in three places for the "direct debits" use case and in three places for the "credit transfers" use case:

Direct debits

- The createCreditorIdentification(SEPAExport) method generates the SEPA creditor ID. This ID is used to identify your company in SEPA payment transactions.
- The createRemittanceInformation(AbstractDebitorWrapper,...) method can be used to individually design the remittance information for SEPA direct debits.
- The fetchDirectDebitReferences() method should determine all payment references that are to be considered in a SEPA export.

Credit transfers

- The createDebitorIdentification(SEPAExport) method generates the SEPA debtor ID. This ID is used to identify your company in SEPA payment transactions.
- The createRemittanceInformation(AbstractCreditorWrapper,...) method can be used to individually design the remittance information for SEPA credit transfers.
- The fetchCreditTransferReferences() method should determine all payment references that are to be considered in a SEPA export.

An example is given in each case in a comment block.

org.nuclet.sepa.logic.SEPALogic

```
package org.nuclet.sepa.logic;


import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.xml.bind.JAXBElement;

import org.nuclos.api.businessobject.BusinessObject;
import org.nuclos.api.businessobject.Query;
import org.nuclos.api.context.JobContext;
import org.nuclos.api.context.RuleContext;
import org.nuclos.api.exception.BusinessException;
import org.nuclos.api.provider.BusinessObjectProvider;
import org.nuclos.api.provider.QueryProvider;

import org.nuclet.common.logging.LogLevel;
import org.nuclet.common.logic.AbstractBusinessLogic;

import org.nuclet.sepa.SEPAExport;
import org.nuclet.sepa.SEPAPaymentType;
import org.nuclet.sepa.jaxb.sdd.Document;
import org.nuclet.sepa.logic.*;
import org.nuclet.sepa.logic.PaymentType;
import org.nuclet.sepa.logic.SequenceType;
import org.nuclet.sepa.wrapper.*;

// @replace!
//
// import org.nuclet.businesstemplate.Client;
// import org.nuclet.businesstemplate.ClientBilling;
```

```
// import org.nuclet.businesstemplate.ClientBillingPosition;
// import org.nuclet.businesstemplate.Supplier;
// import org.nuclet.businesstemplate.SupplierBilling;
// import org.nuclet.businesstemplate.SupplierBillingPosition;


/**
 * Concrete class implementing the declarations found in abstract class AbstractSEPALogic
 *
 * @note This class is to be read as a dummy implementation. Its methods should be overwritten
 * and filled/adapted with application specific code.
 *
 * @usage org.nuclet.sepa.job.SEPAExportJob
 *
 * @version 2.0
 * @date 03.02.2015
 * @nuclet org.nuclet.SEPA
 * @nucletversion 2.0.0
 * @sincenucletversion 1.0.0
 * @since 21.03.2014
 *
 * @author frank.lehmann@nuclos.de
 *
 */
public class SEPALogic extends AbstractSEPALogic
{
 private static final LogLevel LOGLEVEL = LogLevel.DEBUG;

 public SEPALogic(final JobContext context)
 {
 super(context, LOGLEVEL);
 }

 public SEPALogic(final JobContext context, final LogLevel logLevel)
 {
 super(context, logLevel);
 }

 public SEPALogic(final RuleContext context, final LogLevel logLevel)
 {
 super(context, logLevel);
 }

 public SEPALogic(final RuleContext context)
 {
 super(context, LOGLEVEL);
 }


 /**
 * Builds a creditor identification, based on a given SEPA export (direct debit initiation)
 *
 * @note To be implemented in a dedicated subclass!
 * @see org.nuclet.sepa.logic.CreditorIdentification
 *
 * @param boSEPAExport A given oject of type <code>SEPAExport</code>
 *
 * @return a creditor identification, based on a given SEPA export
 *
 * @throws BusinessException
 * @throws BusinessException SPA.483.001
 */
 protected CreditorIdentification createCreditorIdentification(final SEPAExport boSEPAExport) throws
BusinessException
 {
 // @replace!
 //
 // Im folgenden Beispiel wird der Kreditorenstammdatensatz in der Tabelle
 // der Kunden ("Client") gespeichert und als "intern" ("is internal") gekennzeichnet.
