EDBC Framework

This document provides a brief overview of the main packages and classes in the EDBC framework. Each section deals with the classes in a package. A class diagram is provided in every section showing the dependencies among classes of the package.

Configuration

This package holds all the classes & interfaces that deal with the configuration details of an Ev ent Driven Business Component.

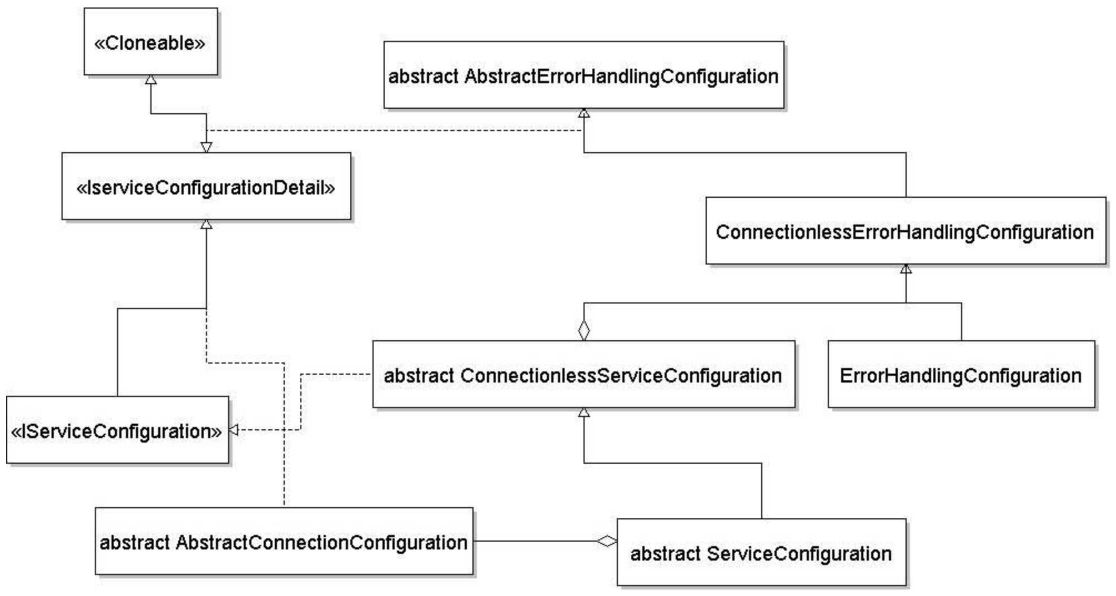

«Cloneable»
abstract AbstractErrorHandlingConfiguration
«IserviceConfigurationDetail»
ConnectionlessErrorHandlingConfiguration
abstract ConnectionlessServiceConfiguration
ErrorHandlingConfiguration
«IServiceConfiguration»
abstract AbstractConnectionConfiguration
abstract ServiceConfiguration

Figure 1: Class diagram showing the dependencies

Interfaces Used

IServiceConfiguration

Specifies the behavior for classes that hold the service configuration.

IServiceConfigurationDetail

This interface is extended by classes which form a part of IServiceConfiguration.

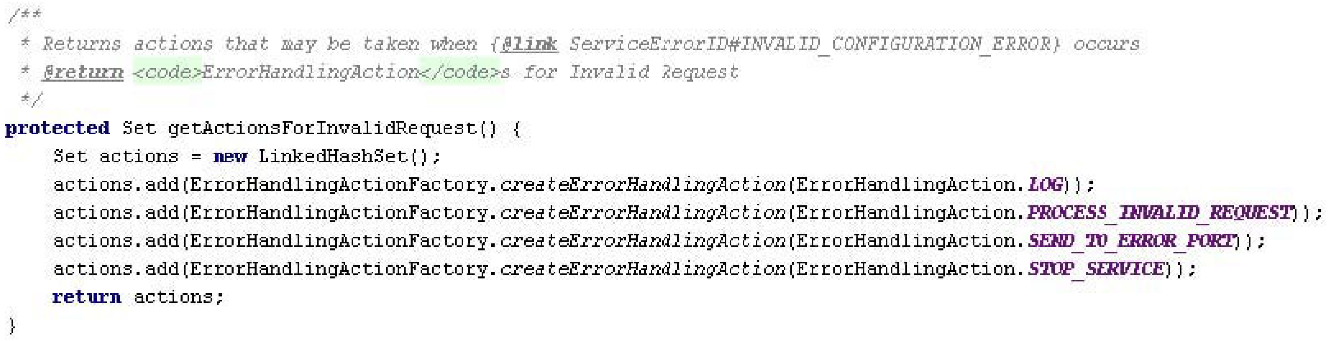
Abstract Classes

AbstractConnectionConfiguration

Abstracts out the behaviour of classes which hold
configuration details (metadata) required to establish a connection to an EIS system.

AbstractErrorHandlingConfiguration

Abstracts out the configuration details for actions to be
taken when an error/exception occurs during the runtime of service. Different actions that can be taken for a specific error id are defined in this class.
For example, the code shown in Figure 2 shows
the error handling actions for an invalid request. Sub classes should override and implement loadErrorActions() to define mappings for different ServiceErrorIDs.



