integrators
part of Allstar Group

ESB JPA (Enterprise Service Bus - Java Process Automat)

Author: Michal Toman

Date: 22.07.2012

Version: 1.0

Content

1. BUSINESS INTRODUCTION

ESB JPA (Enterprise Service Bus - Java Process Automat) is a software product that serves to implement an enterprise service bus in the company environment.

The core of the enterprise service bus is formed by a set of processes within one instance. Each process defines different business logic.

Business logic is determined with modules linked in a tree structure. Modules are of defined types. These types are derived from basic integration operations with data. Specific implementations of modules and their configurations determine the final logic of data-processing.

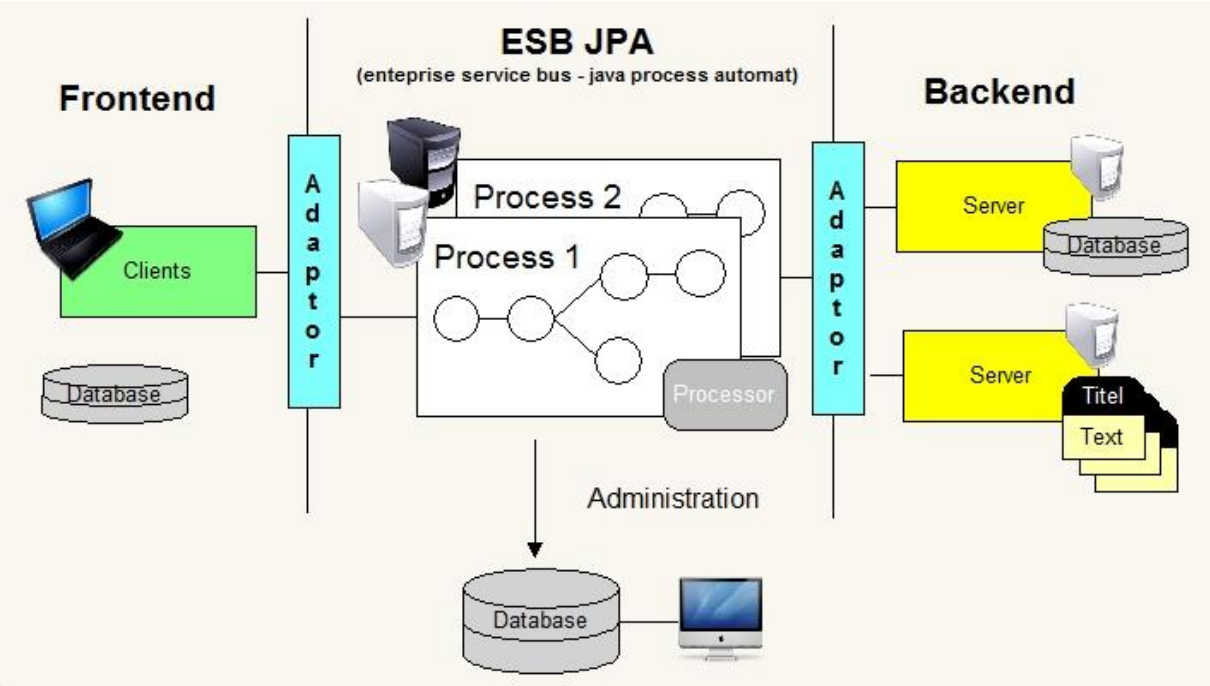

Figure 1 Basic Scheme of the ESB JPA

Clients connect to the enterprise service bus using the interfaces of thin adapters. Thin adapters are technological interfaces for specific communication protocols or for specific data storages.

The main functionalities of the ESB JPA are:

- receiving data from inputs
- transforming data
- sending data to outputs
- handling error states
- generating statistical data or reports

In addition, there is support for transaction processing and parallel data processing.

The ESB JPA contains as well the functionalities for administration as follows:

- local or remote administration of the application (and of single processes or tasks)
- runtime monitoring of the application (and of single processes or tasks)
- monitoring of data traffic (including the results of data-processing)

A useful feature that you can configure the enterprise service bus to run scheduled tasks at defined times.

The application is written in the standard of JSE (Java Standalone Application). There is also a plug-in that plugs the functionality of ESB JPA into JBoss ESB.

2. CORE ARCHITECTURE

The main control module of an instance is a manager. The manager manages processes and scheduler.

Processes define and implement data-processing, which is controlled by the arrival of input data. Scheduler runs scheduled tasks that implement data-processing planned in time.

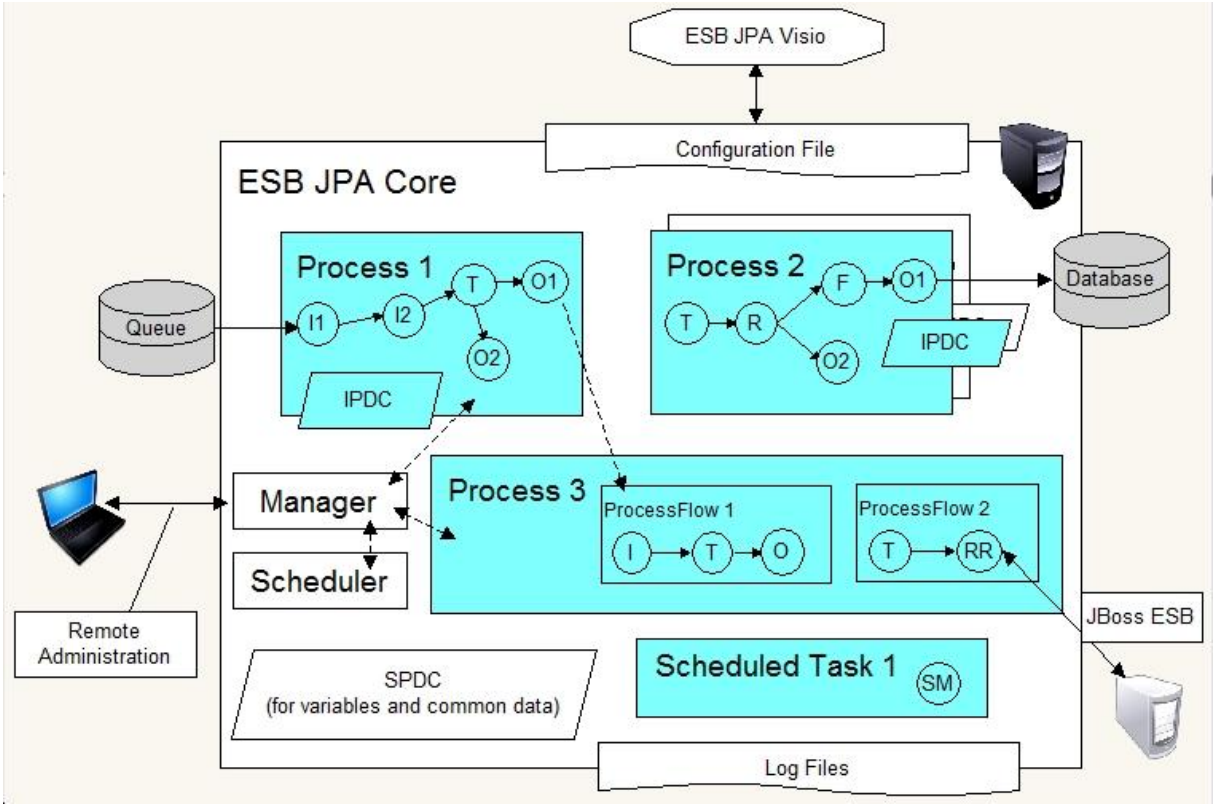

Figure 2 Core Architecture of the ESB JPA

A process is defined by the sequence of ProcessFlows which carry out the serial data-processing. Specific logic of data-processing within one ProcessFlow is defined by the

tree structure of configured modules. The tree structure of modules determines in what order which module is called. Modules can be of more types. There are predefined methods for the modules. The methods are called from the core of the ESB JPA. The specific implementations of the modules, called plugins, define concrete logic of data-processing in the module.

ProcessFlow defines a transaction unit of work. ProcessFlow is being performed serially. Parallelization can be achieved by running multiple threads in the process.

I/O (Input/Output) thin adapters are implemented also in the modules. These modules are designed specially to read data from inputs and write data to outputs, e.g. database or file system. The same applies for communication protocols (e.g. TCP/IP or SMTP).