 // Die Kreditorenkennung wird also aus genau jenem Stammdatensatz erzeugt.
 //
```

```
 // final Query qryCreditor = QueryProvider.create(Client.class)
 // .where(Client.IsInternal.eq(Boolean.TRUE));
 // final List<Client> lstClients = QueryProvider.execute(qryCreditor);
 //
 // logDebug("Creditor list size: " + lstClients.size());
 //
 // if (lstClients != null && lstClients.size() == 1) {
 // final Client boCreditor = lstClients.iterator().next();
 //
 // final CreditorIdentification creditorId = new CreditorIdentification(
 // boCreditor.getName(),
 // boCreditor.getBic(),
 // boCreditor.getIban(),
 // boCreditor.getSEPACreditorId());
 //
 // return creditorId;
 // } else if (lstClients != null && lstClients.size() > 1) {
 // throw new BusinessException("More than one clients marked as \"internal\" exist!");
 // } else {
 // throw new BusinessException("Client marked as \"internal\" is missing!");
 // }

 return null;
 }

 /**
 * Builds a creditor identification, based on a given SEPA export (credit transfer initiation)
 *
 * @note To be implemented in a dedicated subclass!
 * @see org.nuclet.sepa.logic.CreditorIdentification
 *
 * @param boSEPAExport A given oject of type <code>SEPAExport</code>
 *
 * @return a debitor identification, based on a given SEPA export
 *
 * @throws BusinessException
 * @throws BusinessException SPA.483.001
 */
 protected DebitorIdentification createDebitorIdentification(final SEPAExport boSEPAExport) throws
BusinessException
 {
 // @replace!
 //
 // Im folgenden Beispiel wird der Debitorenstammdatensatz in der Tabelle
 // der Kunden ("Client") gespeichert und als "intern" ("is internal") gekennzeichnet.
 // Die Debitorenkennung wird also aus genau jenem Stammdatensatz erzeugt.
 //
 // final Query qryDebitor = QueryProvider.create(Client.class)
 // .where(Client.IsInternal.eq(Boolean.TRUE));
 // final List<Client> lstClients = QueryProvider.execute(qryDebitor);
 //
 // logDebug("Debitor list size: " + lstClients.size());
 //
 // if (lstClients != null && lstClients.size() == 1) {
 // final Client boDebitor = lstClients.iterator().next();
 //
 // final DebitorIdentification debitorId = new DebitorIdentification(
 // boDebitor.getName(),
 // boDebitor.getBic(),
 // boDebitor.getIban(),
 // boDebitor.getSEPACreditorId());
 //
 // return debitorId;
 // } else if (lstClients != null && lstClients.size() > 1) {
 // throw new BusinessException("More than one clients marked as \"internal\" exist!");
 // } else {
 // throw new BusinessException("Client marked as \"internal\" is missing!");
 // }

 return null;
 }
```

```
/**
* Builds the remittance information, related to the given creditor and reference
*
* @param creditor An object of type <code>AbstractCreditorWrapper</code>, representing the debitor
* @param reference An object of type <code>AbstractCreditorReferenceWrapper</code>, representing the reference
* @param bgdAmount The payment amount, related to the given debitor and reference
*
* @return the remittance information, related to the given debitor and reference
*
* @throws BusinessException
*/
public String createRemittanceInformation(final AbstractCreditorWrapper debitor,
final AbstractCreditorReferenceWrapper reference,
final BigDecimal bgdAmount) throws BusinessException
{
// @replace!
//
// final SupplierBilling boSupplierBilling = (SupplierBilling)reference.getBusinessObject();
// final StringBuffer sbRemittanceInfo = new StringBuffer();
//
// sbRemittanceInfo.append("ACCT. ");
// sbRemittanceInfo.append(creditor.getSEPAMandateIdentification());
// sbRemittanceInfo.append(" DATE ");
// sbRemittanceInfo.append(reference.getReferenceDate());
// sbRemittanceInfo.append(" AMNT. ");
// sbRemittanceInfo.append(bgdAmount);
// sbRemittanceInfo.append(" BGNR ");
// sbRemittanceInfo.append(reference.getCreditTransferReference());
//
// return sbRemittanceInfo.toString();

return null;
}

/**
* Builds the remittance information, related to the given debitor and reference
*
