

**STPG**

# **File Adapter**

## Specifications

---

***STPG ENVIRONMENT***

---

Revision 3; Software version: V1.0.53

December 2015

**CMA Small Systems AB**

**2015**

# ***STPG ENVIRONMENT***

## **File Adapter**

### ***Specifications***

Revision 3

The SOFTWARE is owned by CMA Small Systems AB or its suppliers and is protected by copyright laws, international treaty provisions, and all other applicable national laws. Therefore, you must treat the SOFTWARE like any other copyrighted material (e.g. a book) except that if the software is not copy protected you may either (a) make one copy of the SOFTWARE solely for the backup or archival purposes, or (b) transfer the SOFTWARE to a single hard disk provided you keep the original solely for backup or archival purposes. You may not copy the Product manual(s) or written materials accompanying the SOFTWARE.

Copyright © by Highex AB. All right reserved worldwide.

CMA Small Systems AB is the exclusive distributor of STPG.

## Version History

Version	Date	Author	Notes
2	23.11.2015	CMA	Review. 3.2.2 STPG Connectivity Configuration: description of parameter <b>reConnectionTimeout, sender, reciever</b> 5.5 Digital signing for acknowledgments 6.3 Installation procedure. Directory structure.
3	23.12.2015	CMA	Review. WSDL and examples correction. 8.3 System requirements Digital signing and verification examples

# Contents

<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>2. DATA STRUCTURES.....</b>	<b>3</b>
2.1 GENERAL DATA FORMAT DESCRIPTION.....	3
2.2 BARE MT FORMAT.....	3
2.2.1 <i>Message header to be sent to the system</i> .....	3
2.2.2 <i>Message footer to be sent to the system</i> .....	5
2.2.3 <i>Message header received from the system</i> .....	6
2.2.4 <i>Message footer received from the system</i> .....	8
2.3 XML BASED FORMATS.....	9
2.3.1 <i>Fields for messages to be sent to the system</i> .....	11
2.3.2 <i>Fields for messages received from the system</i> .....	12
2.3.3 <i>Positive acknowledgements (ACK) Fields</i> .....	14
2.3.4 <i>Negative acknowledgements (NAK) Fields</i> .....	14
2.4 CORRESPONDENCE BETWEEN BARE SWIFT AND XML MESSAGE HEADER AND FOOTER FIELDS.....	15
2.4.1 <i>Fields for messages to be sent to the system</i> .....	15
2.4.2 <i>Fields for messages received from the system</i> .....	17
<b>3. STPG FILE ADAPTER CLIENT DESCRIPTION.....</b>	<b>19</b>
3.1 GENERAL DESCRIPTION.....	19
3.2 COMMON CONFIGURATION PARAMETERS.....	19
3.2.1 <i>File System Configuration</i> .....	20
3.2.2 <i>STPG Connectivity Configuration</i> .....	21
3.2.3 <i>Cryptography settings</i> .....	21
3.2.3.1 <i>Private key</i> .....	21
3.2.3.2 <i>LDAP settings</i> .....	22
3.2.3.3 <i>Examples of crypt.conf</i> .....	22
3.3 RUNNING FILE ADAPTER.....	22
3.3.1 <i>Logging subsystem</i> .....	23
3.3.2 <i>Troubleshooting</i> .....	23
<b>4. SECURITY.....</b>	<b>24</b>
<b>5. DIGITAL SIGNATURES – DEVELOPER’S GUIDE.....</b>	<b>25</b>
5.1 GENERAL REQUIREMENTS AND RECOMMENDATIONS.....	25
5.1.1 <i>Private key with the corresponding certificate</i> .....	26
5.1.2 <i>Public certificates</i> .....	26

5.1.3	Digital signatures of block4 .....	26
5.1.4	Digital signatures of MX (and other XML) messages .....	26
5.1.5	Checks required during verification of digital signatures.....	26
5.2	DIGITAL SIGNING OF MT MESSAGE (BLOCK4) .....	27
5.3	VERIFYING DIGITAL SIGNATURE OF MT MESSAGE (BLOCK4) .....	29
5.4	DIGITAL SIGNING AND VERIFYING SIGNATURES OF MX OR OTHER XML.....	31
5.5	DIGITAL SIGNING FOR ACKNOWLEDGMENTS .....	32
<b>6.</b>	<b>INSTALLATION NOTES.....</b>	<b>34</b>
6.1	SYSTEM REQUIREMENTS .....	34
6.2	JAVA REQUIREMENTS .....	35
6.3	INSTALLATION PROCEDURE .....	35

---

## 1. Introduction

---

**STPG environment** is a complete set of software modules and components providing several facilities for integration with CMA solutions: Payment Gateway, DEPO/X, RTS/X and others (hereafter — the Core System). It gives opportunity to choose the most convenient protocol and desired architecture depending on customer needs for establishing client to system interaction on Straight-through processing (STP) principles.