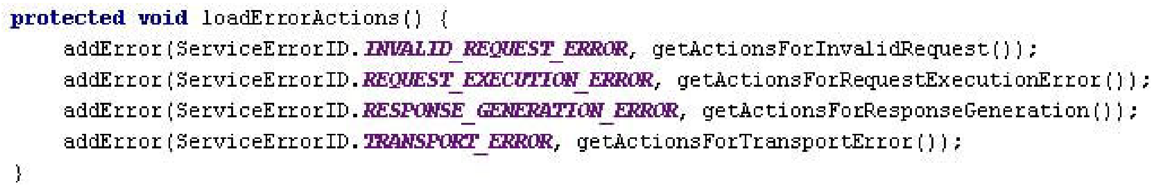
```
/**
 * Returns actions that may be taken when {@link ServiceErrorID#INVALID_CONFIGURATION_ERROR} occurs
 * @return <code>ErrorHandlingAction</code>s for Invalid Request
 */
protected Set getActionsForInvalidRequest() {
    Set actions = new LinkedHashSet();
    actions.add(ErrorHandlingActionFactory.createErrorHandlingAction(ErrorHandlingAction.LOG));
    actions.add(ErrorHandlingActionFactory.createErrorHandlingAction(ErrorHandlingAction.PROCESS_INVALID_REQUEST));
    actions.add(ErrorHandlingActionFactory.createErrorHandlingAction(ErrorHandlingAction.SEND_TO_ERROR_PORT));
    actions.add(ErrorHandlingActionFactory.createErrorHandlingAction(ErrorHandlingAction.STOP_SERVICE));
    return actions;
}
```

Figure 2: Code that specifies the error handling actions for an invalid request.

Classes

ConnectionlessErrorHandlingConfiguration

Defines error handling configuration for services which do not connect to an EIS. Supported ServiceErrorIDs are INVALID_REQUEST_ERROR, REQUEST_EXECUTION_ERROR, RESPONSE_GENERATION_ERROR and TRANSPORT_ERROR. This class extends AbstractErrorHandlingConfiguration and overrides loadErrorActions() method as shown in Figure 3.

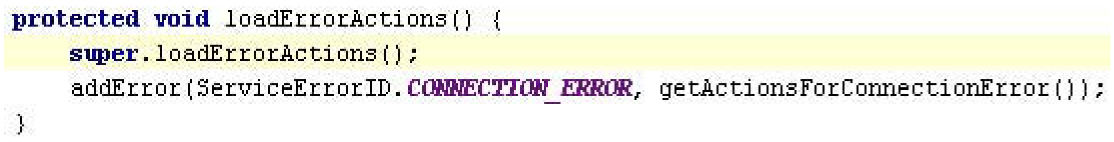


```
protected void loadErrorActions() {
    addError(ServiceErrorID.INVALID_REQUEST_ERROR, getActionsForInvalidRequest());
    addError(ServiceErrorID.REQUEST_EXECUTION_ERROR, getActionsForRequestExecutionError());
    addError(ServiceErrorID.RESPONSE_GENERATION_ERROR, getActionsForResponseGeneration());
    addError(ServiceErrorID.TRANSPORT_ERROR, getActionsForTransportError());
}
```

Figure 3: Loading the remedial actions for different kinds of errors

ErrorHandlingConfiguration

Defines error handling actions that may be taken when an error / exception occurs in a service which connects to an EIS. This class extends ConnectionlessErrorHandlingConfiguration and loads error handling actions for connection related errors as shown in Figure 4.



```
protected void loadErrorActions() {
    super.loadErrorActions();
    addError(ServiceErrorID.CONNECTION_ERROR, getActionsForConnectionError());
}
```

Figure 4: Loading the remedial actions

ConnectionlessServiceConfiguration

Defines the service configuration for services which do not connect to an EIS. This class provides some common properties which might be of use in a service.

ServiceConfiguration

Defines the configuration of services that connect to an EIS. This class extends ConnectionlessServiceConfiguration and adds functionality specific to connection configuration. ServiceConfigurationSerializer – Utility class used to serialize or de-serialize service configuration.

CPS

This package holds all the classes & interfaces used for displaying the component configuration in its Configuration Property Sheet.