* @param debitor An object of type <code>AbstractDebitorWrapper</code>, representing the debitor
* @param reference An object of type <code>AbstractReferenceWrapper</code>, representing the reference
* @param bgdAmount The payment amount, related to the given debitor and reference
*
* @return the remittance information, related to the given debitor and reference
*
* @throws BusinessException
*/
public String createRemittanceInformation(final AbstractDebitorWrapper debitor,
final AbstractDebitorReferenceWrapper reference,
final BigDecimal bgdAmount) throws BusinessException
{
// @replace!
//
// final ClientBilling boClientBilling = (ClientBilling)reference.getBusinessObject();
// final StringBuffer sbRemittanceInfo = new StringBuffer();
//
// sbRemittanceInfo.append("ACCT. ");
// sbRemittanceInfo.append(debitor.getSEPAMandateIdentification());
// sbRemittanceInfo.append(" DATE ");
// sbRemittanceInfo.append(reference.getReferenceDate());
// sbRemittanceInfo.append(" AMNT. ");
// sbRemittanceInfo.append(bgdAmount);
// sbRemittanceInfo.append(" BGNR ");
// sbRemittanceInfo.append(reference.getDirectDebitReference());
//
// return sbRemittanceInfo.toString();

return null;
}

/**
* Fetches all credit transfer references that are to be included in a given SEPA export (credit transfer
```

```
initiation)
 *
 * @param boSEPAExport A given SEPA export
 *
 * @throws BusinessException
 */
 protected List<AbstractCreditTransferReferenceWrapper> fetchCreditTransferReferences(final SEPAExport
boSEPAExport)
 throws
 BusinessException
 {
 // @replace!
 //
 // final List<AbstractCreditTransferReferenceWrapper> lstCreditTransferReferences = new
ArrayList<AbstractCreditTransferReferenceWrapper>();
 //
 // final Query qrySupplierBillingPositions = QueryProvider.create(SupplierBillingPosition.class)
 // .where(SupplierBillingPosition.DueDate.Gte(boSEPAExport.getDateFrom()))
 // .and(SupplierBillingPosition.Faelligkeitsdatum.Lte(boSEPAExport.getDateUntil()))
 // .and(SupplierBillingPosition.SEPAExportDate.isNull())
 // .and(SupplierBillingPosition.IsOpen.eq(Boolean.TRUE))
 // .orderBy(SupplierBillingPosition.SupplierBilling, Boolean.TRUE)
 // .orderBy(SupplierBillingPosition.DueDate, Boolean.TRUE);
 // final List<SupplierBillingPosition> lstSupplierBillingPositions = QueryProvider.execute
(qrySupplierBillingPositions);
 //
 // for (final SupplierBillingPosition boSupplierBillingPosition : lstSupplierBillingPositions) {
 // lstCreditTransferReferences.add(creditTransferReferenceFacade.getWrapper(boSupplierBillingPosition));
 // }
 //
 // return lstCreditTransferReferences;

 return null;
 }

 /**
 * Fetches all direct debit references that are to be included in a given SEPA export (direct debit initiation)
 *
 * @param boSEPAExport A given SEPA export
 *
 * @throws BusinessException
 */
 protected List<AbstractDirectDebitReferenceWrapper> fetchDirectDebitReferences(final SEPAExport boSEPAExport)
 throws
 BusinessException
 {
 // @replace!
 //
 // final List<AbstractDirectDebitReferenceWrapper> lstDirectDebitReferences = new
ArrayList<AbstractDirectDebitReferenceWrapper>();
 //
 // final Query qryClientBillingPositions = QueryProvider.create(ClientBillingPosition.class)
 // .where(ClientBillingPosition.DueDate.Gte(boSEPAExport.getDateFrom()))
 // .and(ClientBillingPosition.Faelligkeitsdatum.Lte(boSEPAExport.getDateUntil()))
 // .and(ClientBillingPosition.SEPAExportDate.isNull())
 // .and(ClientBillingPosition.IsOpen.eq(Boolean.TRUE))
 // .orderBy(ClientBillingPosition.ClientBilling, Boolean.TRUE)
 // .orderBy(ClientBillingPosition.DueDate, Boolean.TRUE);
 // final List<ClientBillingPosition> lstClientBillingPositions = QueryProvider.execute
(qryClientBillingPositions);
 //
 // for (final ClientBillingPosition boClientBillingPosition : lstClientBillingPositions) {
 // lstDirectDebitReferences.add(directDebitReferenceFacade.getWrapper(boClientBillingPosition));
 // }
 //
 // return lstDirectDebitReferences;

 return null;
 }
```

```
}
```