**STPG** - Straight-through processing Gateway is the main component of the integration solution. On the one hand it organizes interchange with the Core System, on the other hand it serves connections, subscriptions and exchange session with other systems and external users. Usually it is installed in the central area playing the role of the front server in the Core System architecture.

**GWClient** – is the Client Communication Component. This component provides customers with an easy-to-use programming interface (API) for online message exchange with STPG. Communication between GWClient (i.e. client application that uses GWClient) and STPG is based on HTTP/HTTPS as network transport.

The GWCLIENT provides simplicity, ease of use and robustness. GWClient is implemented and distributed as a set of Java libraries (.jar files). Authentication, sending and receiving of messages by Participant's application are implemented with calls to methods (functions) provided by GWClient API. This document lists the methods that should be used to connect a user application to STPG. So, GWClient Java API allows Participants to establish interaction between STPG and participant's software (such as core banking system) via private network (i.e. local area network, or VPN, or similar network with TCP connectivity).

**STP Adapters.** CMA supplies integration solution with the several adapters that can be installed in a client environment and support the following interfaces:

- SOAP based web services
- File exchange
- Message queues (MQ)

All adapters use the GWClient component to organize connection and exchange session with STPG from one side and their particular interfaces to the client side. For every adapter certain settings can be enabled to automatically digitally sign content of outgoing messages and verify digital signatures of messages incoming from the Core System.

**STPG Web Services** – web services exposed on the STPG server. This option could be used by clients without installing any CMA software in their infrastructure. For security purposes HTTP should be protected with TLS/SSL encryption – HTTPS. It is also recommended to use client authentication with certificate to ensure that STPG accepts web service requests only from a system (or systems) that have a certificate issued by designated Certification Authority. Digital signing of content of outgoing messages and verification of digital signatures of incoming messages have to be implemented within client application (such as core banking system).

To work with the STPG, one must be familiar with

- SWIFT message formats.
- XML data formats.

This document describes STP Adapter with file exchange interface.

*Exchange of message files* with the STPG relies on a special program installed in the participant's computer. Two directories must be dedicated to file exchange: one to send messages to the STPG, and the other to read any messages received from the STPG. The names of such directories are specified in the configuration file when installing the STPG interaction software. A set of optional directories may be involved into interaction process: for storing acknowledgments and archiving processed files.

Each message to be sent to the STPG is packaged as a separate file. The program sends out all the files that the participant places in the outgoing directory. Any messages received from the STPG are placed into the incoming directory.

---

## 2. Data Structures

---

### 2.1 General Data Format Description

This software is intended to establish interaction between the STPG and any participants connected to it via the private network. To provide compatibility with terminology used to describe message formats '*participant*' hereinafter will be mentioned as '*user*'.

Three main message formats are supported. One of them is plain text-based format representing MT SWIFT format. Two others are XML based:

- XML "envelope" with MT message inside;
- XML envelope with MX (ISO20022) message inside.

These formats are described in chapters below.

### 2.2 Bare MT format

For the purpose of compatibility with SWIFT Alliance terminal STPG adapter supports bare MT message format for input and output messages. Short description of bare MT format is provided below.

#### 2.2.1 Message header to be sent to the system

Each message prepared for sending to the system includes a header that consists of blocks 1, 2, and 3 and should contain at least:

- BIC of the sender (8-character BIC of the sender's bank, terminal code, for example, A or B, three ending BIC characters if any or «XXX»);
- System BIC of the recipient (8-character BIC of the system, terminal code A or B, three ending BIC characters if any or «XXX»);
- Message type;
- Message reference (optional);
- Priority;
- Message text

The listed information, except for the messages text (Block 4) constitutes the message header.

When using the SWIFT-like format header information is specified in blocks 1, 2 and 3. Each block begins with a string sequence «{n:» (n-block number) and ends with character "}".

**Block 1:**

No.	Field name	Field format	Comments	Example
1.	Block opening (constant)	1!x	{ - should not be changed	{
2.	Block number	1!n1!x	Block number identifier and block divider (character “:”)	1:
3.	System constant	1!a2!n	F01 - should not be changed	F01
4.	Sender	12!c	8-character BIC of the sender’s bank, terminal code, for example, A or B, three terminating BIC characters if any or «XXX»	AZEGAZ22AXXX
5.	Number of communication session and sequence	4!n6!n	This field is automatically corrected by means of the software at the time of message sending. By default the user must put the value in this field ("0001000001")	0001000001
6.	Block closing (constant)	1!x	} - should not be changed	}

All fields are mandatory.