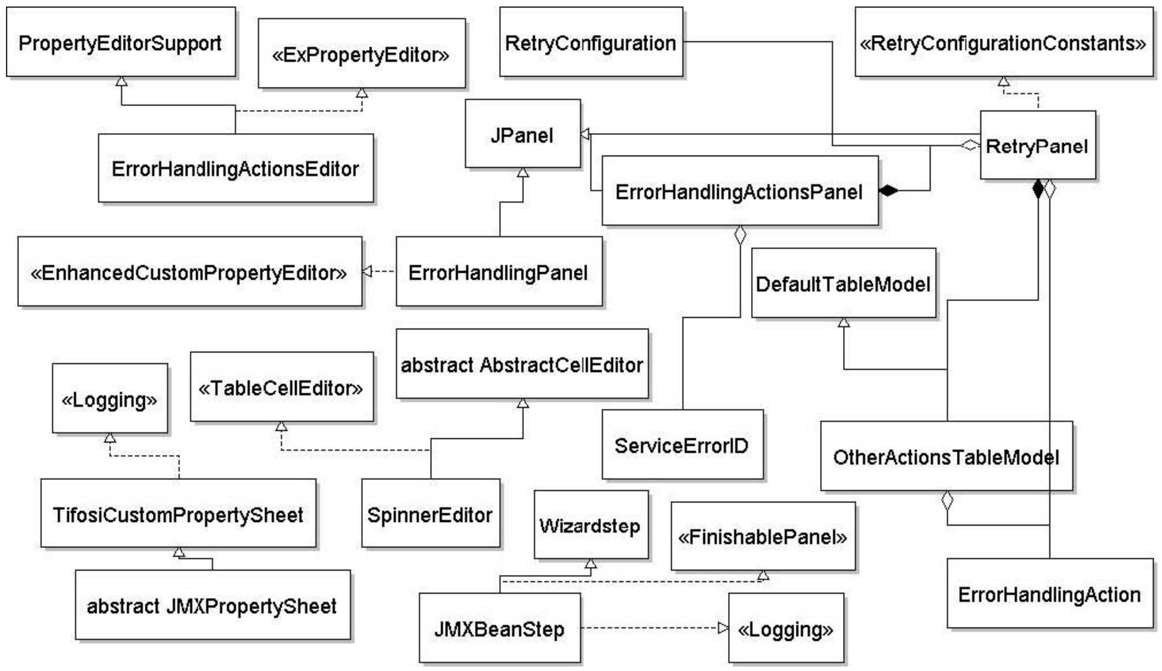


Figure 5: Class diagram showing the dependencies

Interfaces Used

Logging

Provides the anonymous logger used for logging.

Abstract Classes

JMXPropertySheet

Provides implementation to launch Custom Property Sheet for configuring a service. Services should extend this class and override getDefaultConfiguration() to return the configuration that will be shown in the CPS when launched for the first time.

Classes

ErrorHandlingActionsEditor

Property Editor that displays the error handling actions in the Configuration property sheet as shown in Figure 6.

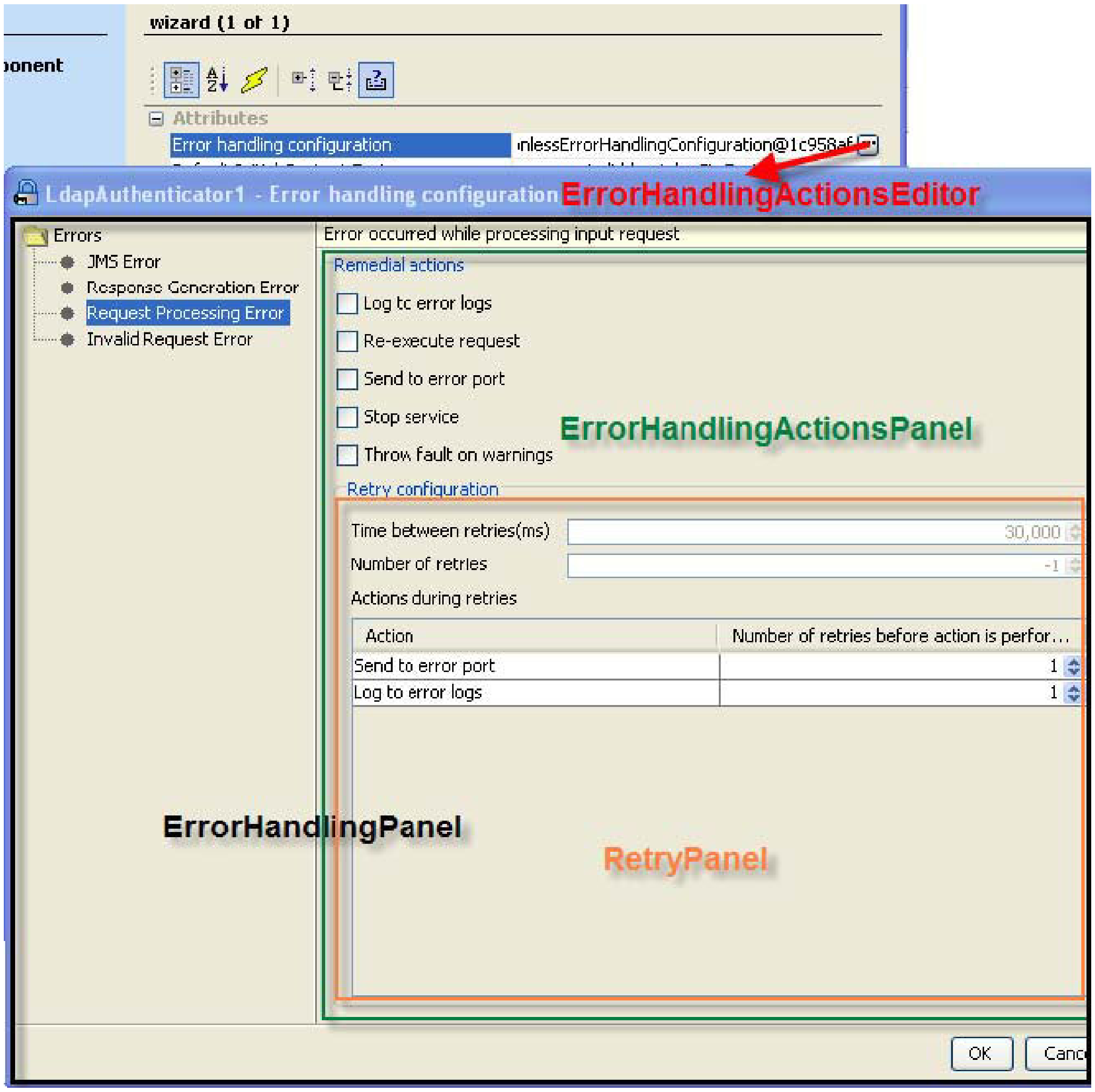


Figure 6: Property Editor showing the Error handling configuration

ErrorHandlingPanel

Panel that shows the possible errors in the service and the respective remedial actions (shown in Figure 6).

