Version 3 Release 23

BOX Messaging Hub Instant Payments

Concept and Implementation

Revision 1.0
Table of Contents

1 INTRODUCTION ........................................................................................................3
  1.1 INSTANT PAYMENTS ..........................................................................................3

2 ARCHITECTURE .........................................................................................................4
  2.1 BOX, THE MESSAGING HUB ..............................................................................4
  2.2 BOX AND INSTAN ST PAYMENTS ....................................................................4
  2.3 SUBMISSION PROFILES ....................................................................................5
      2.3.1 Example SWIFTnet Instant Payment Enrichment .................................5
      2.3.2 Example EBICS Instant Payment Enrichment .......................................6
  2.4 MULTI NETWORK CONNECTIVITY ....................................................................7
      2.4.1 BOX connecting to EBICS .......................................................................7
      2.4.2 BOX connecting to SWIFT .....................................................................8
      2.4.3 BOX connecting to SIA Net .................................................................8

3 CONFIGURATION OPTIONS .....................................................................................9
  3.1 CONNECTIVITY CHANNEL TO SWIFT NETWORK ........................................9
      3.1.1 AGI Plugin (expgi_swift_agi) .................................................................9
          3.1.1.1 Parameters .....................................................................................9
          3.1.1.2 SWIFTnet Connectivity Channel Configuration Example ............10
      3.1.2 Backend Connectivity ............................................................................11
      3.1.3 Lau Key Security ....................................................................................12
  3.2 CONNECTIVITY CHANNEL TO EBICS ..........................................................12
      3.2.1.1 Connectivity Channel Configuration Example ...............................12

4 OPERATING BOX IN NON-PERSISTENCE MODE .............................................14
  4.1 MESSAGE PROCESSING ....................................................................................15
  4.2 ADDRESS BOOK CACHE ................................................................................15
  4.3 BACKOUT COUNT ............................................................................................16
  4.4 DELIVERY COMPOSITION ..............................................................................17

5 OPERATING BOX IN FULL PERSISTENCE MODE ........................................19
  5.1.1 Duplicate Check ...........................................................................................19

6 OFAC CHECK INTEGRATION .............................................................................20
  6.1.1 Asynchronous Communication ..................................................................20
  6.1.2 Workflow Concept ......................................................................................20
      6.1.2.1 Exemplary Workflow of the OFAC Integration .............................20
      6.1.2.2 The Send to SWIFT or Reject .........................................................21
  6.1.3 The Interrupt OFAC Check .........................................................................21
  6.1.4 Check of Already Sent Messages ...............................................................21
  6.1.5 Message Enrichment ...................................................................................21
  6.1.6 Interfaces ....................................................................................................22

7 BOX ARCHIVE FOR INSTAN ST PAYMENTS .................................................23
  7.1 ARCHIVE PERSISTENCE MODE ....................................................................23
  7.2 CONFIGURATION OF BOX ARCHIVER ......................................................23
      7.2.1 Archiver within the Server ...................................................................24
      7.2.2 ARCHIVER as External Tool ...............................................................25

8 MANUAL MESSAGE ENTRY FOR TESTS ....................................................27
  8.1.1.1 Pacs.008.001.02 ..................................................................................27

9 WORKFLOW ...........................................................................................................28
  9.1 EXEMPLARY SWIFTNET WORKFLOW OF AN ARCHIVE- PERSISTENCE MODE ...........................................28
  9.2 MESSAGE EXCEPTION WORKFLOW ...........................................................29
  9.3 SIGNS AND SYMBOLS ....................................................................................29

Table of Graphs
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>SEPA Instant Credit Transfer /SCT Inst) Overview</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2</td>
<td>BOX Messaging Hub Messaging Overview</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Instant Payments Message Processing</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Overview of Instant Payments Configuration of two active sites</td>
<td>5</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Submission Profiles – Example SWIFT Instant Payments</td>
<td>6</td>
</tr>
<tr>
<td>Figure 6</td>
<td>EBICS LCG Configuration</td>
<td>7</td>
</tr>
<tr>
<td>Figure 7</td>
<td>BOX Messaging Hub Connecting to SIAnet</td>
<td>8</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Connecting to SIAnet: Command- and Business Phases</td>
<td>8</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Non-Persistence Mode</td>
<td>14</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Overview Instant Payment Message Processing</td>
<td>15</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Full Persistence Mode</td>
<td>19</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Exemplary OFAC Integration workflow</td>
<td>20</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Archive Persistence Mode</td>
<td>23</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Overview SWIFTnet Instant Payment Workflow (TIPS)</td>
<td>28</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Overview Exception Message Workflow</td>
<td>29</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Workflow Signs and Symbols</td>
<td>29</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Enrichment with Submission Profile, configured in Content Processing Modules-FIA Subm. Prof. Instance</td>
<td>30</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Excerpt Workflow containing Analysis 1</td>
<td>30</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Overview Monitoring in Non-Persistence Mode</td>
<td>31</td>
</tr>
<tr>
<td>Figure 20</td>
<td>BOX Server Health Monitoring Example</td>
<td>32</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Instant Payments

Instant payments – also known as real-time or immediate payments – are defined by the Euro Retail Payments Board (ERPB) as electronic retail payments that are available 24/7/365. They require the immediate or close-to-immediate interbank clearing of the transaction and crediting of the payee’s account with confirmation to the payer (usually within a maximum of 10 seconds of payment initiation).

Instant payment focuses on low value retail payment systems (RPS); which differ from real-time gross settlement systems (RTGS) and distributed ledger payment systems.

Instant payments systems tend to have the following characteristics:

**Immediate Credit**

The funds become available in the payee’s account immediately (within a few seconds) of the payment being initiated by the payer.

**Irrevocability**

Once the payer has initiated the payment, the payment process cannot be cancelled.

**Certainty of Fate**

When the payer initiates the payment, he/she is informed immediately (within a few seconds) whether the payment has successfully reached the payee’s account or not.

The following graph gives an overview of what is achieved by Instant Payments and which components are generally involved in the transaction.

**Figure 1** SEPA Instant Credit Transfer (SCT Inst) Overview

This document is designed and written to outline the BOX Messaging Hub (BOX) Instant Payment concept and implementation for SEPA Credit Transfer (SCT) Instant Payments scheme in Europe.
2 Architecture

2.1 BOX, the Messaging Hub

The BOX represents a financial gateway and Messaging Hub integrating with back-office systems and multiple networks as illustrated below in the graph. BOX is a multi-network solution. Within that, BOX is SWIFT Customer Security Programme (CSP) certified and supports all Swift business areas and interfaces (FIN, Interact, RMA) on the one platform.