2.1. Modules and Methods

The idea of modules is based on the transformation of pulses in data networks. The input signal is transformed to an output signal as the network is now connected.

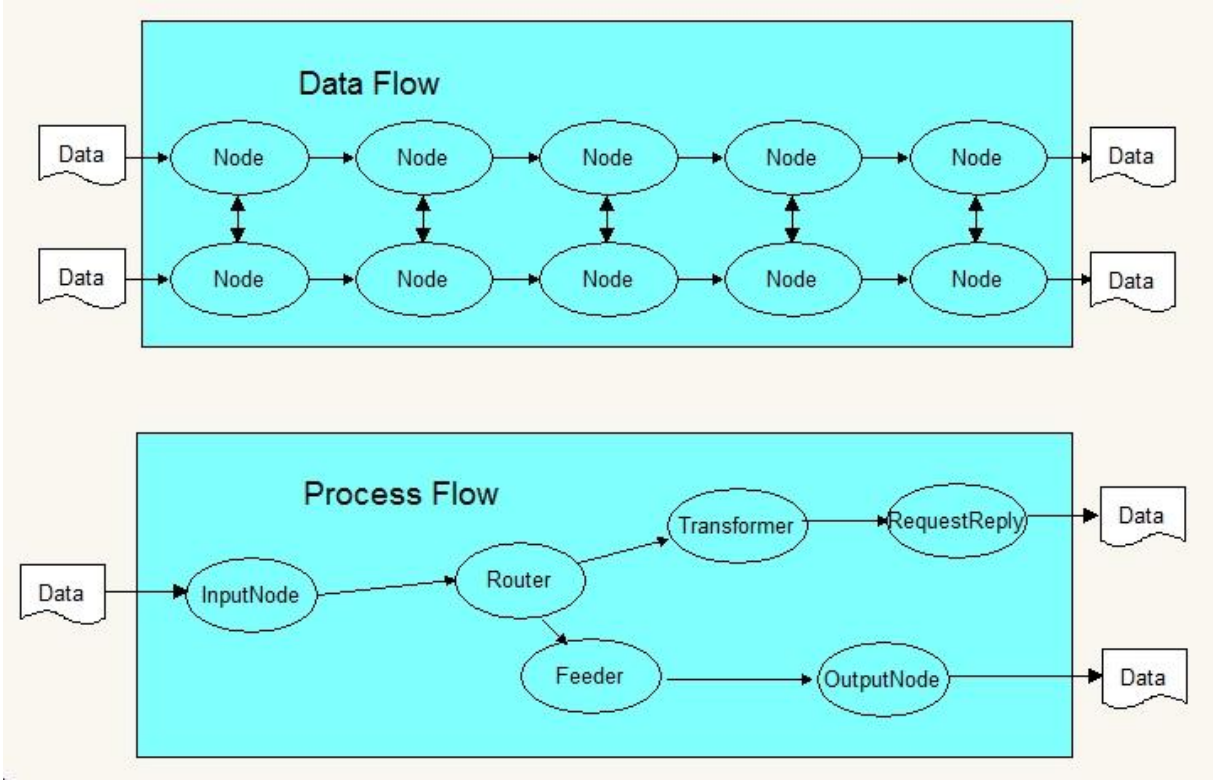

Figure 3 Modules in the ESB JPA as the simple abstraction for data network

Modules in the ESB JPA are very simple abstraction for such data network.

Using the configuration tree structure of modules, it is prescribed for an instance of the ESB JPA as follows:

- which and what inputs the data should be read from
- which and what transformations should be used to transform the input data to the output data
- which and what outputs the data should be written to

- how errors states should be handled

Modules in the ESB JPA (in one ProcessFlow) can be of defined type, as it is described in the following table.

Type of Module	Description
IONode	Module to manage the input and the output (e.g. reading data from files or writing data to files)
Transformer	Module to transform data (e.g. conversion of data formats)
Router	Module to route the processing of data to more branches on the basis of data content (e.g. using the content of one data field with the given name)
RequestReply	Module to send a request and receive a reply (e.g. for HTTP protocol or Web Services)
Aggregator	Module to aggregate data from more inputs (e.g. aggregation of text lines from more files to one file)
SubFlow	Module to define a procedural data-processing that can be called from more places in the Process (e.g. repeated action to send a report)
DataFeeder	Module to split one large data record (e.g. content of one file) to multiple smaller partial data records (e.g. single text lines)
JoinedAction	Module to perform a set of actions (e.g. call a set of actions in Groovy environment)
Workflow	Module to run a thread, in which the data are being processed in a parallel way. The data is being processed as the body of the workflow is defined

The following table shows the overview of basic methods for the modules.

Method	Module	Description
readData	IONode	Reads data from the inputs
writeData	IONode	Writes data to the outputs
transform	Transformer	Transforms input data to output data
route	Router	Routes data to output destinations
sendRequest	RequestReply	Sends a regest to the target server
receiveReply	RequestReply	Receives reply from the target server
collectData	Aggregator	Collects data from more inputs until it is complete
feedData	DataFeeder	Parses input data and feeds data parts to the ProcessFlow
perform	JoinedAction	Performs a set of actions (Note: action in the set is a specific class that receives input data as the argument and returns transformed data)
execute	Workflow	Executes the body of workflow
begin	IONode, RequestReply,	Begins the transaction for the node in the ProcessFlow

	Aggregator	
commit	IONode, RequestReply, Aggregator	Commits the transaction for the node in the ProcessFlow
rollback	IONode, RequestReply, Aggregator	Rolls the transaction back for the node in the ProcessFlow

2.2. Data Memory

Data is kept in the memory of the ESB JPA. Data can be passed among processes (or tasks) using the SPDC and inside the process using the IPDC.

2.2.1. IPDC and Data Packages or Variables

IPDC (Internal Process Data Container) serves to pass data inside the process. IPDC is managed by a thread of the process.

Data is a part of data packages. The data packages are identified by the keys. There can be packages in the IPDC as many as the memory is available.

Data can be stored in variables as well. The variables are identified by the keys. There can be variables in the IPDC as many as the memory is available.

2.2.2. SPDC and Variables

SPDC (Shared Process Data Container) serves to pass data among processes or tasks under one manager. SPDC is managed by the manager.

Data is stored in variables. The variables are identified by the keys. There can be variables in the SPDC as many as the memory is available.

2.3. Data Formats

There are several data formats that can be used to pass data within the process of the ESB JPA. The formats are described in the following table.

Data Format	Description
JPAData	format derived from the map where each data field is identified by a key and contains a value
ASCII text	plain text that is passed as the String
XML	plain text in the XML syntax that is passed as the String
binary data	binary data that is passed as the array of bytes

2.4. Data-Processing

Modules process data in the memory. Modules work with data in the related packages. The packages are identified using the keys.

Modules are a part of a process flows. Modules are configured through nodes in the configuration of process flow. Related keys of packages are configured in the configuration of node.

Each type of module works with a set of keys as written in the following table.

Type of Module	Related keys of packages
IONode	Related package with/for data is the first configured packageKey, or dataToProcessPackageKey. Only in case of the input node, it applies further: Package with data to the DLD-destination can be configured as the second one, or using dataToDLDPackageKey Package with data for the report can be configured as the third one, or using reportDataPackageKey
Transformer	Related package with input data is the first configured packageKey, or dataToProcessPackageKey. Related package for transformed data is the second configured packageKey, or transformedDataPackageKey. Package with data for the report can be configured as the third one, or using reportDataPackageKey
Router	Related package with data is the first configured packageKey, or dataToProcessPackageKey.
RequestReply	Related package with request data is the first configured packageKey, or dataToProcessPackageKey. Related package for reply data is the second configured packageKey, or replyDataPackageKey.
Aggregator	Related package for data is the first configured packageKey, or aggregatedDataPackageKey.
DataFeeder	Related package with data is the first configured packageKey, or dataToFeedPackageKey.
JoinedAction	Related package with data is the first configured packageKey, or dataToProcessPackageKey.
Workflow	Related package with data is the first configured packageKey, or dataToProcessPackageKey.

2.4.1. Data-Processing in Child Nodes

Using the linkage-tag in the configuration of node, you can build a tree structure of nodes (resp. modules). First data is processed in the parent node and then data is processed in its child nodes from left to right.

It is useful mainly in connection with the Router-module because in the Router-module you can route the processing of data only to some of defined child nodes, e.g. based on the content of data.