ErrorHandlingActionsPanel

Panel that shows the list of remedial actions for the error chosen.
Figure 6 shows the remedial actions for request processing error.

OtherActionsTableModel

Table model to show the actions in the retry panel.

RetryPanel

Panel that displays the configuration required for re-executing a request (shown in Figure 6).

JMXBeanStep

This class is responsible for displaying the Configuration property sheet. Also
handles the execution when Help/Test/Validate buttons are clicked.

Engine

This package holds all the classes & interfaces required for executing the business logic of the component.

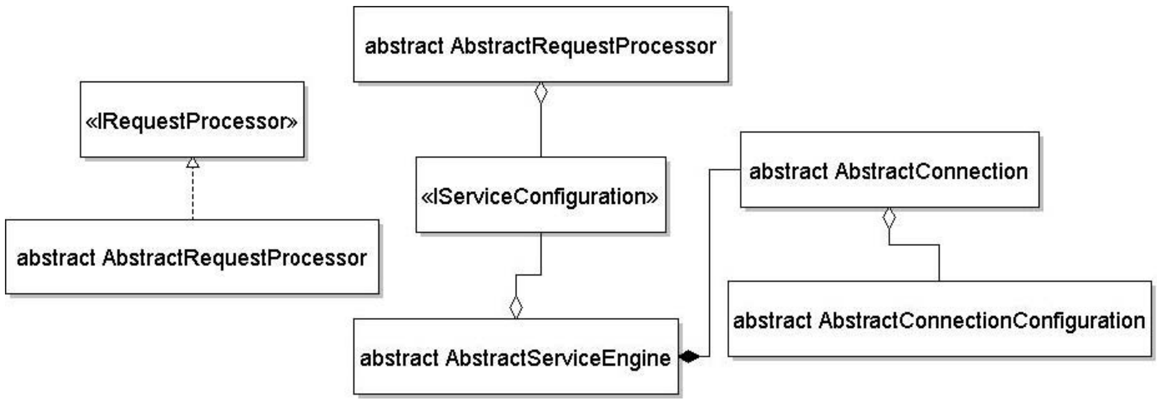

abstract AbstractRequestProcessor
«IRequestProcessor»
abstract AbstractRequestProcessor
«IServiceConfiguration»
abstract AbstractConnection
abstract AbstractConnectionConfiguration
abstract AbstractServiceEngine

Figure 7: Class diagram showing the dependencies

Interfaces Used

IRequestProcessor

Specifies the behavior of class that processes the input request.

Abstract Classes

AbstractConnection

Holds actual physical EIS connection and its metadata. This class
handles connection related operations like connection creation, validation etc.

AbstractRequestProcessor

This class implements validation of request. Sub classes should provide request processing implementation for process(String) and process(Object) methods. This class provides default behavior for process(Message) method.

AbstractServiceEngine

This class holds all data (objects) required for executing the business
logic of the service and drives the service with the help of classes in this package.

Exception

This package holds classes that represent the exceptions thrown by the EDBC components and the actions that could be performed when exceptions occur.