![Diagram of BOX Messaging Hub](image)

Figure 2  BOX Messaging Hub Messaging Overview

2.2 BOX and Instant Payments

BOX for Instant Payments is installed on the same code-based platform as it is used for other schemes, such as FileAct, FIN and MX. The setup for Instant Payments is based on messages being read from MQ, these messages are processed, transformed, enriched and finally written to a specific network gateway (SWIFT, SIA, EBICS).
The recommended configuration for an Instant Payments System is two active sites, with an MQ and database cluster, with each site having an active-active BOX Cluster with access to the database and MQ cluster.

### 2.3 Submission Profiles

Submission profiles are used to provide data enrichment with network envelope data. BOX has several submission profiles to handle all network related information. The following highlighted Submission Profiles are used for SWIFT and EBICS Instant Payments.

**Available Profiles:**

- Interact
- FileAct – Put File
- File Act – Get File
- SIA FLS – Send File
- SIA FTS – Send File
- SIA T2S – Send Message/File
- **EBICS – Send Instant Payments Realtime Message**
- EBICS – Interbank File Transfer
- **SWIFT – Send Instant Payments Realtime Message**

#### 2.3.1 Example SWIFTnet Instant Payment Enrichment

**Submission Profile SWIFT**

![SWIFT Submission Profile](image)

---

*Architecture* 5
2.3.2 Example EBICS Instant Payment Enrichment

Submission Profile EBICS

**Interact Profile to send SWIFTNet Instant Payment realtime message**

<table>
<thead>
<tr>
<th>[ - MD ]</th>
<th>Interact Profile Parameters to send SWIFTNet Instant Payment realtime message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Name:</td>
<td>TIPS</td>
</tr>
<tr>
<td>Reference Name:</td>
<td>TIPS</td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
</tr>
<tr>
<td>Active:</td>
<td>Yes</td>
</tr>
<tr>
<td>Filter Regular Expression:</td>
<td></td>
</tr>
</tbody>
</table>

**SWIFTNet Instant Payment Realtime Message Sending Information:**

Sender Distinguished Name(DN): cm-olaf, o=pleasess, o=s-boost
Receiver Distinguished Name(DN): cm-olaf, o=pleasess, o=s-boost
Service Name: swift.sf_systems

Figure 5  Submission Profiles – Example SWIFT Instant Payments

**EBICS Profile Parameters to send Instant Payments Realtime Message**

<table>
<thead>
<tr>
<th>[ - MD ]</th>
<th>EBA Profile Parameters to send Instant Payment Realtime Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Name:</td>
<td>pace.002.001.02_IPRT</td>
</tr>
<tr>
<td>Reference Name:</td>
<td>pace.002.001.02_IPRT</td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
</tr>
<tr>
<td>Active:</td>
<td>Yes</td>
</tr>
<tr>
<td>Filter Regular Expression:</td>
<td>C TrênER{MESSAGE_TYPE}=&quot;pace.002.001.02_IPRT&quot;</td>
</tr>
</tbody>
</table>

**Instant Payment Realtime Sending Information:**

EBICS Client Mandator: PP1BDEFF
EBICS Partner: PP1BDEFF
EBICS Host ID: PP1BDEFF
EBICS Partner ID: PP1BDEFF
User ID: PP1BDEFF
EBICS Order Type: EBICS Partner ID to be used
2.4 Multi Network Connectivity

2.4.1 BOX connecting to EBICS

Figure 6  EBICS LCG Configuration
2.4.2 BOX connecting to SWIFT

BOX uses the Alliance Gateway Instant (AGI) Plugin to connect to the SWIFTNet Instant Messaging solution.

The Alliance Gateway Instant enables the exchange of ISO 20022 messages over IBM MQ or through the Alliance Messaging Hub Instant and acts as a local gateway between a customer's back-office application and the SWIFT network.

The following types of AGI setups are supported:
- a one-node AGI (a single host runs one AGI)
- a three-node AGI (the AGI software is installed on three hosts and operates as a single AGI over the three hosts)

Please refer to chapter 3.1.1 for a detailed description on the Alliance Gateway Instant (AGI) Plugin.

2.4.3 BOX connecting to SIA Net

BOX connects to the SIAnet via a Messaging Integration exchanging Real-Time messages and files. BOX offers the following integrations with SIAnet:
- SEPA / EBA CLEARING : Smart Integrator Advanced, File Transfer Service
- T2S : Smart Integrator Advanced, T2S Protocol
- RNI : EAS, Message Switching Service
- CIT (Assegni – Cheques) : EAS, Fast&Lite (File) Service

Extending into Instant Payments
- SCT-Inst : TIPS, RT1

Figure 7  BOX Messaging Hub Connecting to SIAnet

Real-Time message and bulk message exchange are done using the SIA Smart Integrator Advanced.

BOX registers its infrastructure to SIAnet Central Services and is then connected via a secure link to SIAnet and the transport configuration (queue manager, queue names, queue options) for the link to communicate with the SIAnet infrastructure. By registration BOX, as a Business User, joins a Domain (DOM) with a Business User Address. BOX represents a Business User. For Instant Payments, the Instant Message eXtended (IMX) Service provided by the FEMS XS is used. The following graph depicts the phases Command Phases, including the Logon and Subscription and the Business Phases Send and Receive.

Figure 8  Connecting to SIAnet: Command- and Business Phases
3 Configuration Options

3.1 Connectivity Channel to SWIFT Network

The Connectivity Channel is a dedicated channel to connect to the SWIFT Network using the Alliance Gateway Instant (AGI) Plugin to receive and send SWIFT Instant Messages.

3.1.1 AGI Plugin (expgi_swift_agi)

The BOX Alliance Gateway Instant (AGI) Plugin is configured within the server configuration done using a ChangeSet of the BOX Web Client and is especially adapted to deal with Instant Payment messages.