2.5. Connection Pools

Connections can be managed (opened and closed) directly in the modules, or these connections can be managed by the pools. Using the pools, the access to the data storage can be more flexible. Connecting to a database (or other data storage) typically consists of several time-consuming steps.

2.6. Initialization and Finalization

Each plug-in must implement the method initialize() and the method finalize(). If there is an error during the initialization, the plug-in can throw an exception to stop starting its related process.

2.7. Error-Handling

An error state can occur in 2 principal cases. These cases are:

- an error state when initializing an application
- an error state when processing data

2.7.1. Initialization

If there is an error during the initialization phase and if an exception is thrown, the related process is not started. The exception can be thrown from the core of the ESB JPA, or from single plug-ins.

There are 2 types of exceptions that can be thrown:

- JPAException
- JPADSCPoolException

Details are described in the following table.

Exception	Description
JPAException	There is an error while processing data, but the data is OK
JPADSCPoolException	There is an error while connecting to a data source

2.7.2. Data-processing

The processing of data can end in the plug-in OK, or with errors. If there is an error, an exception is thrown from the plug-in (or from the core of ESB JPA).

There are 3 types of exceptions that can be thrown:

- JPAException
- JPADataException
- JPADSCPoolException

Details are described in the following table.

Exception	Description
JPAException	There is an error while processing data, but the data is OK
JPADataException	There is an error in the data (data is invalid, or content of data is not correct)
JPADSCPoolException	There is an error while connecting to a data source

2.7.3. Handling an error state

An error state can be handled by:

- rolling data back to the input(s)
- sending data to the DLD-destination (dead-letter destination)
- commiting the transaction, even with errors

The default behaviour is that an error state is handled by rolling data back to the inputs.

2.7.4. Error Handlers

Error handler is a special type of ProcessFlow with own tree structure of modules, and more with predefined methods that are called.

If there is an error state, the methods and the body of error handler are called to handle the error state. The apprropriate error handler can return to the ProcessFlow how the error state should be handled.

The methods of error handlers are described in the following table.

Method	Description
preProcessError	This method is called before invoking the body of ErrorHandler
postProcessError	This method is called after invoking the body of ErrorHandler
getTypeOfTransactionHandling	This method is called after the method postProcessError and returns type of transaction handling – see the following table
handleTransactionInSpecific	This method is called in case of transaction handling equal to _TH_NO_HANDLING

The types of transaction handling are described in the following table.

Type	Description
_TH_COMMIT_ALL	Calls the method commit for all nodes in the ProcessFlow.
_TH_ROLLBACK_ALL	Calls the method rollback for all nodes in the ProcessFlow.
_TH_PASS_TO_DLD	Passes data to DLD by invoking the DLD handler and calls the method commit for all input nodes in the ProcessFlow and the method rollback for all output nodes in the ProcessFlow.
_TH_NO_HANDLING	Calls the method handleTransactionInSpecific of the parent ErrorHandler

2.7.5. DLDHandlers

DLDHandlers are modules to write data to DLD-destinations. Plug-ins must implement the method writeDataToDLD as it is described in the following table.

Method	Description
writeDataToDLD	writes data to the DLD-destination associated with the DLD handler

3. SECURITY

There are 3 options how to provide some security using the ESB JPA:

- data encryption
- setting profiles when processing administration commands
- security provided by underlying data storages

3.1. Data Encryption

TJEncrypt-transformer and TJDecrypt-transformer can be used to encrypt and decrypt data. Standards to encrypt data are provided by the underlying JSE.

3.2. Setting Profiles When Processing Administration Commands

An instance of the ESB JPA can be administered using the Administration Manager Interface. The ESB JPA can be administered locally, or remotely. The ESB JPA provides mechanism how to set profiles to secure processing of the commands. See the section Configuration of Security.

3.3. Security Provided by Underlying Data Storages

Most of security is based on the security provided by underlying data storages. You can set privileges to access files in the file system, or you can set privileges to access tables in the database. The same is valid for sending emails and so on.

4. CONFIGURATION

An instance of the ESB JPA is configured with the local files. The configuration is in the XML syntax. In the following table (and sections) configuration parameters are described.

Configuration file	Parameters
Configuration of Manager	• Configuration of manager's attributes • Links to configuration files of managed processes • Links to configuration files of managed scheduled tasks • Configurations of used connection pools related to the manager
Configuration of Process	• Configuration of process' attributes • Configurations of included process flows • Configurations of modules in process flows • Configurations of used connection pools related to the process
Configuration of Scheduled Task	• Configuration of scheduled task's attributes
Configuration of Log4J	• Configuration of attributes for runtime logging, logging of statistical data or data auditing

Besides the runtime environment, there is ESB JPA Visio that allows to configure basic processes and tasks using the GUI.

4.1. Attributes vs. Arguments

Attributes are basic configuration tokens that are implicit in the configuration of the ESB JPA.

Explicit configuration parameters of modules are called arguments. Arguments are defined by the modules. The same applies to error handlers, tasks or connection pools.

Each attribute or argument is a pair that is identified by a key and contains a value.

4.2. Configuration of Manager

Configuration of manager is read from a local file, e.g. manager.xml. Configuration is in the XML syntax.

The root tag of the configuration is manager.

The following table shows a list of attributes allowed in the configuration of manager.

Attribute Tag	Description
name	name of the manager
workProcess	link to a local file with configuration of a working process
adminProcess	link to a local file with configuration of an administration

	process
task	link to a local file with configuration of a scheduled task
scheduler	tag to configure scheduler under the manager. See the following table
pool	tag to configure a pool. More pools can be configured. See the section of Configuration of Connection Pool
roundtripSleepTimeout	timeout [in milliseconds] to sleep in manager's method run() between round-trips
defaultDataCharset	default characters set
Simplesecurityprofile	link to a local file to configure the security when processing administration commands – see the section Configuration of Security

Configuration of scheduler

Attribute Tag	Description
name	name of scheduler
roundtripSleepTimeout	timeout [in milliseconds] to sleep in scheduler's method run() between round-trips

4.3. Configuration of Process

Configuration of process is read from a local file, e.g. process_work_1.xml. Configuration is in the XML syntax.

The root tag of the configuration is process.

The following table shows a list of attributes allowed in the configuration of process.

Attribute Tag	Description
name	name of the process
countOfThreads	count of thread instances of the process
errorHandler	tag to configure an error handler. More error handlers can be configured. See the configuration of error handler as follows
subFlow	tag to configure a subflow. More subflows can be configured. See the configuration of subflow as follows
pool	tag to configure a pool. More pools can be configured. See the section of Configuration of Connection Pool
processFlow	tag to configure a process flow. More process flows can be configured. See the configuration of process flow as follows
roundtripSleepTimeout	timeout [in milliseconds] to sleep in thread's method run() between round-trips

sleepTimeoutWhenNoData	timeout [in milliseconds] to sleep in a process flow when no data has been read or processed
sleepTimeoutWhenError	timeout [in milliseconds] to sleep in a process flow when there is an error state (e.g. an application cannot connect to a database and the transaction must be rolled back)
intervalToCollectStatistics	interval [in milliseconds] to collect statistical data and write this data to a log file using the log4J
interruptWhenNoData	true/false> if there are more process flows configured and if there is no data in the actual process flow, true means not continue to next process flows in the row
auditData	true/false> true sets the option of full data auditing, false the option of minimal data auditing
defaultDataCharset	default characters set

Configuration of process flow

Attribute Tag	Description
name	name of the process flow
node	tag to configure a node. More nodes can be configured. See the configuration of node as follows
inputNode	tag to configure an input node. More input nodes can be configured. See the configuration of node as follows
outputNode	tag to configure an output node. More output nodes can be configured. See the configuration of node as follows
errorHandler	link to the name of a configured error handler. One error handler can be linked with one process flow
checkIfAnyDataUpToNode	name of a node to check if any data has been read. If there is no data, go to sleep for a configured timeout and begin a new round-trip
isTHandleDrivenByAppl	true/false> true sets the option of committing the transaction in the modules. This applies when more data records are read from the input in one batch. In case of false, the transaction is committed by the core of the ESB JPA after one round-trip had been finished
hCommitOKSubflow	link to the name of a configured subflow that is called when the transaction is committed properly. One this subflow can be link with one process flow
hCommitErrorSubflow	link to the name of a configured subflow that is called when the transaction is NOT committed properly. One this subflow can be link with one process flow

Configuration of node (resp. inputNode or outputNode)