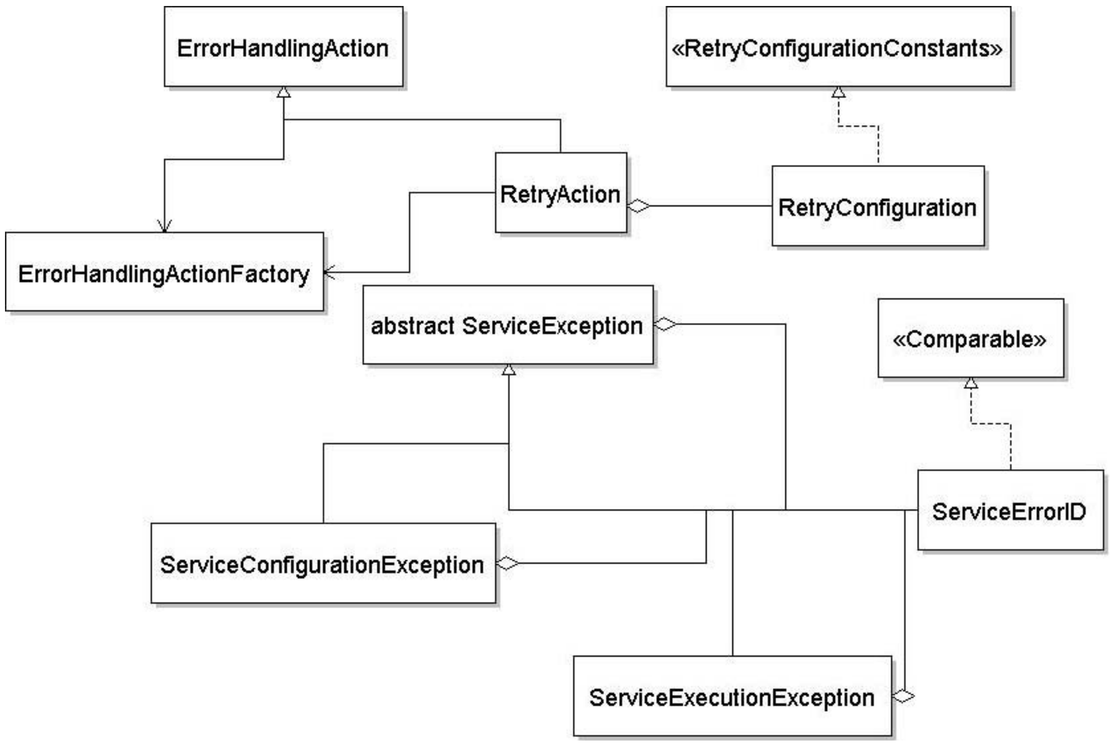

ErrorHandlingAction
«RetryConfigurationConstants»
RetryAction
RetryConfiguration
ErrorHandlingActionFactory
abstract ServiceException
«Comparable»
ServiceErrorID
ServiceConfigurationException
ServiceExecutionException

Figure 8: Class diagram showing the dependencies

Interfaces Used

RetryConfigurationConstants

Constants used for retry configuration.

Abstract Classes

ServiceException

Base class for the exceptions thrown by a service.

Classes

ErrorHandlingAction

Holds the configuration of action that may be taken when an exception occurs during execution of service.

ErrorHandlingActionFactory

Factory class for creating ErrorHandlingAction objects.

RetryAction

Represents an action that repeats a remedial task at regular intervals of time until the action succeeds or repetition count equals configured number of retries.

RetryConfiguration

Holds configuration details of retries and other actions that can be performed during retries of a retry action.

ServiceErrorID

Holds error id’s to indicate different types of exceptions that occur during service execution.

ServiceConfigurationException

Exception class to indicate invalid service configuration details.

ServiceExecutionException

Exception class to indicate an exception that occurred during service execution.

JMS

This package holds the classes that handle the creation of all JMS objects (Connections, Sessions, MessageProducers, MessageConsumers) required for execution of the service instance.

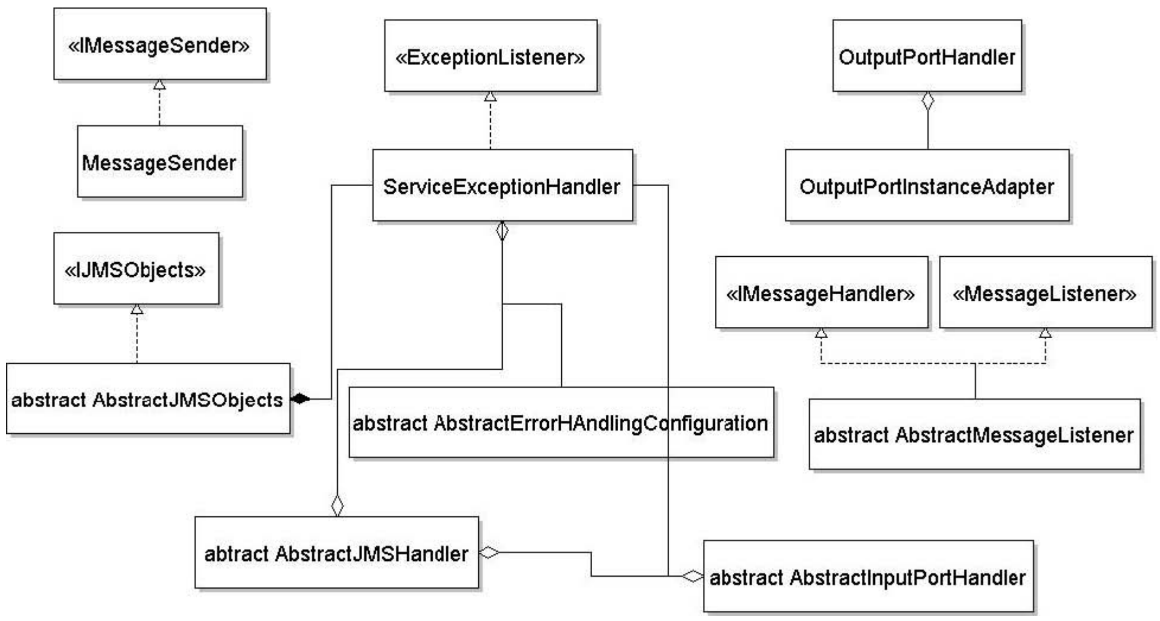

«IMessageSender»
«ExceptionListener»
OutputPortHandler
MessageSender
ServiceExceptionHandler
OutputPortInstanceAdapter
«IJMSObjects»
«IMessageHandler»
«MessageListener»
abstract AbstractJMSObjects
abstract AbstractErrorHAndlingConfiguration
abstract AbstractMessageListener
abtract AbstractJMSHandler
abstract AbstractInputPortHandler

Figure 9: Class diagram showing the dependencies

Interfaces

IJMSObjects

Specifies the behavior for classes which handle creation of all JMS related objects.

IMessageHandler

Specifies the behavior for classes which process the input message.

IMessageSender

This interface specifies the behavior for sending messages onto the required destination.