3.1.1.1 Parameters

The expgi_swift_agi BOX Alliance Gateway Instant (AGI) Plugin has the following configuration parameters in the SWIFT_AGI_PLUGIN section:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAU_KEY_1</td>
<td>At least one key must be configured</td>
<td>AES encrypted LAU key used for HMAC calculation/verification. The AES encryption is done on the 32-character hexascii representation of the 16 bytes binary key. The keys have to match the LAU keys configured in SWIFT_AGI.</td>
</tr>
<tr>
<td>LAU_KEY_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAU_KEY_1_ID</td>
<td>If LAU_KEY_N is configured in the plugin,</td>
<td>ID of the keys configured in LAU_KEY_1, LAU_KEY_2, must match the values configured in SWIFT_AGI.</td>
</tr>
<tr>
<td>LAU_KEY_2_ID</td>
<td>LAU_KEY_ID_N must also be specified</td>
<td></td>
</tr>
<tr>
<td>LAU_KEY_1_VALID_UNTIL</td>
<td>Date in YYYY-MM-DD</td>
<td>LAU_KEY_1_VALID_UNTIL, LAU_KEY_2_VALID_UNTIL: these parameters define the end of validity for LAU_KEY_1, LAU_KEY_2. If left empty, the validity of the key will never expire. The parameter can be set either to an ISO8601 timestamp or to a date in YYYY-MM-DD format. If set to a date, the date will be expanded to the end of the date in UTC time zone, e.g. 2019-01-31 will be expanded to 2019-01-31T23:59:59Z. If both LAU_KEY_1 and LAU_KEY_2 are set, the LAU_KEY_1_VALID_UNTIL, LAU_KEY_2_VALID_UNTIL parameters must be set to different values (keys cannot have same validity period). If both LAU keys are set the following rule applies for HMAC calculation/verification: - If the actual time is before the valid until time of the oldest key, this key will be used for HMAC calculation/verification - if the actual time is after the valid until time of the oldest key but before the valid until</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| LAU_KEY_OVERLAP_PERIOD                    | 2       | Overlap period in hours if both LAU_KEY_1, LAU_KEY_2 are configured (default 2 hours). This parameter allows a reverification of the HMAC of a received message with the other key if:  
  - the verification with the oldest key failed but the actual time is midway the overlap period after the valid until time of the oldest key  
  - the verification with the newest key failed but the actual time is in the overlap period before the valid until time of the oldest key |
| SUPPRESS_CACHE_ALERTS                    |         | This parameter [LCG<lcgname>].SUPPRESS_CACHE_ALERT S to was implemented to reduce system load caused by frequent alerts reporting critical cache state. |

Table 2  AGI Plugin Configuration Parameters

The following chapter shows a sample configuration of the Connectivity Channel and the AGI Plugin linked to this particular channel.

3.1.1.2 SWIFTnet Connectivity Channel Configuration Example

```
[LCG<IPSWIFT>]
2PC_FOR_MESSAGES  NO
2PC_FOR_REPORTINGS NO
APPLICATION_GROUP_NAME IPSWIFT
CACHE_CONTENT YES
CACHE_SIZE 200
CGTW_HOST ; <Protocol>:ModuleID,LCG-Number
CHANNEL_TYPE IP-RT
DISABLE_LCG NO; YES
IMPORTER_COUNT 32
MAX_ITEM_SIZE 1000
OVERFLOW_CACHE_SIZE 0
TRANSMITTER_BLOCK_THRESHOLD 0
TRANSMITTER_COUNT 32
TRANSMITTER_UNBLOCK_THRESHOLD 0
TRANSPORT_COST 0

[LCG<IPSWIFT>.PEXA]
CREATOR_PREFIX TIPS
DEFAULT_CREATOR Intercope
DEFAULT_EXCEPTION_SHORTLABEL INSTP XXX SWI 014 Exception_Messages
DEFAULT_IPS_SHORTLABEL INSTP XXX SW0 013 Process_Incoming_Message
DEFAULT_MPS_INITMODE 2 ; 1 - Instantiated, 2 - Pattern
DEFAULT_REPORTING_SHORTLABEL ;ReportingPattern2
DELIVERY_MONITOR YES
DEVICE_TYPE 0xF002
IMPORT_CHECK_CYCLE 5
MONITOR_CARRIER_DELIVERY NO
MPS_PERSISTENCE_LEVEL FullPersistence
STORAGE_PERIOD

[LCG<IPSWIFT>.F002]
DEFAULT_OUTBOUND_QUEUE TO.SWIFT_AGI
DEFAULT_OUTBOUND_QUEUE_MANAGER $$SRV_QMGR_NAME$$
DEFAULT_REPLY_QUEUE FROM.SWIFT_AGI ;not used, but must be specified
```
3.1.2 Backend Connectivity

The format of messages from a backend application can be either MQMD + file data or MQMD + RFH2 Header + file data.

If the backend application provides only MQMD and file data, the BOX Backend Application plug-in processes the data and generates a message in BOX XML format (MPS).

If the backend application provides an RFH2 Header, the BOX server creates an internal XML data structure that contains all name values of the message as children of the root node (canonical RFH2.xml).

This internal XML is of the form:

```xml
<RFH2>
  <namevalue1> ... </namevalue1>
  <namevalue2> ... </namevalue2>
  <namevalue3> ... </namevalue3>
  ...
</RFH2>
```

The data from this internal XML is transformed into a message in BOX XML format (MPS) by means of XSLT. The message can then (optionally) be enriched with data from a Submission Profile (chapter2.3). The format of Output messages to be routed to a backend application is either MQMD + file data or MQMD + RFH2 Header + file data.

The format to be used depends on the backend application, i.e. on the format that the application expects.

If the backend application expects only MQMD and file data, the Backend Application LCG processes the BOX format message and hands it over to the backend application in the expected format.

If the backend application shall receive a message with RFH2 Header, the message in BOX XML format is transformed into an internal XML data structure by means of XSLT. And follows the format described above.

Please also refer to the ISO20022 Backend Application Plugin in box_plugins.pdf for further information on importing ISO20022 messages (MX, FACT, SWIFT/SIA T2S, SIA FTS) from a backend application as well as for exporting ISO20022 messages to a backend application.
3.1.3 Lau Key Security