Attribute Tag	Description
name	name of the node

class	reference to a class implementing the node
packageKey	tag to configure key of related package with/for data. A node works with these related packages. More keys can be configured. Instead of packageKey and its index in the configuration, specific tags can be used as follows: • dataToProcessPackageKey • transformedDataPackageKey • reportDataPackageKey • dataToDLDPackageKey • replyDataPackageKey • dataToFeedPackageKey • aggregatedDataPackageKey See the section Data-Processing
arg(ument) or property	tag to configure argument (or property) of the node. Specific arguments are described for the specific plug-ins
linkage	tag to link a child node to the node. See the section Data-Processing in Child Nodes
dynamicRouting	true/false> true means that thread variables in the IPDC can be used, e.g. to build names of output files. It is described for the specific plug-ins

Configuration of linkage

Attribute Tag	Description
name	name of the linkage to link a child node to its parent node
node	tag to configure a related node. See the configuration of node in the above text
inputNode	tag to configure a related input node. See the configuration of node in the above text
outputNode	tag to configure a related output node. See the configuration of node in the above text

Configuration of subflow

Attribute Tag	Description
name	name of the subflow
node	tag to configure a node in the body of subflow. See the configuration of node in the above text
inputNode	tag to configure an input node in the body of subflow. See the configuration of node in the above text
outputNode	tag to configure an output node in the body of subflow. See

	the configuration of node in the above text

Configuration of error handler

Attribute Tag	Description
name	name of the error handler
class	reference to a class implementing the error handler
DLDHandler	tag to configure DLD handler related to the error handler. See the following table
arg(ument) or property	tag to configure argument (or property) of the error handler. Specific arguments are described for the specific plug-ins
node	tag to configure a node in the body of error handler. See the configuration of node in the above text
inputNode	tag to configure an input node in the body of error handler. See the configuration of node in the above text
outputNode	tag to configure an output node in the body of error handler. See the configuration of node in the above text

Configuration of DLD handler

Attribute Tag	Description
name	name of the DLD handler
class	reference to a class implementing the DLD handler
arg(ument) or property	tag to configure argument (or property) of the DLD handler. Specific arguments are described for the specific plug-ins

4.4. Configuration of Scheduled Task

Configuration of task is read from a local file, e.g.task_1.xml. Configuration is in the XML syntax.

The root tag of the configuration is task.

The following table shows a list of attributes allowed in the configuration of task.

Attribute Tag	Description
name	name of the scheduled task
class	reference to a class implementing the scheduled task
daemon	true/false> true means that the scheduled task is a daemon thread and it is not dependent on the planning in scheduler. False means that the scheduled task is planned and started by scheduler where one start of the task is equal to its one round-trip

roundtripSleepTimeout	timeout [in milliseconds] to sleep in task's method run() between round-trips when it is running as a daemon thread
timeToStartSinceDayBegin	timeout [in milliseconds] since the beginning of the day to start the task. Note: this can be configured using the HH:MM (e.g. 17:58) syntax as well
delayToRepeatAfterStart	delay [in milliseconds] to repeat starting the task after the first start in the day
retryCount	maximum count [integer > 0] to repeat starting the task after the first start in the day
arg(ument) or property	tag to configure argument (or property) of the scheduled task. Specific arguments are described for the specific plug-in-tasks

4.5. Configuration of Connection Pool

Configuration of connection pool is a part of the configuration of manager, or a part of the configuration of process. In the following table, there is an overview of attributes that can configure an instance of connection pool.

Attribute	Description
class	reference to a class implementing the connection pool
name	name (identifier) of the connection pool
countOfConnections	count of connections to manage in the pool
arg(ument) or property	tag to configure argument (or property) of the connection pool. Specific arguments are described for the specific plug-ins

4.6. Configuration of Log4J

Configuration of runtime logging, logging of statistical data or data auditing is based on the package log4J. Log4J is used to provide these types of logging. Log4J is a well-known logging package of the Apache Software Foundation.

In the configuration of log4J, more loggers can be configured. The name of logger to use in the module of ESB JPA is taken as:

- logger_prefix.process_name.thread_name (e.g. LOG.PROCESS1.JPAThread0)
- or logger_prefix.process_name (e.g. LOG.PROCESS1)
- or logger_prefix (e.g. LOG)

The above mentioned loggers must be configured in the log4j.properties file.

Logger prefixes are described in the following tables.

Logger Prefix	Description
LOG	runtime logging

ADMINLOG	logging for administration
STATS	logging of statistical data
AUDIT	data auditing

4.7. Configuration of Security

Configuration of security profiles is read from a local file, e.g. simplesecuritycfg.txt. Configuration is in the plain text.

One line configures one security profile. The syntax of the line is as follows:

```
user:password:role:access_profile
```

Security roles are described in the following table.

Role	Description
Administrator	This role is an administrator. Administrator has all privileges
User	This role is a normal user. Normal user must verify privileges to access objects

An access profile is a profile to verify privileges to access objects with given names. Regular expressions are used.

If the security is configured and the user in an administration messages in not configured in the security, this user has no privileges to submit administration commands.

5. ADMINISTRATION

The application (an instance of the ESB JPA) can be administered using signals or the Administration Manager Interface.

5.1.1. Signals

Java supports to handle signals in the application (e.g. TERM, or INT). These signals TERM and INT are used to stop the application.

5.1.2. Administration Manager Interface

There are commands that can be sent to the manager. The commands serve to administer processes, tasks and the application. The commands can be sent to the manager e.g. via TCP/IP.

Command	Action
STOPAPP	Stops the application.
STARTPROCESS	Starts the configured process of the given name, e.g. STARTPROCESS PROCESS_1
STOPPROCESS	Stops the configured process of the given name, e.g.

	STOPPROCESS PROCESS_1
SUSPENDTASK	Suspends the configured task of the given name, e.g. SUSPENDTASK TASK_1
RESUMETASK	Resumes the configured task of the given name, e.g. RESUMETASK TASK_1
GETAPPLSTATE	Gets the state of application.
GETPROCESSSTATE	Gets the state of the configured process with the given name, e.g. GETPROCESSSTATE PROCESS_1
SENDEVENT	Sends an event to the queue, e.g. SENDEVENT eventQueue eventData
RECVEVENT	Receives an event from the queue, e.g. RECVEVENT eventQueue

6. AUTOMATED SOLUTION

An instance of the ESB JPA writes messages to the log files. These messages can be monitored. Depending on the messages, reactions (e.g. to START PROCESS) can be set.

The status of processes or scheduled tasks can be monitored as well, using Administration Interface Manager and its administration commands.

6.1. Runtime Logging, Statistical Data and Data Auditing

Parameters of logging can be set by the configuration of log4J. An instance of the ESB JPA can write runtime messages to the log files, conduct a data audit into the log files, or write statistical data to the log files. See the section of the configuration of log4J.

This logging and its messages can be used to set up automated solution of an application built using the ESB JPA.

6.2. Start/Stop-Scripts

There are Start/Stop-scripts with the distribution package of the ESB JPA. These scripts show how to manipulate with an instance of the ESB JPA.

These scripts can be used to set up automated solution of an application built using the ESB JPA.

6.3. Administration Manager Interface

Administration Manager Interface and its commands can be used to monitor the state of an application built using the ESB JPA, and to administer its processes or tasks, and to exchange events between the application and third-party components.

6.4. Integration to the JBoss Enterprise Service Bus

Besides the basic distribution package of the ESB JPA, there is a plug-in that can be used to integrate the functionalities of the ESB JPA into the JBoss ESB of the JBoss Community.

GUI and other features of the JBoss ESB can be used to set up automated solution of an application built using the plug-in and the JBoss ESB.

7. PREDEFINED STANDARDIZED PLUG-INS

There are predefined plug-ins in the standard distribution of the ESB JPA. The overview of plug-ins is described in the following section. Specific configuration arguments of the plug-ins are described in next sections.