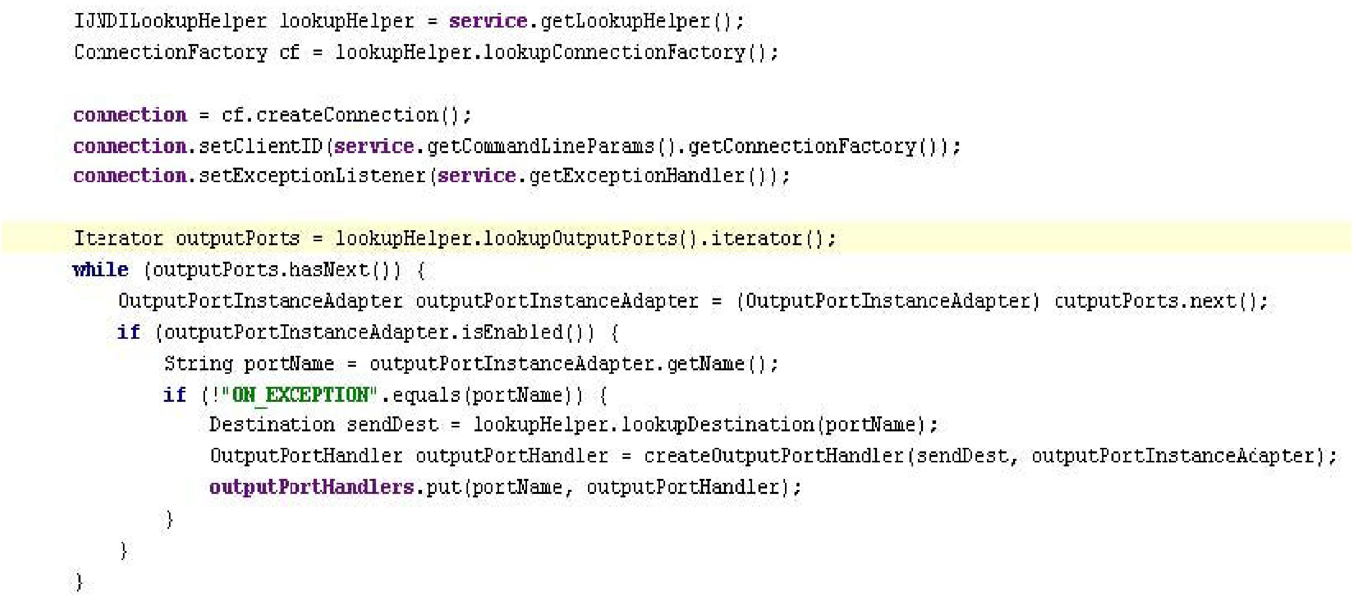
Abstract Classes

AbstractJMSHandler

Handles creation of JMS objects like sessions, MessageProducers, MessageConsumers, and so on.

AbstractJMSObjects

This class provides methods which lookup the connection factory, create and start the connection. This class handles creation of an OutputPortHandler for every Output port and an Input Port Handler for every Input port as shown in figures 10 and 11 respectively.

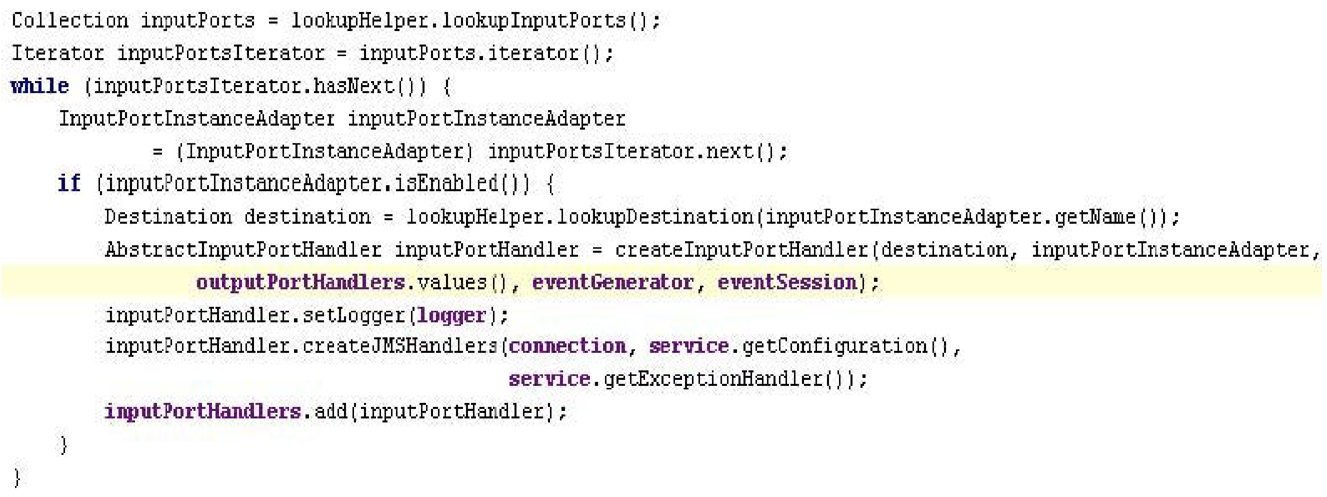


```
IJNDILookupHelper lookupHelper = service.getLookupHelper();
ConnectionFactory cf = lookupHelper.lookupConnectionFactory();

connection = cf.createConnection();
connection.setClientID(service.getCommandLineParams().getConnectionFactory());
connection.setExceptionListener(service.getExceptionHandler());

Iterator outputPorts = lookupHelper.lookupOutputPorts().iterator();
while (outputPorts.hasNext()) {
    OutputPortInstanceAdapter outputPortInstanceAdapter = (OutputPortInstanceAdapter) outputPorts.next();
    if (outputPortInstanceAdapter.isEnabled()) {
        String portName = outputPortInstanceAdapter.getName();
        if (!"ON_EXCEPTION".equals(portName)) {
            Destination sendDest = lookupHelper.lookupDestination(portName);
            OutputPortHandler outputPortHandler = createOutputPortHandler(sendDest, outputPortInstanceAdapter);
            outputPortHandlers.put(portName, outputPortHandler);
        }
    }
}
```