- Checksum (HMAC) for RFH2 header data and / or payload data.
- No encryption of payload data.
- Different LAU-Keys for different back-office applications for all messages and technical responses.

```
[LCG<TRAVIC_IP_IN_01>] F002
CGTW_HOST ; <Prococol>:ModuleID,LCG-Number
CHANNEL_TYPE IP-RT
APPLICATION_GROUP_NAME TRAVIC_IP_IN
;DEFAULT_DELIVERY_COMPOSITION 0x012101 // include origination report and owner attributes
SUPPORTED_ADDRESSTYPES IPRT_EB
2PC_FORMESSAGES NO
2PC_FORREPORTINGS NO
CACHE_CONTENT YES
CACHE_SIZE 200
MAX_ITEM_SIZE 1000
OVERFLOWCACHE_SIZE 0
IMPORTERCOUNT 10
TRANSMITTERCOUNT 18

[LCG<TRAVIC_IP_IN_01>.PEXA]
IMPORTCHECKCYCLE 2
DEVICE_TYPE 0xF002
PEXA_LIBRARY eximf002_cl
CREATORPREFIX $R$PFX1$
DEFAULTCREATOR $R$DEFAULT_CREATOR$
DEFAULTOWNER $R$DEFAULT_CREATOR$
DEFAULTIPS_SHORTLABEL INSTPselectedIndexIncomingMessage
DEFAULT_MPS_INITMODE 2
DEFAULTEXCEPTIONLABELPREFIX $R$PFX1$
DEFAULTEXCEPTION_SHORTLABEL INSTPselectedIndexExceptionMessages
DELIVERY_MONITOR NO
MONITOR_CARRIER_DELIVERY NO
STORAGE_PERIOD 24
MPS_PERSISTENCE_LEVEL NoPersistence

[LCG<TRAVIC_IP_IN_01>.F002]
PLUGIN_LIBRARY_NAME expgi_travic_ip
LOCAL_QUEUE_MANAGER $R$QMGR$
DEFAULTOUTBOUND_QUEUE_MANAGER $R$QMGR$
DEFAULTOUTBOUND_QUEUE $R$TO.TRAVIC_IP$
INBOUND_QUEUE $R$FROM.TRAVIC_IP$
DEFAULT_REPLY_QUEUE_MANAGER $R$QMGR$
```

3.2 Connectivity Channel to EBICS

3.2.1.1 Connectivity Channel Configuration Example
DEFAULT_REPLY_QUEUE           $$R$FROM.TRAVIC_IP$$
TRASH_QUEUE_NAME              TRASH
;MQ_USER_IDENTIFIer            mqm
MAX_MSGLEN_IN_GROUP           0
DELIVERY_REPORT_GENERATION    0;4
                        ; 0 // delivery report is submission report
                        ; 1 // delivery report through COA
                        ; 2 // delivery report through COD
                        ; 3 // delivery report through PAN/NAN
                        ; 4 // delivery report through reply
MESSAGE_DUMP_LIMIT            100000
GENERATE_COMMAND_REPORT      NO ; YES
EXCEPTION_BACKOUT_LEVEL       10
TRASH_BACKOUT_LEVEL           20
MQ_MSG_PERSISTENT             NO

[LCG<TRAVIC_IP_IN_01>.F002.TRAVIC_IP_PLUGIN]; no config parameters (yet)
4 Operating BOX in Non-Persistence Mode

The non-persistence mode allows BOX to operate without database connectivity. The initial database connection is used to read the workflow configuration, but no operational message data is stored in the database. To allow this mode, not only the processing of messages, which are not written to a database with all its attributes, but also the composition messages to be delivered has changed. It is important to understand, the concept of message processing to avoid any configuration mistakes.

![Transaction Security Window](image)

**Figure 9**  Non-Persistence Mode

**Dispatcher**
Collecting all data for delivery and submits it to the destination LCG Transmitter.

**LCG (Local Channel Group)**
The Transmitter sends the order received from the Dispatcher to the destination. The Importer receives message and writes it to the Cache. Unit of Work is the Outgoing transaction, unless rolled back.

**MPS Handler**
Processes the configured Workflow (DLI, TGI, CPI, SBI, WTI).
4.1 Message Processing

The Instant Payments message flow can be roughly divided into 4 important steps:

- **STEP 1**: Message is read from a queue and transferred to the BOX Server for processing
- **STEP 2**: Message is processed involving a transformation, an enrichment and encoding
- **STEP 3**: Message is written to the SWIFTnet network, a technical response/acknowledgement is received
- **STEP 4**: An acknowledgement is transferred to the sender. If the delivery was successful, the message is now deleted from the queue. If the delivery was unsuccessful, the cycle of STEP 1 till STEP 4 will restart, until the configured ‘Backout Count’ has reached its configured maximum. Please refer to chapter 4.3.

![Diagram](image)

Figure 10 Overview Instant Payment Message Processing

4.2 Address Book Cache

Address book caching has been implemented to accelerate message processing by avoiding DB-access to address books during message dispatching. In such scenarios routing destinations are preferably configured inside a pattern through delivery instruction pattern destinations in a fixed manner as recipients or (even faster) recipient addresses.

Configuration Central Server

Example

```plaintext
[MPS_HANDLER]
CACHE_AB01 node:demo:demo:ISP_AddressBook
```
**Parameter** | **Description**  
---|---  
CACHE_AB<postfix> | This parameter may be used to accelerate message processing by avoiding DB-access to address books during message dispatching. In such scenarios routing destinations are preferably configured inside a pattern through delivery instruction pattern destinations in a fixed manner as recipients or (even faster) recipient addresses. Address book specification (keyword value) uses following syntax: `<OwnerType>:<ClientPrefix>:<OwnerShortname>:<AddressbookShortname>` with  
OwnerType = Node | User , ClientPrefix is the client prefix of the owner of address book, Shortname is UPM-short name of owner of address book and AddressbookShortname specifies the short name as assigned to this address book. Actual keywords may be CACHE_AB01 or CACHE_AB_FIN, It is possible to cache several address books. Be aware that (currently) changes on cached address books might refresh the cache only after server module restart.  

This parameter might be used in persistent message processing also. Fully non-persistent messages (absolutely no database entry) use a different MPS-ID allocation algorithm.  

**IMPORTANT**  
As configuration data is read from the database, MP/O modules still do require database access during start-up. Server modules processing non-persistent messages should disable Gap-Detection (other security measure might be enabled).  

### 4.3 Backout Count

The Backout Count is a vital part of the Instant Payments Message Processing in Non-Persistence Mode, as this configuration limits the number of processing cycles, hence preventing the BOX Server falling into a processing loop - in the event of a consecutive message delivery failure.

| **Parameter** | **Default** | **Description**  
---|---|---  
SECTION Exchange Adapter F002 | |  
TRASH_BACKOUT_LEVEL | 0 | This parameter may be used to trigger a special MQ exception handling for MQ queue entries using an excessive backout count. Setting this parameter to 0 disables this function.  
This special handling includes copying the message into configured TRASH-Queue (parameter TRASH_QUEUE_NAME in this section), requested MQ-Reporting and copying into dead-letter-queue (if configured: USE_DEAD_LETTER_QUEUE). If fully persistent processing is configured, then an exception MPS is created also.  
If a queue entry carrying a backout level equal to or larger than the configured value is read, then the described processing is performed. See also parameter EXCEPTION_BACKOUT_LEVEL in this same section and be aware that this check is performed prior to exception-backout check.  

4.4 Delivery Composition

The delivery composition describes the parts, a message is made of for delivery and can be configured according to specific requirements. The configuration can be done either in the respective channel or in the Delivery Instruction of the workflow, though it is quite often simply set to the channel’s settings as default.