7.1. Overview

IONodes

Plugin	Basic Description
FSIONode	I/O node to read data from files or write data to files
JDBCIONode	I/O node to read data from the database or write data to the database (using SQL procedures or direct access to the tables)
SMTPIONode	I/O node to send emails via SMTP
JMSIONode	I/O node to send JMS messages or receive JMS messages
MemIONode	I/O node to read data from the internal memory of the ESB JPA or write data to the internal memory of the ESB JPA
CSVIONode	I/O node to read data from CSV-files or write data to CSV-files
FTPIONode	I/O node to download data from the FTP server or upload data to the FTP server

Transformers

Plugin	Basic Description
TGrepData	node to extract data from the text using the given grammatical rules
TXSLT	node to apply XSL transformation to the given XML
TKEYVAL	node to transform keys and values of JPAData-instance using the given set of transformation rules
TJPATXT / TTXTJPA	nodes to transform data between data formats JPAData-instance and ASCII text
TJPAXML / TXMLJPA	nodes to transform data between data formats JPAData-instance and XML

TJEncrypt	node to encrypt data using a X509-certificate
TJDecrypt	node to decrypt data using a X509-certificate
TFunctionalScript	node to transform JPAData-instance using the set of predefined functional rules
TKVData2Template	node to fill values (and keys) to the given formatted template where the fields {0}, {1}, … are replaced using the values

Routers

Plugin	Basic Description
BasicRouter	node to route the processing of data to the linked child nodes using the content of a specified variable in the IPDC

Error Handlers

Plugin	Basic Description
BasicEHandler	basic error handler to handle error states by putting data to the defined DLD-destination in case of JPADataException or by rolling the transaction back in other cases

DLDHandlers

Plugin	Basic Description
FSDLDHandler	handler to write data to the DLD-destinations represented as local files in the local directory

JoinedActions and Actions

Plugin	Basic Description
SerialJAction	node to perform chained actions (like GroovyScriptAction, or ConsolePrintlnAction)
GroovyScriptAction	action to run a defined method in the Groovy environment
ConsolePrintlnAction	action to print data to the console

RequestReply-modules

Plugin	Basic Description

HTTPRequestReply	node to send an HTTP request to the URL and receive an reply from the URL

Scheduled Tasks

Plugin	Basic Description
Pop3ReaderSchedTask	task to download emails using the POP3 and store contents of emails to files
ScriptRunner	task to run a shell script in the environment of underlying operating system (e.g. Windows, or Unix)
TCPDataProcessorServer	task to receive data via TCP/IP and passed this data to the queue for further processing
AdminServer	administration task to receive administration commands via TCP/IP (Note: commands are defined in the section of Administration Manager Interface)

Data Feeders

Plugin	Basic Description
TextLinesFeeder	node to parse the whole text to single text lines and to feed the related process flow with the single text lines

Connection Pools

Plugin	Basic Description
DBConnPool	connection pool to manage connections for the access to the database
ShMemConnPool	connection pool to manage connections for the access to the internal shared memory of the ESB JPA

Plug-ins and their arguments are described in the following sections.

7.2. IONodes

IONode are modules to receive data from the inputs or write data to the outputs. Specific plug-ins are described as follows.

7.2.1. FSIONode

FSIONode is a plug-in to:

- browse local directory for files and return binary contents of files. Result data is as a set of binary data
- write text/binary data to a local file

Implementation class is: cz.integrators.esbjpaplugins.fs.FSIONode

Arguments are described in the following table.

Argument	I/O Flag	Description
Directory	Input/Output	local directory to browse for input files, or local directory to store output files This mandatory argument must be specified
DataCharset	Input/Output	characters set of data Default value is: UTF-8
Filter	Input	filter of file names when browse for input files If this argument is null, no filter is applied
XMLRootNodeName	Output	root node in the output XML If this argument is null, no root node is set to the output XML
XMLDataRowNodeName	Output	data node wrapping data rows in the output XML (Note: if there is one record or if there are more records in one batch, each record is wrapped to this node) If this argument is null, no data node is set to wrap data rows
AutoGeneratedFilenamesSuffix	Output	suffix for automatically generated names of output files If this argument is null, no suffix in the names of files is used

If dynamicRouting flag is set to true, then the names of files can be generating using the IPDC-thread variables described in the following table.

Variable	Description
OUTPUT_DEST_NAME	name of output file
OUTPUT_DEST_DIR	directory for output files

7.2.2. JDBCIONode

JDBCIONode is a plug-in to:

- browse a table for new data or call SQL procedure to return data. One result set is a set of JPAData-instances
- insert JPAData-instance to a table by calling INSERT or insert JPAData-instance to a database by calling SQL procedure

Implementation class is: cz.integrators.esbjpaplugins.jdbc.JDBCIONode

Arguments are described in the following table.

Argument	I/O Flag	Description
DSConnectionPoolName	Input/Output	name of connection pool to access the database This mandatory argument must be specified. The related pool must be configured
DataFieldsPropertiesFileName	Input/Output	name of properties file to describe data fields. See a sample and comments below. The order of the fields in the properties file must correspond with order of the parameters in the SQL procedure. This argument is mandatory
Procedure	Input/Output	name of SQL procedure to read data from the database or write data to the database Procedure or table must be specified.
IsSQLProcedureWithNoResultCode	Input/Output	true/false> true means that the SQL procedure returns a result code (like ? = procedure(?, ,…)). False means the SQL procedure does not return a result code Default value: false
TABLE	Input/Output	name of table to read data from or write data to Table or procedure must be specified.
IsAllDataInOnePackage	Output	true/false> true means that all data in one data batch are put to one array under the key of related package. False means that one data record represents one JPAData-instance under the key of

		related package
DataCharset	Input/Output	characters set of data Default value: UTF-8

This plug-in uses jars implementing JDBC drivers for underlying databases.

Tho following sample shows the properties file to describe data fields:

```
Jmeno=*:VARCHAR
Prijmeni=*:VARCHAR
Vek=*:INTEGER
Email=*:VARCHAR
Telefon=*:VARCHAR
```

The syntax of one row to describe one data field is: ***key = defaultvalue:type*** . The types are derived from SQL types.

7.2.3. SMTPIONode

SMTPIONode is a plug-in to:

- send emails via SMTP

Implementation class is: cz.integrators.esbjpaplugins.mail.SMTPIONode

Arguments are described in the following table.

Argument	I/O Flag	Description
Hostname	Output	hostname of target server This mandatory argument must be specified
Port	Output	port of target server Default value: not specified
SenderEmail	Output	one email-address of sender This argument or IPDC-thread variable SENDER_IDENTIFICATION must be specified
ReceiverEmail	Output	email-addresses of receivers delimited by semicolon This argument or IPDC-thread variable RECEIVER_IDENTIFICATION must be specified

Subject	Output	common subject of messages This argument or IPDC-thread variable SUBJECT_CONTENT must be specified
MailBodyPattern	Output	local file with template for the body of message
DeliveryMode	Output	TRY/ENSURE TRY means that the plug-in tries to send an email. If there is an error, writes a warning to a log file and continues to the next node ENSURE means that an exception is thrown if the plug-in cannot send an email Default value is: ENSURE
ConnectUser	Output	user identifier to connect to the target server If ConnectUser is specified, the plug-in sends emails via SMTPS
ConnectUserPwd	Output	user password to connect to the target server This argument must be specified if ConnectUser is specified
RootAttachmentsFolder	Output	local directory where to search for files specified in the IPDC-thread variable FILES_TO_ATTACH Default value is: no directory
DataCharset	Output	characters set of data Default value is: UTF-8

The following thread IPDC-variables can change the result behaviour of the plug-in.

Variable	Description
SENDER_IDENTIFICATION	one email-address of sender dynamicRouting flag must be set to true to allow overwriting senderEmail
RECEIVER_IDENTIFICATION	email-addresses of receivers delimited by semicolon

	dynamicRouting flag must be set to true to allow overwriting receiverEmail
SUBJECT_CONTENT	subject of the message
FILES_TO_ATTACH	names of files to attach to the message

7.2.4. JMSIONode

JMSIONode is a plug-in to:

- read text data from a JMS queue
- send text/binary data to a JMS queue

Implementation class is: cz.integrators.esbjpaplugins.jms.JMSIONode

Arguments are described in the following table.

Argument	I/O Flag	Description
InitialContextFactory	Input/Output	Initial context factory (e.g. org.jnp.interfaces.NamingContextFactory) This mandatory argument must be specified
ProviderURL	Input/Output	provider url (e.g. jnp://127.0.0.1:1099) This mandatory argument must be specified
URLPackagesPrefixes	Input/Output	URL packages prefixes (e.g. org.jboss.naming:org.jnp.interfaces for JBossESB) This mandatory argument must be specified
QueueName	Input/Output	name of JMS queue This mandatory argument must be specified
DataCharset	Input/Output	characters set of data Default value is: UTF-8

This plug-in uses jars of the JBoss Community.