**Example**

**Block 1:** {1:F01AZEGAZ22AXXX0001000001}

**Block 2:**

No.	Field name	Field format	Comments	Example/comment
1.	Block opening (constant)	1!x	{ - should not be changed	{
2.	Block number	1!n1!x	Block number identifier and block divider (character “:”)	2:
3.	"Incoming/ outgoing" flag	1!a	In messages sent to the system should be used “I”.	I
4.	Message type	3!n	Message type is indicated	103
5.	Recipient	12!c	System BIC is indicated	NABZAZ2CXRTS
6.	Constant	1!a	Always N is indicated	N
7.	Block closing (constant)	1!x	}-should not be changed	}

All fields are mandatory.

**Example**

**Block:** {2:I103NABZAZ2CXRTSN}

**Block 3:**

No.	Field name	Field format	Comments	Example/comment
1.	Block opening (constant)	1!x	{ - should not be changed	{
2.	Block number	1!n1!x	Block number identifier and block divider (character “:”)	3:
3.	Operational priority	{113:4!n}	Priority value	{113:0050}
4.	Recipient reference	{108:16x}	Unique message reference, assigned by the Sender	{108:11568}
5.	Block closing (constant)	1!x	} - should not be changed	}

All fields except «User reference» are mandatory.

Operational priority field of the system uses standard SWIFT. SWIFT refers to field as to "Priority Banking".

Participants can submit reports about payments for this field in the header of User block (block 3) of the message. In case it is present, the operational priority determines the priority in the sequence in which the system receives payment.

This field is also used in private message types in Block «Text» (Box 4).

Active priorities of the bank, which can be defined in messages, are in the range from 0010 to 0099. The user interface of the Central Bank may also use priorities from 0001 to 0008, where 0001 is the highest priority, and 0008 - the lowest.

**Example**

**Block 3:** {3:{113:0050}{108:11568}}

Example of message header in SWIFT-like format, prepared for sending to the system:

```
{1:F01AZEGAZ22AXXX0001000001}{2:1103NABZAZ2CXRTSN}{3:{113:0050}{108:11568}}
```

**2.2.2 Message footer to be sent to the system**

Each message prepared for sending to the system includes a footer that consists of block 5 with the following fields.

**Block 5:**

No.	Field name	Field format	Comments	Example/comment
1.	Block opening (constant)	1!x	{ - should not be changed	{

2.	Block number	1!n1!x	Block number identifier and block divider (character “:”)	5:
3.	Test BIC indication	{TNG:}	Indication of test messages (should be present only for T&T BICs)	{TNG:}
4.	Electronic signature of financial information (block 4)	{MAC:<signature>}	optional (subject of rules established in financial environment) electronic signature of the block 4 of the message encoded in the format base64	{MAC:456kjh4fsf09udfgdfg-09srd...d43f4msg==}
5.	Block closing (constant)	!x	} - should not be changed	}

In case of rules established in financial environment require that Participant has to digitally sign financial information of the message (e.g. block4 of MT message) than the signature has to be placed into the field with tag MAC of block5. This field will be transferred to the system and, further, to the receiver of the message (subject of Central system configuration).

Example of the fifth message block:

```
{5:{TNG:}{MAC:456kjh4fsf09udfgdfg-09srd...d43f4msg==}}
```

### 2.2.3 Message header received from the system

Each message prepared for sending to the system consists of locks 1, 2, 3 and 5. The content of these blocks are described below.

#### Block 1:

No.	Field name	Field format	Comments	Example/comment
1.	Block opening (constant)	!x	{ -should not be changed	{
2.	Block number	1!n1!x	Block number identifier and block divider (character “:”)	1:
3.	System constant	1!a2!n	F01	F01
4.	Recipient field	12!c	Receiver's BIC. Note that while working through the SWIFT network 9 <sup>th</sup> character contains actual LT code	AZEGAZ22XXXX

No.	Field name	Field format	Comments	Example/comment
5.	Number and sequence of the communication session	4!n6!n	Session numbers and sequences of messages are supported in the system automatically	0001000003
6.	Block closing (constant)	1!x	} – should not be changed	}

All fields are mandatory.