The general delivery composition and its possible values to be configured in the channel’s configuration:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTION [LCGXXX]</td>
<td>(derived from channel type)</td>
<td></td>
</tr>
<tr>
<td>DEFAULT_DELIVERY_COMPOSITION</td>
<td></td>
<td>0x0000100: include rendered content 0x0000010: include original content (mutual exclusive with 0x0000100) 0x0000001: include report data 0x0001000: include report item (only possible with delivery reports) 0x0002000: include generic attributes of content version 0x0010000: include effective properties of default owner</td>
</tr>
</tbody>
</table>

Example: DEFAULT_DELIVERY_COMPOSITION = 0x00100101

IMPORTANT

As messages are not written to a database in the Non-Persistence Mode, the delivery composition must change.
The Non-Persistence delivery composition and its possible values to be configured in the channel's configuration or workflow ‘Delivery Instruction’:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTION [LGGXXX]</td>
<td></td>
</tr>
<tr>
<td>DEFAULT_DELIVERY_COMPOSITION</td>
<td>0x0000010 (derived from channel type)</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

0x0000010: include original content

Example: `DEFAULT_DELIVERY_COMPOSITION = 0x0000010`
5 Operating BOX in Full Persistence Mode

![Diagram]

**Dispatcher**
Collecting all data for Delivery and submits it to the destination LCG Transmitter.

**Archiver**
Archives the MPS to the Online Archive, when processing is finished.

**LCG (Local Channel Group)**
Transmitter sends the Order received from the Dispatcher to the destination Importer, which in turn receives a message and writes it to the Cache and the Database (Unit of Work)

**MPS Handler**
Processes the configured Workflow (DLI, TGI, CPI, SBI, WTI)

5.1.1 Duplicate Check
Duplicate checking, if required, is only possible in Full Persistence Mode.
6 OFAC Check Integration

OFAC is the Office of Foreign Asset Control, part of the U.S. Department of Treasury. OFAC is responsible for administering and enforcing economic and trade sanctions against certain nations, entities and individuals. OFAC maintains a listing of these restricted counter parties in a document called the "Specially Designated Nationals List" (SDN).

The BOX OFAC Check Integration uses IBM WebSphere MQ and is included in the BOX workflow.

6.1.1 Asynchronous Communication

Architectural Overview

SWIFT input messages are routed by the BOX workflow to an "OFAC Check Waiting Queue". A CPI with Custom Mode "OFACCheck" writes these messages to an MQ queue. The OFAC application reads these messages and writes either a SWIFT NAK or the original message to the ReplyToQueue. The result of the check is stored with the message. Based on this information BOX decides whether the message is further processed or interrupted.

6.1.2 Workflow Concept

New IPS “Forward to OFAC”

Based on the workflow BOX decides whether messages must be routed to the OFAC check. For this purpose a new IPS “Forward to OFAC” is implemented consisting of:

➢ An SBI (Analysis 1) takes the decision if the OFAC check has to be executed
➢ A CPI Writes the message to an MQ queue and waits for the result which is appended to the message as report.
➢ An SBI (Analysis 1) Checks the report. If the message is rejected it is either routed with a TGI to an application queue "Declined by OFAC" or sent back with a DLI to the backend application as “merged Ack” (generated by the backend channel).

6.1.2.1.1 Exemplary Workflow of the OFAC Integration

![Exemplary OFAC Integration workflow](image)

Figure 12  Exemplary OFAC Integration workflow
6.1.2.2 The Send to SWIFT or Reject

Application Queue “Declined by OFAC“
Messages which have been routed to the “Declined by OFAC“ queue are manually processed including the following operations:

➢ Forward to SWIFT, Release the message and continue processing
➢ Route to Backend, Reject the message and send “Merged Ack“ to the backend application.
➢ View OFAC Result Visualization of the message together with the result of the OFAC check

6.1.3 The Interrupt OFAC Check

Application Queue “Wait For OFAC“
Three operations (tasks) are provided for messages queued in “Wait For OFAC“:

➢ Interrupt OFAC check (multi selection possible)
  The asynchronous CPI is immediately terminated and a report (interrupted by operator) appended to the message. This report has the same format as a report generated by the OFAC check. A subsequent SBI (Analysis 1) decides if the message is sent or routed to a backend application
➢ MPS Details
  Shows details of the message
➢ Show
  Shows the payload of the message
➢ The IPS “RouteToWBIFN_ACK“ is extended by an SBI (Analysis 1) analysing the result of the OFAC check. If the message has not yet been verified it is routed to the application queue „OFAC Check after transmission”.

6.1.4 Check of Already Sent Messages

The application queue “OFAC check after transmission“
Messages which have been sent without a valid OFAC check are routed to the queue “OFAC check after transmission“. A CPI writes the message to a MQ queue and waits for the result of the OFAC check. The following two operations (tasks) are provided for this queue:

➢ MPS Details
  Shows details of the message
➢ Show
  Shows the payload of the message

6.1.5 Message Enrichment

Message format extensions
The result of the OFAC check is stored in the following folder:

```
<meadow>
<OFACValidationData>
</OFACValidationData>
</meadow>
```

The exact structure of the folder will be defined during development:
<meadow>
<OFACValidationData>
<MessageValidationStatus numVal="2">Valid</MessageValidationStatus>
<MessageValidationDescription>Message is valid</MessageValidationDescription>
</OFACValidationData>
</meadow>

6.1.6 Interfaces

The interface between BOX and the OFAC application is IBM WebSphere MQ. All messages, which are to be checked are written to an MQ queue. A temporary queue with a dynamic queue name is specified as ReplyToQueue. The queue name is unique for each message forwarded to the OFAC application. When the OFAC application rejects a message, it sends back a pseudo SWIFT NAK. If the messages pass the OFAC check the message is sent back in wire format.
7 BOX archive for Instant Payments

7.1 Archive Persistence Mode

The Archive Persistence Mode caters for a possible roll back and is based on caching message data before writing it to the archive database.

![Diagram of Archive Persistence Mode](image)

**Dispatcher**

The Dispatcher collects all data for Delivery and submits it to the destination LCG Transmitter.

**Archiver**

This archives the MPS to the Online Archive, when processing is finished.

**LCG (Local Channel Group)**

The transmitter sends the order received from the Dispatcher to the destination. The importer receives message and writes it to the Cache. The Unit of Work is the Outgoing transaction, unless rolled back.

7.2 Configuration of BOX Archiver

Keeping track of messages for Instant Payments in a Non-Persistence Mode is quite different to a Mode, where messages are written to a database and the archive holds here a central role as a journal and an archive rather than just an archive.