7.2.5. MemIONode

MemIONode is a plug-in to:

- get text data from a queue in the internal memory of the ESB JPA
- put text/binary data to a queue in the internal memory of the ESB JPA

Implementation class is:

Arguments are described in the following table.

Argument	I/O Flag	Description
DSConnectionPoolName	Input/Output	name of connection pool to access the internal memory queue of the ESB JPA This mandatory argument must be specified. The related pool must be configured
DataCharset	Input/Output	characters set of data Default value is: UTF-8
Timeout	Input	timeout to wait for data Default value is: 0

7.2.6. CSVIONode

CSVIONode is a plug-in to:

- browse local directory for CSV-files and return contents of files. Result data is as a set of JPAData-instances
- write JPAData-instance to a local CSV-file

Implementation class is: cz.integrators.esbjpaplugins.fs.csv.CSVIONode

Arguments are same like for FSIONode. There is one argument more described in the following table.

Argument	I/O Flag	Description
...		See the plug-in FSIONode
CSVDataDelimiter	Input/Output	delimiter of data fields in a CSV-file This mandatory argument must be specified

7.2.7. FTPIONode

FTPIONode is a plug-in to:

- download files from a remote directory in the FTP server to a local directory
- upload files from a local directory to a remote directory in the FTP server

Implementation class is: cz.integrators.esbjpaplugins.ftp.FTPIONode

Arguments are described in the following table.

Argument	I/O Flag	Description
LocalDirectory	Input/Output	Local directory to browse for input files, or to store output files This mandatory argument must be specified
RemoteDirectory	Input/Output	Remote directory to browse for input files in the FTP server, or to store output files to the FTP server This mandatory argument must be specified
Server	Input/Output	Target FTP server This mandatory argument must be specified
ConnectUser	Input/Output	user to connect to target FTP server This mandatory argument must be specified
ConnectUserPwd	Input/Output	user password to connect to target FTP server This mandatory argument must be specified
IsBinaryMode	Input/Output	true/false> true means binary mode, false means ASCII mode Default value: true
DataCharset	Input/Output	characters set of data Default value is: UTF-8

This plug-in uses commons-net-1.4.1.jar (and related jars) of the Apache Software Foundation.

7.3. Transformers

Transformers are modules to transform input data to output data. Specific plug-ins are described as follows.

7.3.1. TGrepData

TGrepData is a plug-in to:

- extract data from the text using the given grammatical rules

Implementation class is: cz.integrators.esbjpaplugins.transformer.grep.TGrepData

Arguments are described in the following table.

Argument	Description
GrammarRulesFileName	name of local file with grammar rules to extract data from the input text. See the following table This mandatory argument must be specified
DataCharset	characters set of data Default value is: UTF-8
CountOfLinesToAddAfterMatch	count of lines to add after the match Default value is: 0
CountOfLinesToBeforeBeforeMatch	count of lines to add before the match Default value is: 0

Grammatical rules are described in the following table.

Rule	Description
reg_expr_pattern->IGNORE	ignore lines in the input text data matching reg_expr_pattern
reg_expr_pattern->REPLACE_SUBSTRING, [patternToReplace >> target]	replace substrings in the lines of input text data matching the reg_expr_pattern. To replace substrings, use patternToReplace to target
regular_expression_pattern->TRANSIT	transit lines from the input text data matching reg_expr_pattern to output text data

7.3.2. TXSLT

TXSLT is a plug-in to:

- apply XSL transformation to the given XML

Implementation class is: cz.integrators.esbjpaplugins.transformer.xslt.TXSLT

Arguments are described in the following table.

Argument	Description
XSLTTransformFile	name of local file with XSL transformations This mandatory argument must be specified
DataCharset	characters set of data Default value is: UTF-8

7.3.3. TKEYVAL

TKEYVAL is a plug-in to:

- transform keys and values of JPAData-instance using the given set of transformation rules

Implementation class is: cz.integrators.esbjpaplugins.transformer.keyvalue.TKEYVAL

Arguments are described in the following table.

Argument	Description
MappingsFileName	name of local file with transformation rules. See the following table This mandatory argument must be specified
WorkMode	TRY/ENSURE TRY means that the plug-in tries to transform input data. If there is no input data, writes a warning to a log file and continues to the next node ENSURE means that an exception is thrown if the plug-in cannot transform data because of no input data Default value is: ENSURE
DataCharset	characters set of data Default value is: UTF-8

Transformation rules are described in the following table.

Rule	Description
key1:: newKey=vPattern::vToAssign	replaces the variable [key, value] by a variable: • [newKey, vPattern] if the key1 is equal to the key and vPattern does

	not match to the value • [newKey, vToAssign] if the key1 is equal to the key and vPattern matches to the value To match values and patterns, regular expressions are used
$key1:: newKey=vPattern::vToAssign	creates a new variable [newKey, vToAssign] in JPAData-instance for the variable [key, value] if the key1 is equal to the key and vPattern matches to the value To match values and patterns, regular expressions are used
@key1:: newKey=vPattern::vToAssign	creates a new variable [newKey, vToAssign] in the internal memory IPDC for the variable [key, value] if the key1 is equal to the key and vPattern matches to the value To match values and patterns, regular expressions are used

7.3.4. TJPATXT / TTXTJPA

TJPATXT is a plug-in to:

- nodes to transform data between data formats JPAData-instance and ASCII text

Implementation class is: cz.integrators.esbjpaplugins.transformer.txtjpa.TJPATXT

Arguments are described in the following table.

Argument	Description
TransformJPADataIntoKVPairsAsText	true/false> true means to return the content of JPAData-instance as the text where one data field is equal to [key,value]-paired line. False means to return the content of JPAData-instance in the XML syntax where one data field is equal to one <key>value</key>-row Default value is: false
WorkMode	TRY/ENSURE TRY means that the plug-in tries to transform input data. If there is no input data, writes a warning to a log file and continues to the next node

	ENSURE means that an exception is thrown if the plug-in cannot transform data because of no input data Default value is: ENSURE
DataCharset	characters set of data Default value is: UTF-8

TTXTJPA is a plug-in to:

- nodes to transform data between data formats ASCII text and JPAData-instance

Implementation class is: cz.integrators.esbjpaplugins.transformer.txtjpa.TTXTJPA

Arguments are described in the following table.

Argument	Description
GrammarRulesFileName	name of local file with grammar rules to extract data from the input text. See the following table This mandatory argument must be specified
IsEmptyWhiteSpaceAllowed	true/false> true means that empty lines in the text are allowed as data lines. False means that empty lines in the text are ignored. Default value is: true
DataCharset	characters set of data Default value is: UTF-8

Grammatical rules are described in the following table.

Rule	Description
delimiter->SUBSTRINGTODELIMITER,key	gets a substring of input data from the actual startIndex up to the delimiter, assing this substring to a value and puts the result [key, value] to the related JPAData-instance

7.3.5. TJPAXML / TXMLJPA

TJPAXML is a plug-in to:

- nodes to transform data between data formats JPAData-instance and XML

Implementation class is: cz.integrators.esbjpaplugins.transformer.jpaxml.TJPAXML

Arguments are described in the following table.

Argument	Description
XMLRootNodeName	name of root tag in the output XML Default value is: root
SourceDataCharset	characters set of input data Default value is: UTF-8
DestinationDataCharset	characters set of output data Default value is: UTF-8

TXMLJPA is a plug-in to: cz.integrators.esbjpaplugins.transformer.jpaxml.TXMLJPA

- nodes to transform data between data formats XML and JPAData-instance

Implementation class is:

Arguments are described in the following table.

Argument	Description
DataCharset	characters set of data Default value is: UTF-8

These plug-ins uses xercesImpl-2.0.2.jar and jdom.jar (and related jars) of the Apache Software Foundation.

7.3.6. TJEncrypt

TJEncrypt is a plug-in to:

- encrypt data using a X509-certificate

Implementation class is: cz.integrators.esbjpaplugins.security.TJEncrypt

Arguments are described in the following table.

Argument	Description
CipherInstanceType	type of encryption standard (like RSA, or DES). These values are same like for native JSE

CertificateFileName	name of local file with a certificate to encrypt data (use e.g. keytool to generate this certificate)
CertificateType	type of generated certificate (like X.509) . These values are same like for native JSE

7.3.7. TJDecrypt

TJDecrypt is a plug-in to:

- decrypt data using a X509-certificate

Implementation class is: cz.integrators.esbjpaplugins.security.TJDecrypt

Arguments are described in the following table.