**Block 2:**

No.	Field name	Field format	Comments	Example/comment
1.	Block opening (constant)	1!x	{ - should not be changed	{
2.	Block number	1!n1!x	Block number identifier and block divider (character “:”)	2:
3.	"Incoming/ outgoing" flag	1!a	In messages received from the system should be used “O”.	O
4.	Message type	3!n	Message type sent to participant from the system	999
5.	Sending time	4!n	Sending time of messages from the system is indicated in the format DDMM	1622
6.	Sending date	6!n	Sending date of messages from the system is indicated in the format YYMMDD	051122
7.	Message sender	12!c	System BIC is indicated with terminal LT code	NABZAZ2CARTS
8.	Number and sequence of the session	4!n6!n	Session numbers and message sequences	0001000003
9.	Message receiving date	6!n	Message receiving date is indicated in the format YYMMDD. Work station of the system participant (installed at the bank side) will fix message receiving date in this field	051122
10.	Receiving time	4!n	Message receiving time from the system is indicated in the format HHMM.	1622

11.	Constant	1!a	Always N is indicated	N
12.	Block closing (constant)	1!x	} – should not be changed	}

All fields are mandatory.

### Block 3:

No.	Field name	Field format	Comments	Example/comment
1.	Block opening (constant)	1!x	{ - should not be changed	{
2.	Block number	1!n1!x	Block number identifier and block divider (character “:”)	3:
3.	Operational priority	{113:4!n}	Priority value	{113:0050}
4.	Recipient reference	{108:16x}	Unique message reference, assigned by the Sender	{108:11568}
5.	Block closing (constant)	1!x	} - should not be changed	}

All fields except « User reference” are mandatory.

Example of message header received from the system through VPN:

```
{1:F01AZEGAZ22XXXX0001000003}{2:O9991624051122NABZAZ2CARTS00010000030511221624N}{3:{113:0050}{108:11568}}
```

- Note that for the messages received over the SWIFT network block 1 will contain BIC with LT code (e.g. AZEGAZ22AXXX) instead of AZEGAZ22XXXX

## 2.2.4 Message footer received from the system

Each message prepared for sending to the system includes a footer that consists of block 5 with the following fields.

### Block 5:

No.	Field name	Field format	Comments	Comment
1.	Block opening (constant)	1!x	{ - should not be changed	{
2.	Block number	1!n1!x	Block number identifier and block divider (character “:”)	5:

3.	Test BIC indication	{TNG:}	Indication of test messages (will be present only for T&T BICs)	{TNG:}
4.	Electronic signature of financial information (block 4)	{MAC:<signature>}	optional (subject of rules established in financial environment) electronic signature of the block 4 of the message encoded in the format base64	{MAC: 456kjh4fsf09udfgdfg- 09srd...d43f4msg==}
5.	Verification sign of a message by adapter	{PAC:1!n}	optional sign of verification of the electronic signature of the whole message	{PAC:1} means the signature is verified successfully {PAC:0} means signature verification error
6.	Possible duplication tag	{PDE:1n}	Optional indication of possible duplication of the message (as marked by Central system)	{PDE:1} means that a message is possibly duplicated
7.	Block closing (constant)	!x	} - should not be changed	}

In case of rules established in financial environment require that Participant has to digitally sign financial information of the message (e.g. block4 of MT message) than the signature has to be placed into the field with tag MAC of block5. This field will be transferred to the system and, further, to the receiver of the message (subject of Central system configuration). Messages received from the system may contain digital signature of block4 made by sender Participant (subject of system configuration) in the field tag MAC of block 5. In this case STP File adapter saves this field into the file unchanged. Participant software may check this signature for compliance with financial information in block 4.

Example of the fifth message block:

```
{5:{TNG:}{MAC:456kjh4fsf09udfgdfg-09srd...d43f4msg==}{PAC:1}}
```

## 2.3 XML based formats

For MT message the set and structure of service fields in such messages match the set and structure of the fields in S.W.I.F.T. MT message, while all the fields of Block 4 are copied unchanged to the [block4](#) field of a respective XML message.

For MX message the entire structure of XML document is preserved and body of message copied into [block4](#) field of a respective XML message.