Figure 10: Output port Handler

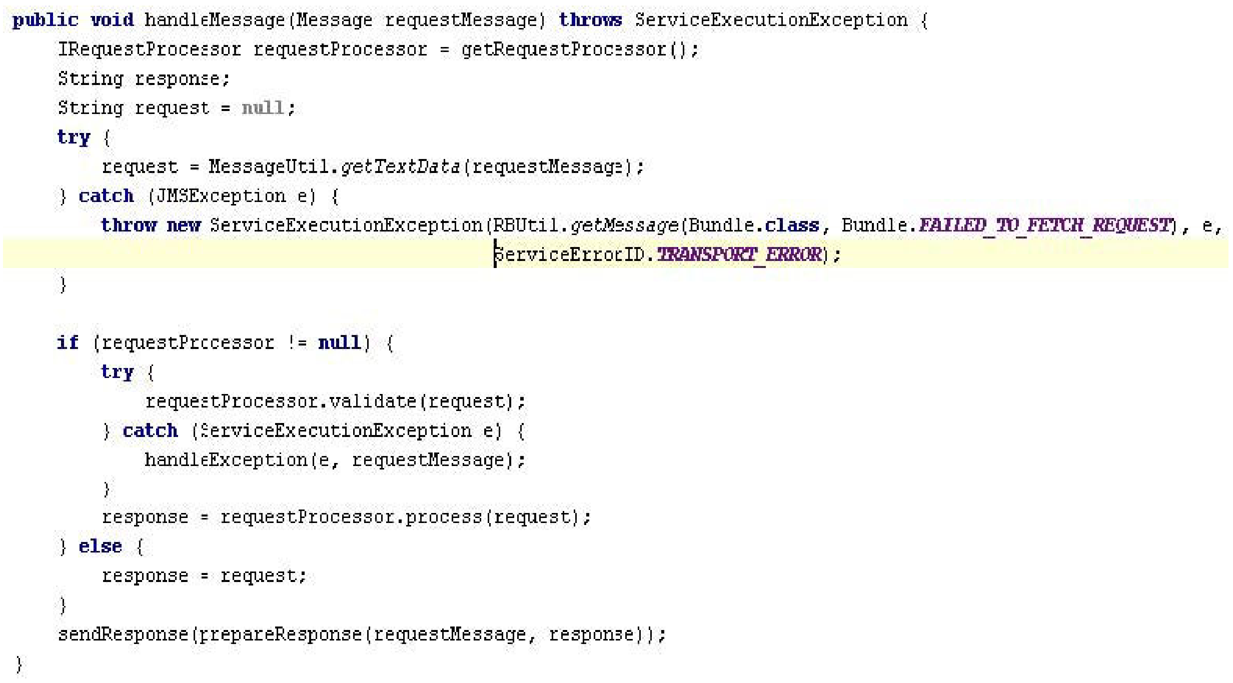


```
Collection inputPorts = lookupHelper.lookupInputPorts();
Iterator inputPortsIterator = inputPorts.iterator();
while (inputPortsIterator.hasNext()) {
    InputPortInstanceAdapter inputPortInstanceAdapter
            = (InputPortInstanceAdapter) inputPortsIterator.next();
    if (inputPortInstanceAdapter.isEnabled()) {
        Destination destination = lookupHelper.lookupDestination(inputPortInstanceAdapter.getName());
        AbstractInputPortHandler inputPortHandler = createInputPortHandler(destination, inputPortInstanceAdapter,
                outputPortHandlers.values(), eventGenerator, eventSession);
        inputPortHandler.setLogger(logger);
        inputPortHandler.createJMSHandlers(connection, service.getConfiguration(),
                                           service.getExceptionHandler());
        inputPortHandlers.add(inputPortHandler);
    }
}
```

Figure 11: Input port Handler

AbstractMessageListener

Provides the abstract functionality for listening messages on ports
and delegating request processing, response delivery. This class provides the default
implementation for handling the input message as shown in Figure 12. Subclasses can overrid
e this method and specify the required handling mechanism.



```
public void handleMessage(Message requestMessage) throws ServiceExecutionException {
    IRequestProcessor requestProcessor = getRequestProcessor();
    String response;
    String request = null;
    try {
        request = MessageUtil.getTextData(requestMessage);
    } catch (JMSException e) {
        throw new ServiceExecutionException(RBUtil.getMessage(Bundle.class, Bundle.FAILED_TO_FETCH_REQUEST), e,
                                            ServiceErrorID.TRANSPORT_ERROR);
    }

    if (requestProcessor != null) {
        try {
            requestProcessor.validate(request);
        } catch (ServiceExecutionException e) {
            handleException(e, requestMessage);
        }
        response = requestProcessor.process(request);
    } else {
        response = request;
    }
    sendResponse(prepareResponse(requestMessage, response));
}
```