4.8.3.2 ExportSEPAMessage

In the ExportSEPAMessage class, the SEPALogic class must be replaced by the process logic you created in section 4.8.3.1.

org.nuclet.sepa.rule.ExportSEPAMessage

```
package org.nuclet.sepa.rule;

import org.nuclos.api.annotation.Rule;
import org.nuclos.api.context.CustomContext;
import org.nuclos.api.exception.BusinessException;
import org.nuclos.api.rule.CustomRule;

import org.nuclet.sepa.SEPAExport;
import org.nuclet.sepa.logic.SEPALogic;


/**
 * @name ExportSEPAMesssage
 * @description Export SEPA message without processing the references
 * @usage
 * @change
 *
 * @version 2.0
 * @date 30.01.2015
 * @nuclet org.nuclet.SEPA
 * @nucletversion 2.0.0
 * @sincenucletversion 1.0.0
 * @since 14.03.2013
 *
 * @author frank.lehmann@nuclos.de
 */
@Rule(name="ExportSEPAMesssage", description="Export SEPA Messsage")
public class ExportSEPAMesssage implements CustomRule
{

 public void custom(CustomContext context) throws BusinessException
 {
 final SEPAExport boSEPAExport = context.getBusinessObject(SEPAExport.class);

 SEPALogic logic = new SEPALogic(context);
 logic.initialize(context, boSEPAExport);
 logic.export();
 }
}
```

4.8.3.3 ExportSEPAMessageAndProcessReferences

In the ExportSEPAMessageAndProcessReferences class, the SEPALogic class must be replaced by the process logic you created in section 4.8.3.1.

org.nuclet.sepa.rule.ExportSEPAMessageAndProcessReferences

```
package org.nuclet.sepa.rule;

import org.nuclos.api.annotation.Rule;
import org.nuclos.api.context.StateChangeContext;
import org.nuclos.api.exception.BusinessException;
import org.nuclos.api.rule.StateChangeRule;

import org.nuclet.sepa.SEPAExport;
import org.nuclet.sepa.logic.SEPALogic;


/**
 * @name ExportSEPAMesssageAndProcessReferences
 * @description Export SEPA messsage and process references
 * @usage
 * @change
 *
 * @version 2.0
 * @date 30.01.2015
 * @nuclet org.nuclet.SEPA
 * @nucletversion 2.0.0
 * @sincenucletversion 1.0.0
 * @since 21.03.2014
 *
 * @author frank.lehmann@nuclos.de
 */
@Rule(name="ExportSEPAMesssageAndProcessReferences", description="Export SEPA messsage and process references")
public class ExportSEPAMesssageAndProcessReferences implements StateChangeRule
{

 public void changeState(StateChangeContext context) throws BusinessException
 {
 final SEPAExport boSEPAExport = context.getBusinessObject(SEPAExport.class);

 SEPALogic logic = new SEPALogic(context);
 logic.initialize(context, boSEPAExport);
 logic.export();
 logic.processReferences(boSEPAExport);
 }
}
```

Related pages

📖 Nuclet: SEPA

SEPA Nuclet data sheet

Open

📖 Interfaces

All integrations

Open