Argument	Description
CipherInstanceType	type of encryption (decryption) standard (like RSA, or DES). These values are same like for native JSE
CertificateFileName	name of local file with a certificate to decrypt data (use e.g. keytool to generate this certificate)
CertificateType	type of generated certificate (like X.509) . These values are same like for native JSE

7.3.8. TFunctionalScript

TFunctionalScript is a plug-in to:

- transform JPAData-instance using the set of predefined functional rules

Implementation class is: cz.integrators.esbjpaplugins.script.TFunctionalScript

Arguments are described in the following table.

Argument	Description
FunctionalScriptFileName	name of local file with functional rules to transform data. See the following table This mandatory argument must be specified
WorkMode	TRY/ENSURE TRY means that the plug-in tries to transform input data. If there is no input data, writes a warning to a log file and continues to the next node ENSURE means that an exception is thrown if the plug-in

	cannot transform data because of no input data Default value is: ENSURE
DataCharset	characters set of data Default value is: UTF-8

The syntax of the script is as follows:

```
Rule1:      command1
            command2
            ...
            commandK

Rule2:      command1
            ...
            commandL


RuleM:      command1
            ...
            commandK
```

In the script:

- $-syntax can be used in the script to get variable values, e.g. ${variablekey1}
- brackets can be used to wrap compound conditions, e.g. ((1==1) && (${key1} > 10))
- $EVALUATE_MATH{expression) can be used to evaluate a mathematical expression, e.g. $EVALUATE_MATH{${keyN} + 10})

Functional rules are described in the following table.

Rule	Description
intValue1 == intValue 2	integer comparison
intValue 1 != intValue 2	integer comparison
intValue1 > intValue2	integer comparison
intValue1 < intValue2	integer comparison
stringValue1 == stringValue 2	string comparison
stringValue 1 != stringValue 2	string comparison
stringValue1 > stringValue2	string comparison
stringValue1 < stringValue2	string comparison

Commands are described in the following table.

Command	Description
assign (*key, value*)	assigns a value to the identified with the

	key
replace (*key, value, pattern, replacement*)	replaces the patterns with the replacements in the value of variable identified with the key
substring (*key, value, startIndex, endIndex*)	returns a substring from the value of variable identified with the key. The result substring starts at startIndex and ends at endIndex
printf (*filename, text*)	prints text to a file
sendemail (*mailserver, port, user, password, receiveremail, senderemail, subject, content, ensuredelivery*)	sends an email to the mailserver(port). Parameters of the message are given with receiveremail, senderemail, subject and content. User and password serve to connect to the mailserver. Ensuredelivery can be true/false
callgroovy (*key, scriptfilename, method, data*)	calls a script in the groovy environment. When calling the script, the method is invoked. The method is of the same syntax like for the plug-in GroovyScriptAction
gotostart ()	goes to the first functional rules and starts evaluating the rules from the beginning

7.3.9. TKVData2Template

TKVData2Template is a plug-in to:

- fill values (and keys) to the given formatted template where the fields {0}, {1}, … etc. are replaced using the values

Implementation class is: cz.integrators.esbjpaplugins.transformer.format.TKVData2Template

Arguments are described in the following table.

Argument	Description
OutputBodyTemplateFile	name of local file with a predefined template for output data. This template includes data fields {0}, {1}, … to be replaced by the fields in the related JPAData-instance This mandatory argument must be specified
StartDataLinePattern	regular expression to indicate the first data line to be processed It can be null. If it is null the input text is processed from the beginning
EndDataLinePattern	regular expression to indicate the last data line to be processed

	It can be null. If it is null the input text is processed to the end
DataCharset	characters set of data Default value is: UTF-8
PassDataContentToIPDC	true/false> trues means that result [key,value]-paired data fields are put to the IPDC. False means that result [key,value]-paired data fields are put to the related JPAData-instance Default value is: false

7.4. Routers

Routers are modules to route the processing of data to more branches, e.g. on the content of data. Specific plug-ins are described as follows.

7.4.1. BasicRouter

BasicRouter is a plug-in to:

- route the processing of data to the linked child nodes using the content of a specified variable in the IPDC

Implementation class is: cz.integrators.esbjpaplugins.router.BasicRouter

Arguments are described in the following table.

Argument	Description
ControlVariable	Key of variable to check its content against the pattern. Implicitly, the variable and its content are taken from the related JPAData-instance. If the key is prefixed with @ (e.g. like @INPUT_DEST_NAME) the variable and its value are taken from the IPDC This mandatory argument must be specified
DataRoutingPattern	Pattern to define where to route which data Pattern is defined as the array of fields where one field is defined as reg_expr_pattern::target_linkage_name. The sample can be:

	.* DEST1_.*::OUTDEST1 .* DEST2_.*::OUTDEST2 This mandatory argument must be specified
DefaultRouting	Name of default target_linkage_name if no pattern is matched This argument is not mandatory

7.5. Error Handlers

ErrorHandlers are modules to handle error states in a different way the the default one is. Specific plug-ins are described as follows.

7.5.1. BasicEHandler

BasicEHandler is a plug-in to:

- handle error states by putting data to the defined DLD-destination in case of JPADataException or by rolling the transaction back in other cases

Implementation class is:

Arguments are described in the following table.

Argument	Description
CallEHandlerBody	true/false> true means to execute the body of the error handler as well to handle an error state. False means to skip the body and invoke only the methods of error handler Default value is: true

7.6. DLDHandlers

DLDHandlers are modules to write data to DLD-destinations. Specific plug-ins are described as follows.

7.6.1. FSDLDHandler

FSDLDHandler is a plug-in to:

- write data to the DLD-destinations represented as local files in the local directory

Implementation class is: cz.integrators.esbjpaplugins.dld.FSDLDHandler

Arguments are described in the following table.

Argument	Description
Destination	local directory for files including data to DLD-destination
DLDDataPackageKey	key of package with data

7.7. JoinedActions and Actions

JoinedActions are modules to perform chained actions. Actions are predefined classes to be performed. Specific plug-ins are described as follows.

7.7.1. SerialJAction

SerialJAction is a plug-in to:

- perform chained actions in a serial way (like GroovyScriptAction, or ConsolePrintlnAction)

Implementation class is: cz.integrators.esbjpaplugins.actions.SerialJAction

Arguments are described in the following table.

Argument	Description
Action_	arguments Action_1, Action_2, ... define actions to perform. The values of these arguments define implementation classes of the actions At least one action must be defined
DataCharset	characters set of data Default value is: UTF-8

7.7.2. GroovyScriptAction

GroovyScriptAction is an action to:

- run a method of defined script in the Groovy environment

Implementation class is: cz.integrators.esbjpaplugins.actions.GroovyScriptAction

The method is in the following syntax:

```
def  MethodNameToInvoke (String data) {
  return "Transformed data: \n --->" + data
}
```

Arguments are described in the following table.

Argument	Description
GroovyScriptFileName	local file with the groovy script to invoke This mandatory argument must be specified
MethodNameToInvoke	name of the method to invoke in the script This mandatory argument must be specified

Details about the Groovy environment, you can read on http://groovy.codehaus.org/.

7.7.3. ConsolePrintlnAction

ConsolePrintlnAction is an action to:

- print data to the console

Implementation class is: cz.integrators.esbjpaplugins.actions.ConsolePrintlnAction

No arguments are provided for this action.

7.8. RequestReply-Modules

RequestReply-modules are modules to send a request to the server and receive a reply from the server (e.g. for HTTP protocol or Web Services). Specific plug-ins are described as follows.

7.8.1. HTTPRequestReply

HTTPRequestReply is a plug-in to:

- send an HTTP request to the URL
- and receive an reply from the URL

Implementation class is: cz.integrators.esbjpaplugins.http.HTTPRequestReply

Arguments are described in the following table.

Argument	Description
URL	target URL This argument or IPDC-thread variable TARGET_URL must be specified
HTTPDataCharset	characters set of HTTP data Default value is: UTF-8

RDParameter_Key_	arguments RDParameter_Key_1, RDParameter_Key_2, ... define input parameters for HTTP request No parameter can be specified
DataCharset	characters set of input data Default value is: UTF-8

The following thread IPDC-variables can change the result behaviour of the plug-in.