Viewing messages is a central part of the BOX Web Client, but without an existent Message Warehouse, views must be directed toward the archive. But within this particular context, it is by choice to operate an archive, it is not mandatory for the Non-Persistence Mode.
The configuration is set within the respective channel configuration and is no different to any other archive setup and

The Archiver can be configured either to run as an external tool ('offline') or to run within the Server ('online').

7.2.1 Archiver within the Server

If the Archiver runs within the Server, the Server’s configuration file must include the section [SECURITY] containing the security related parameters and the section [BOX_ARCHIVER] containing at least the mandatory parameter SUPPORTED_MESSAGE_TYPES. Additionally the parameter ARCHIVER in section [MPS_CACHE] must specify the archive library box_archiver.

The database connection related parameters are per default read from the section [DB_INTERFACE] in the Server’s configuration file (i.e. the Archiver uses the same database connection as BOX).

<table>
<thead>
<tr>
<th>Section</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section [BOX_ARCHIVER]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUPPORTED_MESSAGE_TYPES</td>
<td></td>
<td>This parameter defines a list of message types that are archived by the BOX Archiver. Currently the following message types are allowed: FIN, SWIFT (alias for FIN), MX, FACT, SIA_FTS, RNI, T2SMSG, T2SFILE, SIA_FLS</td>
</tr>
<tr>
<td>IGNORED_MESSAGE_TYPES</td>
<td></td>
<td>This parameter defines a comma separated list of message types that are to be ignored by the BOX archiver. The message types that can be used are the same as for SUPPORTED_MESSAGE_TYPES. If the message type of an MPS is not listed in SUPPORTED_MESSAGE_TYPES but in IGNORED_MESSAGE_TYPES, the archive flag of the MPS will not be set to NEVERARCHIVED but to NOTARCHIVED.</td>
</tr>
<tr>
<td>DATA_COMPRESSION_METHOD</td>
<td></td>
<td>Data compression method used for archiving. Default value is ‘No compression’.</td>
</tr>
<tr>
<td>ARCHIVE_FILE_DATA</td>
<td>NO</td>
<td>The box_archiver allows archiving file transfer data for SWIFT FileAct, T2S File and SIA FTS messages. If this config parameter is set to YES (default value is NO), the file transfer data will be written to a separate table &lt;qualifier&gt;.FACTARCHFD_&lt;partition_postfix&gt; [year_postfix&gt;].</td>
</tr>
<tr>
<td>Section [MPS_CACHE]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CACHE_SIZE</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>MAX_ITEM_SIZE</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ARCHIVER</td>
<td>box_archiver</td>
<td>This parameter specifies an application (and customer) specific archive library. This archive library is used to generate an archive entry for an MPS. An empty value disables archiving.</td>
</tr>
</tbody>
</table>

Further parameters in these sections and their description can be found in the BOX Configuration Guide.
7.2.2 ARCHIVER as External Tool

If the Archiver runs as an external tool, the configuration file of the tool must contain the following sections:

[SECURITY]
This section is mandatory. It contains security related parameters.

<table>
<thead>
<tr>
<th>Section</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPPORTED_MESSAGE_TYPES</td>
<td></td>
<td>This parameter defines a list of message types that are archived by the BOX Archiver. Currently the following message types are allowed: FIN, SWIFT (alias for FIN), MX, FACT, SIA_FTS, RNI, T2SMG, T2SF, SIA_FLS</td>
</tr>
<tr>
<td>IGNORED_MESSAGE_TYPES</td>
<td></td>
<td>This parameter defines a comma separated list of message types that are to be ignored by the BOX archiver. The message types that can be used are the same as for SUPPORTED_MESSAGE_TYPES. If the message type of an MPS is not listed in SUPPORTED_MESSAGE_TYPES but in IGNORED_MESSAGE_TYPES, the archive flag of the MPS will not be set to NEVERARCHIVED but to NOTARCHIVED.</td>
</tr>
<tr>
<td>DATA_COMPRESSION_METHOD</td>
<td></td>
<td>Data compression method used for archiving. Default value is 'No compression'.</td>
</tr>
<tr>
<td>ARCHIVE_FILE_DATA</td>
<td>NO</td>
<td>The box_archiver allows archiving file transfer data for SWIFT FileAct, T2S File and SIA FTS messages. If this config parameter is set to YES (default value is NO), the file transfer data will be written to a separate table (&lt;qualifier&gt;.FACTARCHFD_&lt;partition_postfix&gt; [year_postfix]).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE_NAME</td>
<td></td>
<td>Name of database used by BOX.</td>
</tr>
<tr>
<td>DATABASE_QUALIFIER</td>
<td></td>
<td>Name of database qualifier used by BOX.</td>
</tr>
<tr>
<td>DATABASE_USERNAME</td>
<td></td>
<td>Name of database user used for logging in</td>
</tr>
<tr>
<td>DATABASE_PASSWORD</td>
<td></td>
<td>Encrypted password for logging into database</td>
</tr>
<tr>
<td>DATA_SOURCE_NAME</td>
<td></td>
<td>Leave blank!</td>
</tr>
<tr>
<td>DB_INTERFACE_LIBRARY</td>
<td></td>
<td>mp_db2cli (for db2) or mp_oraoci (for Oracle).</td>
</tr>
<tr>
<td>CACHE_SIZE</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>MAX_ITEM_SIZE</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ARCHIVER</td>
<td>box_archiver</td>
<td>This parameter specifies an application (and customer) specific archive library. This archive library is used to generate an archive entry for an MPS. An empty value disables archiving</td>
</tr>
</tbody>
</table>
Example:

[SECURITY]
SECURITY_ZIP_FILE security.zip
BASE_SECRET_LIST ;

[BOX_ARCHIVER]
SUPPORTED_MESSAGE_TYPES FIN, FACT; MX
DATA_COMPRESSION_METHOD GZIP

[MPS_CACHE]
CACHE_SIZE 120
CACHE_CONTENT YES
MAX_ITEM_SIZE 200
ARCHIVER box_archiver

[DB_INTERFACE]
DATA_SOURCE_NAME ; Leave blank
DATABASE_NAME ; to be defined
DATABASE_QUALIFIER ; to be defined
DATABASE_USERNAME ; to be defined
DATABASE_PASSWORD ;
DB_INTERFACE_LIBRARY mp_db2cli
MAX_CONNECTIONS 30 ; make sure the value is high enough.
Especially when the offline archiver tool is used, the value must be at least as high as the number of threads to be started by the archiver tool (start parameter (-t)).
8 Manual Message Entry for Tests

8.1.1.1 Pacs.008.001.02

The message Pacs.008.001.02 is used to transport the Payment instruction from the Originator Bank to the Beneficiary Bank, directly or through intermediaries. The message caters for bulk and single payment instructions.