Figure 12: Default Implementation for Handling the Input Message

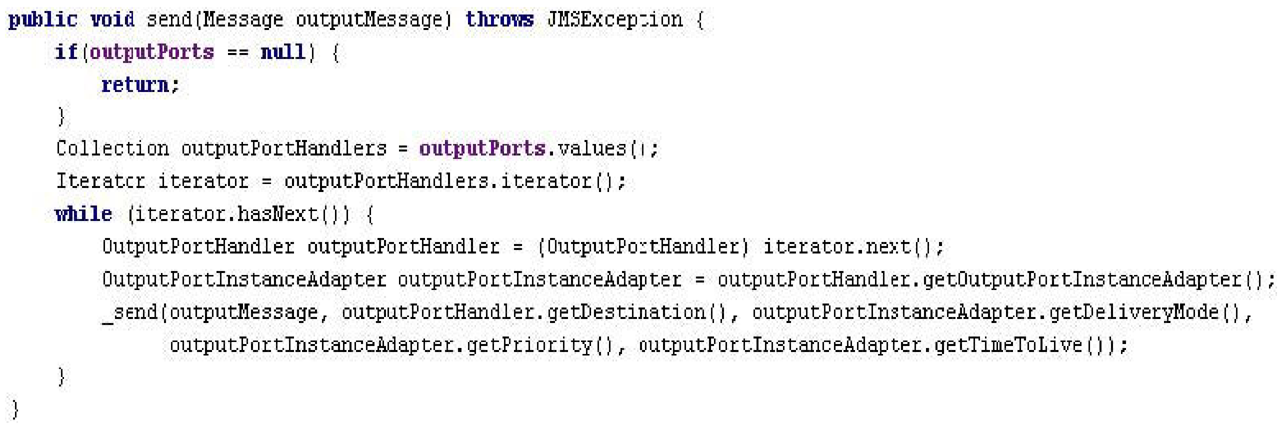
AbstractInputPortHandler

Holds reference to an input port and different JMSHandlers based
on session configuration details of the input port.

Classes

MessageSender

This class provides the implementation for sending messages onto destination(s). The messages can be delivered by specifying the destination or by specifying the output port name. If neither is specified the message will be delivered to all the output ports as shown in Figure 13.

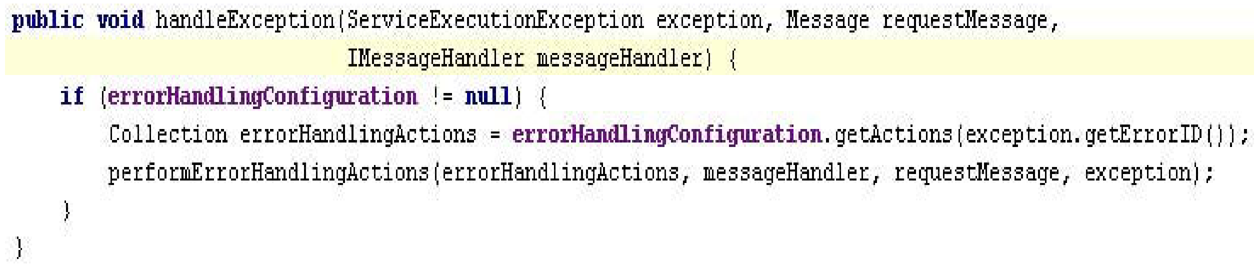


```
public void send(Message outputMessage) throws JMSException {
    if(outputPorts == null) {
        return;
    }
    Collection outputPortHandlers = outputPorts.values();
    Iterator iterator = outputPortHandlers.iterator();
    while (iterator.hasNext()) {
        OutputPortHandler outputPortHandler = (OutputPortHandler) iterator.next();
        OutputPortInstanceAdapter outputPortInstanceAdapter = outputPortHandler.getOutputPortInstanceAdapter();
        _send(outputMessage, outputPortHandler.getDestination(), outputPortInstanceAdapter.getDeliveryMode(),
              outputPortInstanceAdapter.getPriority(), outputPortInstanceAdapter.getTimeToLive());
    }
}
```

Figure 13: Message Output ports

ServiceExceptionHandler

Handles the exceptions thrown in the service. This class performs the error handling actions depending on the errorID as shown in figure 14.



```
public void handleException(ServiceExecutionException exception, Message requestMessage,
                            IMessageHandler messageHandler) {
    if (errorHandlingConfiguration != null) {
        Collection errorHandlingActions = errorHandlingConfiguration.getActions(exception.getErrorID());
        performErrorHandlingActions(errorHandlingActions, messageHandler, requestMessage, exception);
    }
}
```

Figure 14: Error Handling

OutputPortHandler

Holds an output port object and the destination for it.

Other classes

AbstractInmemoryService

Provides the abstract behavior of an inmemory launchable service instance.

AbstractService

This class provides the implementation for starting and stopping the service.