Variable	Description
TARGET_URL	target URL dynamicRouting flag must be set to true to allow overwriting senderEmail

7.9. Scheduled Tasks

Scheduled tasks serve to run processing of data or to start getting data at planned time. Specific plug-ins are described as follows.

7.9.1. Pop3ReaderSchedTask

Pop3ReaderSchedTask is a plug-in-task to:

- download emails using the POP3 and store contents of emails to files

Implementation class is: cz.integrators.esbjpaplugins.mail.pop3.Pop3ReaderSchedTask

Arguments are described in the following table.

Argument	Description
Hostname	hostname of target server This mandatory argument must be specified
Port	port of target server Default value is: 995
UseSSL	true/false> true means to use POP3S. False means to use POP3 Default value is: false
UserId	email address to access This mandatory argument must be specified

UserPassword	user password to access This mandatory argument must be specified
Folder	name of folder to access (e.g. INBOX) This mandatory argument must be specified
Directory	directory where to store files with received emails This mandatory argument must be specified
DLDDirectory	DLD-destination where to store invalid or NOT accepted emails This mandatory argument must be specified
MaximumCount	maximum count of emails to receive within one day Default value is: unlimited
DataAsAttachment	true/false> true means data is received as attachments of the email. False means data is the part of the body of email. Default value is: true
AcceptedSenders	List of sender email addresses to accept. To describe one address a regular expression is used The syntax is: `address_1  address_2  address_3` Note: delimiter is 2 spaces This argument is not mandatory
DataCharset	characters set of data Default value is: UTF-8

7.9.2. ScriptRunner

ScriptRunner is a plug-in-task to:

- run a shell script in the environment of underlying operating system (e.g. Windows, or Unix)

Implementation class is: cz.integrators.esbjpaplugins.script.ScriptRunner

Arguments are described in the following table.

Argument	Description
ShellCommand	shell script to run This mandatory argument must be specified

7.9.3. AdminServer

AdminServer is a plug-in-task to:

- receive administration commands via TCP/IP (Note: commands are defined in the section of Administration Manager Interface)

Implementation class is: cz.integrators.esbjpaplugins.admin.AdminServer

Arguments are described in the following table.

Argument	Description
Port	port to listen at Default value is: 8020

7.9.4. TCPDataProcessorServer

TCPDataProcessorServer is a plug-in-task to:

- receive data via TCP/IP
- and passed this data to the queue for further processing

Implementation class is: cz.integrators.esbjpaplugins.tcp.TCPDataProcessorServer

Arguments are described in the following table.

Argument	Description
Port	port to listen at Default value is: 8080
WaitSocketTimeout	Timeout to wait for data at the socket Default value is: 1000
DSConnPoolNameForInputData	name of connection pool to access the internal memory of the ESB JPA where received data is stored This mandatory argument must be specified.

	The related pool must be configured
DSConnPoolNameForOutputData	name of connection pool to access the internal memory of the ESB JPA where replies and their data are stored This mandatory argument must be specified. The related pool must be configured
IsDataTerminatedByZeroChar	true/false> true means that data is terminated with 0-char. False means that data is not terminated with 0-chars. In case of true, the TCPDataProcessor waits for the character 0 before processing data Default value is: false
TimeoutToSendReply	timeout to wait at the queue for a reply to the client [milliseconds] Default value is: unlimited
TimeoutToSleepBetweenDataLoads	timeout to sleep between data loads [in milliseconds] Default value is: do not sleep between data loads
MaximumCountOfConnections	Maximum count of opened connections Default value is: 10
MaxDataSize	maximum size of exchanged data Default value is: 20480

7.10.Data Feeders

DataFeeders are modules to split one large data record (e.g. content of one file) to multiple smaller partial data records (e.g. single text lines). Specific plug-ins are described as follows.

7.10.1. TextLinesFeeder

TextLinesFeeder is a plug-in to:

- parse the whole text to single text lines
- and feed the related process flow with the single text lines

Implementation class is:

Arguments are described in the following table.

Argument	Description
IsEmptyWhiteSpaceAllowed	true/false> true means that empty lines in the text are allowed as data lines. False means that empty lines in the text are ignored. Default value is: true
DataCharset	characters set of data Default value is: UTF-8

7.11.Connection Pools

Connection pools serve to manage (open and close) connections for the access to the data storages (e.g. a database).

7.11.1. DBConnPool

DBConnPool is a connection pool to:

- manage connections for the access to the database

Implementation class is: cz.integrators.esbjpaplugins.pool.DBConnPool

Arguments are described in the following table.

Argument	Description
DriverPrefix	Driver prefix (e.g. jdbc:mysql://) This mandatory argument must be specified
DriverClass	Driver class (e.g. com.mysql.jdbc.Driver) This mandatory argument must be specified
Database	Database (e.g. /testESBJPASamplesDB) This mandatory argument must be specified
Server	server and port (e.g. localhost:3306) This mandatory argument must be specified
UserId	user identifier This mandatory argument must be specified
UserPassword	user password

	This mandatory argument must be specified

The above connection string for JDBC is composed as follows:

```
DriverPrefix + Server + Database
```

Jars of appropriate JDBC driver must be in the classpath.

7.11.2. ShMemConnPool

ShMemConnPool is a connection pool to:

- manage connections for the access to the internal shared memory of the ESB JPA

Implementation class is: cz.integrators.esbjpaplugins.pool.ShMemConnPool

Arguments are described in the following table.

Argument	Description
MaxQueueSize	Maximum size of the queue Default value is: 2048
MaxQueueItemSize	maximum size of an item Default value is: 1024
SwapDataFileName	Name of file to swat data if the queue is persistent This argument must be specified if persistence == true
Persistence	true/false> true means data in the queue is persistent. False means data in the queue is not persistent Default value: false
ExpirationTimeout	expiration timeout for validity of data [in milliseconds] Default value: validity of data never expires

8. SAMPLES

The following table provides links to samples in the basic distribution package. Theses samples you can find in the directory samples of the distribution package.

Sample	Basic Description

admin2manager	This sample shows how to send administration commands to the manager
archivator2backup	This sample shows how to run a shell script to archive data
csv2sql	This sample shows how to load data from a CSV-file into a database using SQL procedures.
esbjpa2groovy	This sample shows how to run a method in the Groovy environment
esbjpaframework2javaapp	This sample shows how to use the ESB JPA as the framework in a standalone Java application. (Note: the same is valid for a JEE code).
event2reply	This sample shows how administration commands can be used to exchange events with the ESB JPA from external clients.
feeder2file	This sample shows when and how to use a data feeder.
file2jbossesbjmsq	This samples shows to send data to a JMS queue.
functionalscript2file	This sample shows how a functional script should be used to transform JPAData-instance or to perform commands.
http2file	This sample shows how to download data from a HTTP page to a local file.
kvfile2template	This sample shows how to fill [key,value]-paired data from a local file to a template (e.g. HTML page). [key,value]-paired data (e.g. one line is key = value) is filled to the template using formatted fields {0}, {1}, …
manager2processes	This sample shows how to configure the manager to manage more processes
popmail2file	This sample shows how to download emails using the POP3
request2reply	This sample shows how to configure a request-reply communication. The data storages to exchange data are tables in the database.
router2destinations	This sample shows how to route data to more output destinations
tcp2encrypt	This sample show how to receive and process data in the TCP/IP communication from clients, or how to encrypt data
xml2xml	This sample shows how to transform XML to XML using XSL transformations

9. LICENCE

The product is provided under the GPL. All rights are reserved for INTEGRATORS. All provided plug-ins are based on the freeware licences given by Oracle Java Licence, Apache Software Foundation Licence, or JBoss Community.

10. WHEN TO USE JBOSSESB AND WHEN TO USE ESB JPA

About the JBossESB, you can read on the page http://www.jboss.org/jbossesb/. The product has been developed by large community and it is really powerful software to build runtime solutions for the enterprise service bus. The JBossESB supports fully clusters, EAI, SOA and many other features.

A small problem of perfect solutions is always present. Perfect products are complicated, complex and difficult to use for simple cases. Many times you do not need clusters or SOA. In many cases (especially in the night at 2 o'clock) a little framework to build functional application is quite enough. And this is case to use the ESB JPA. The ESB JPA is a simple framework to build functional applications, and finally these applications you can connect to the JBossESB if you want.

Comparing to other cases, the ESB JPA in the JBossESB is something like the Apache Camel in the Apache ServiceMix. And like the Apache Camel, the ESB JPA can run as a standalone application as well.