For manual testing, a Pacs.008.001.02 message is written to a backend queue, which is read by BOX.

The general structure of a test message is:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Document xmlns="urn:iso:std:iso:20022:tech:xsd:pacs.008.001.02"
xmllns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:iso:std:iso:20022:tech:xsd:pacs.008.001.02
../xmlschemas/EPC122-16_2017_v1.1_pacs.008.001.02.xsd">
<FIToFICstmrCdtTrf>
<GrpHdr>
(MsgId>${individually generated by Originator}</MsgId>
<CreDtTm>${individually generated, Time of message generation}</CreDtTm>
<NbOfTxs>1</NbOfTxs>
<TtlIntrBkSttlmAmt Ccy="EUR">${individually generated amount}</TtlIntrBkSttlmAmt>
</GrpHdr>
<CdtTrfTxInf>
<PmtId>
<InstrId>${individually generated and optional}</InstrId>
<EndToEndId>${individually generated by Originator, identifies the SCT Transaction}</EndToEndId>
<TxId>${individually generated by Originator, identifies the SCT Transaction}</TxId>
</PmtId>
<IntrBkSttlmDmtd Ccy="EUR">${individually generated, amount}</IntrBkSttlmDmtd>
<AccptncDtTm>${individually generated, reception time of the SCT Transaction, Originator}</AccptncDtTm>
<ChrgBr>SLEV</ChrgBr>
</GrpHdr>
<CdtrAgt><FinInstnId><BIC></BIC></FinInstnId></CdtrAgt>
</CdtTrfTxInf>
</FIToFICstmrCdtTrf>
</Document>
```
9 Workflow

9.1 Exemplary SWIFTnet Workflow of an Archive-Persistence Mode

Figure 14  Overview SWIFTnet Instant Payment Workflow (TIPS)
9.2 Message Exception Workflow

Figure 15  Overview Exception Message Workflow

9.3 Signs and Symbols

Figure 16  Workflow Signs and Symbols

9.4 Instruction Patterns

9.4.1 Outgoing

INSTP Xxx_SW0_002_Process_Incoming_Message
INSTP Xxx_SW0_002_SBI_01_RoutingIncomingMessage
INSTP Xxx_SW0_007_ReceivedTechnicalResponse
INSTP Xxx_SW0_007_TGI_01_ReceivedTechnicalResponse
INSTP Xxx_SW0_002_DLI_02_Deliver_IncomingMessage
INSTP Xxx_SW0_003_Exception_Messages

9.4.2 Incoming

INSTP Xxx_SWI_001_RouteIPToAG
INSTP Xxx_SWI_001_CPI_01_Enrich (see graph below)
INSTP Xxx_SWI_001_DLI_03_RouteIPToAG
Figure 17  Enrichment with Submission Profile, configured in Content Processing Modules-FIA Subm. Prof. Instance.

9.4.3 Analysis 1

Analysis 1 provides a C-like programming language, which can be used to analyse various data objects, which are part of a Message Processing Sequence in BOX, such as MPS General Attributes, Report data and Content Version.

Analysis 1 is used during the ‘Outgoing’ Instant Payments workflow to determine, whether the message reflect a technical response, or an Instant Payment message received from SWIFTnet to be routed to the Payment Application.

Within Analysis 1 decisions are made following the either/or and if/then pattern.

The following is taken from the Workflow:

```
INSTP XXX SWO 002 Process Incoming Message

CheckForTechnicalResponse
if ( 4 == msga( ADTYPE)){
    print(" Technical Response received ");
    setips INSTP XXX SWO 007 Received_Technical_Response;
}
RoutingIncomingMessage
mq_address = getreplacementtokenvalue
("TAFABA:$$R$TO_PAYAPP_01$$SR$SRV_QGR_NAME$");
addtosimpleaddrlist(mq_address);
return;
```

Figure 18  Excerpt Workflow containing Analysis 1
10 Monitoring BOX with SNMP Dashboard

10.1 Monitoring BOX in non-persistence mode

![Diagram](image)

**Figure 19** Overview Monitoring in Non-Persistence Mode

### 10.1.1 System Monitor Module

The System Monitor Module controls and monitors the domain. It starts all required components it finds in the configuration of the domain and continuously receives heartbeat signals from all active components. If a component fails, it is automatically restarted by the System Monitor. When a domain is shutdown the System Monitor sends a signal to all components so they can stop operations in a controlled way. It uses the stub module to start and stop components and receives alerts (error-, warning-, and information messages) from all modules.

These alerts can either be forwarded and translated into SNMP traps, which in turn are read by an SNMP monitoring backend (respective MIB provided by Intecope) or analysed by alert files read by the Alert Reader tool also provided by Intecope. It is also possible to analyze the respective syslog.

### 10.1.2 Monitor Command Tool mpo_mcmd

With the Monitor Command Tool (mpo_mcmd) you can set the administrative status of the BOX modules and perform status queries against the modules. The administrative status (AdminStatus)
refers to the desired status of a module, while the operational status (OperStatus) refers to the actual status of the module. For further details, please refer to the document box_admig_vXrXX.pdf.

Example
mpo_mcmd 1/I1 /MB3 /G

<table>
<thead>
<tr>
<th>Server LCG Name</th>
<th>AdminStatus</th>
<th>OperStatus</th>
<th>LastChange</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTSADESOX 000273</td>
<td>active</td>
<td>unknown</td>
<td>Fri May 29 12:22:19 2015</td>
</tr>
<tr>
<td>Export</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
<tr>
<td>Import</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
<tr>
<td>PTSADESS_IAFA 000373</td>
<td>disabled</td>
<td>unknown</td>
<td>Fri May 29 12:22:13 2015</td>
</tr>
<tr>
<td>Export</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
<tr>
<td>Import</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
<tr>
<td>FACTBA_1 0006B3</td>
<td>active</td>
<td>unknown</td>
<td>Fri May 29 12:22:20 2015</td>
</tr>
<tr>
<td>Export</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
<tr>
<td>Import</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
<tr>
<td>SEPASTATUS 0006B3</td>
<td>active</td>
<td>active</td>
<td>Fri May 29 12:22:20 2015</td>
</tr>
<tr>
<td>Export</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
<tr>
<td>Import</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
<tr>
<td>SIAT2S 000473</td>
<td>disabled</td>
<td>unknown</td>
<td>Fri May 29 12:22:13 2015</td>
</tr>
<tr>
<td>Export</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
<tr>
<td>Import</td>
<td>active</td>
<td>active</td>
<td></td>
</tr>
</tbody>
</table>

10.1.3 SNMP Integration with Zabbix

To enable a graphical display of the server's health, Zabbix is a tool, which can be used.

Zabbix, an open source monitoring solution created by Alexei Vladishev, is currently actively developed and supported by Zabbix SIA. It is written and distributed under the GPL General Public License version 2.

It monitors parameters of a network and the health and integrity of servers. It uses a flexible notification mechanism and allows the configuration of e-mail-based alerts for events. It provides reporting and data visualization features based on stored data and supports both polling and trapping.

Through the Zabbix web-based frontend, reports, statistics and configuration parameters are accessible.

Figure 20  BOX Server Health Monitoring Example
## 11 Appendix

### 11.1 Parameter

#### 11.1.1 Analysis1

The following aliases have been defined in Analysis1 to retrieve SWIFT instant payment related data:

Please note, that technical responses on input messages (SWIFTNet: Notify and TechnicalAck) use ApplicationDefinedType 4 (Technical Response) in GenericAttributeSet.

<table>
<thead>
<tr>
<th>CV origination report:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CV_OR_PROTREP_IPRTSW_MSG_REF</td>
<td>Message reference, alias for TEXT 511, 1</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_ADDITIONAL_INFO</td>
<td>Additional info, alias for TEXT 511, 2</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_REQUESTOR_DN</td>
<td>Requestor DN, alias for TEXT 255, 1</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_RESPONDER_DN</td>
<td>Responder DN, alias for TEXT 255, 2</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_SERVICE_NAME</td>
<td>Service name, alias for TEXT63, 1</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_MSG_TYPE</td>
<td>Message type, alias for TEXT 63, 2</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_MSG_NETWORK_REF</td>
<td>Message network reference, alias for TEXT 63, 5</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_CHANNEL_NAME</td>
<td>Channel name, alias for TEXT 63, 7</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_PROTOCOL_CODE</td>
<td>Protocol code, alias for TEXT 63, 9</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_DEVICE_CODE</td>
<td>Device code, alias for TEXT 63, 10</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_POSSIBLE_DUPLICATE</td>
<td>Possible duplicate, alias for NUM 4</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_SEND_TIME</td>
<td>Send time, alias for TIME 1</td>
</tr>
<tr>
<td>CV_OR_PROTREP_IPRTSW_RECEIVE_TIME</td>
<td>Receive time, alias for TIME 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDA report:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SDA_PROTREP_IPRTSW_MSG_REF</td>
<td>Message reference, alias for TEXT 511, 1</td>
</tr>
<tr>
<td>SDA_PROTREP_IPRTSW_ADDITIONAL_INFO</td>
<td>Additional info, alias for TEXT 511, 2</td>
</tr>
<tr>
<td>SDA_PROTREP_IPRTSW_PRIMITIVE_ERROR_TEXT</td>
<td>Primitive error text, alias for TEXT 511, 3</td>
</tr>
<tr>
<td>SDA_PROTREP_IPRTSW_REQUESTOR_DN</td>
<td>Requestor DN, alias for TEXT 255, 1</td>
</tr>
<tr>
<td>SDA_PROTREP_IPRTSW_RESPONDER_DN</td>
<td>Responder DN, alias for TEXT 255, 2</td>
</tr>
<tr>
<td>SDA_PROTREP_IPRTSW_SERVICE_NAME</td>
<td>Service name, alias for TEXT63, 1</td>
</tr>
<tr>
<td>SDA_PROTREP_IPRTSW_MSG_TYPE</td>
<td>Message type, alias for TEXT 63, 2</td>
</tr>
<tr>
<td>SDA_PROTREP_IPRTSW_MSG_NETWORK_REF</td>
<td>Message network reference, alias for TEXT 63, 5</td>
</tr>
<tr>
<td>SDA_PROTREP_IPRTSW_CHANNEL_NAME</td>
<td>Channel name, alias for TEXT 63, 7</td>
</tr>
<tr>
<td>SDA_PROTREP_IPRTSW_REPORT_SOURCE</td>
<td>Report source, alias for TEXT 63, 8</td>
</tr>
<tr>
<td>Protocol code, alias for TEXT 63, 9</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td></td>
</tr>
<tr>
<td>Device code, alias for TEXT 63, 10</td>
<td></td>
</tr>
<tr>
<td>Primitive return code, alias for TEXT 15, 1</td>
<td></td>
</tr>
<tr>
<td>Possible duplicate, alias for NUM 4</td>
<td></td>
</tr>
<tr>
<td>Send time, alias for TIME 1</td>
<td></td>
</tr>
<tr>
<td>Receive time, alias for TIME 2</td>
<td></td>
</tr>
</tbody>
</table>

**ISDREP report:**

<table>
<thead>
<tr>
<th>Message reference, alias for TEXT 511, 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional info, alias for TEXT 511, 2</td>
</tr>
<tr>
<td>Primitive error text, alias for TEXT 511, 3</td>
</tr>
<tr>
<td>Requestor DN, alias for TEXT 255, 1</td>
</tr>
<tr>
<td>Responder DN, alias for TEXT 255, 2</td>
</tr>
<tr>
<td>Service name, alias for TEXT 63, 1</td>
</tr>
<tr>
<td>Message type, alias for TEXT 63, 2</td>
</tr>
<tr>
<td>Message network reference, alias for TEXT 63, 5</td>
</tr>
<tr>
<td>Channel name, alias for TEXT 63, 7</td>
</tr>
<tr>
<td>Report source, alias for TEXT 63, 8</td>
</tr>
<tr>
<td>Protocol code, alias for TEXT 63, 9</td>
</tr>
<tr>
<td>Device code, alias for TEXT 63, 10</td>
</tr>
<tr>
<td>Primitive return code, alias for TEXT 15, 1</td>
</tr>
<tr>
<td>Possible duplicate, alias for NUM 4</td>
</tr>
<tr>
<td>Send time, alias for TIME 1</td>
</tr>
<tr>
<td>Receive time, alias for TIME 2</td>
</tr>
</tbody>
</table>

**UPM and address book:**

<table>
<thead>
<tr>
<th>Responder DN, alias for TEXT 255, 2</th>
</tr>
</thead>
</table>

**MQ BACKOUT COUNT**

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQ</td>
<td>CV_OR_PROTREP_MQ_BACKOUT_COUNT</td>
</tr>
<tr>
<td>IPRT_EB</td>
<td>CV_OR_PROTREP_IPRTEB_MQ_BACKOUT_COUNT</td>
</tr>
</tbody>
</table>