All messages (both outgoing and incoming) must comply with the XML Data Definition Diagram shown below:

```
<?xml version="1.0" ?>
<!ELEMENT SWIFT_msg_fields (msg_format, msg_sub_format, msg_sender?,
msg_receiver?, msg_type, msg_priority, msg_del_notif_rq,
msg_user_priority, msg_user_reference?, msg_copy_srv_id?,
msg_fin_validation?, msg_pde?, (msg_session, msg_sequence,
msg_net_input_time, msg_net_output_date, msg_net_mir,
msg_copy_srv_info?, msg_mac_result?, msg_pac_result?, msg_pdm?)?,
format, ref_msg_user_reference?, block4)>
<!ELEMENT msg_format (#PCDATA)>
<!ELEMENT msg_sub_format (#PCDATA)>
<!ELEMENT msg_sender (#PCDATA)>
<!ELEMENT msg_receiver (#PCDATA)>
<!ELEMENT msg_type (#PCDATA)>
<!ELEMENT msg_priority (#PCDATA)>
<!ELEMENT msg_del_notif_rq (#PCDATA)>
<!ELEMENT msg_user_priority (#PCDATA)>
<!ELEMENT msg_user_reference (#PCDATA)>
<!ELEMENT msg_copy_srv_id (#PCDATA)>
<!ELEMENT msg_fin_validation (#PCDATA)>
<!ELEMENT msg_pde (#PCDATA)>
<!ELEMENT msg_session (#PCDATA)>
<!ELEMENT msg_sequence (#PCDATA)>
<!ELEMENT msg_net_input_time (#PCDATA)>
<!ELEMENT msg_net_output_date (#PCDATA)>
<!ELEMENT msg_net_mir (#PCDATA)>
<!ELEMENT msg_copy_srv_info (#PCDATA)>
<!ELEMENT msg_mac_result (#PCDATA)>
<!ELEMENT msg_pac_result (#PCDATA)>
<!ELEMENT msg_pdm (#PCDATA)>
<!ELEMENT msg_num_of_batches (#PCDATA)>
<!ELEMENT msg_amount (#PCDATA)>
<!ELEMENT format (#PCDATA)>
<!ELEMENT ref_msg_user_reference (#PCDATA)>
<!ELEMENT block4 (#PCDATA)>
```

Positive acknowledgements (ACK) received from STPG are rendered as a simple-structure XML documents, complying with the XML Data Definition Diagram shown below:

```
<?xml version="1.0" ?>
<!ELEMENT ack_nak (type, Data)>
<!ELEMENT Data (DateTime, MIR, REF?, Signature?)>
<!ELEMENT DateTime (#PCDATA)>
<!ELEMENT MIR (#PCDATA)>
<!ELEMENT REF (#PCDATA)>
<!ELEMENT Signature (#PCDATA)>
```

Negative acknowledgements (NAK) received from STPG are rendered as a simple-structure XML documents, complying with the XML Data Definition Diagram shown below:

```
<?xml version="1.0" ?>
<!ELEMENT ack_nak (type, Data)>
```

```
<!ELEMENT Data (DateTime, MIR, REF?, Signature?, Code, Description, Info)>
<!ELEMENT DateTime (#PCDATA)>
<!ELEMENT MIR (#PCDATA)>
<!ELEMENT REF (#PCDATA)>
<!ELEMENT Signature (#PCDATA)>
<!ELEMENT Code (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Info (#PCDATA)>
```

### 2.3.1 Fields for messages to be sent to the system

This Section describes the rules for filling message fields to be sent to the STPG:

Element **'msg\_format'** – The message format (must always be 'S' for messages handled by the STPG interface)

Element **'msg\_sub\_format'** – char 'I' for messages to be sent to the STPG.

Element **'msg\_sender'** - the 12-character BIC address of the message sender.

Element **'msg\_receiver'** - the 12-character BIC address of the message recipient.

Element **'msg\_type'** - the message type. It is the 3-character message type as specified in the Application Header Block of S.W.I.F.T. message for MT messages or entire message type for MX messages.

Element **'msg\_priority'** – 1-character transport priority level (should always be 'N' for messages handled by the STPG interface).

Element **'msg\_del\_notif\_rq'** – 1-character message delivery notification request (should always be 'N' for messages handled by the STPG interface).

Element **'msg\_user\_priority'** – 4-character indication of any priority the sender assigns for processing by the recipient. It is the banking priority field (field 113) in the User Header Block of S.W.I.F.T. message. The STPG uses this field to specify the payer's priority status.

Element **'msg\_user\_reference'** – optional 16-character user reference. It is the MUR (Message User Reference) field (field 108) in the User Header Block.

Element **'msg\_copy\_srv\_id'** – optional 3-character identification of network service.

Element **'msg\_fin\_validation'** – optional 8-character network validation services requested for the message.

Element **'msg\_pde'** – optional 1-character 'possible duplication emission' flag. User should set this element to 'Y' when sending the message second time

Element **'msg\_session'** – 4-character session number.

Element **'msg\_sequence'** – 6-character output sequence number.

Element **'msg\_net\_input\_time'** – 4-character remote input time in HHMM format.

Element **'msg\_net\_output\_date'** – 10-character local output date and time in YYMMDDHHMM format.

Element **'msg\_net\_mir'** – 28-character message input reference.

Element **'msg\_copy\_srv\_info'** – optional 32-character FIN Copy service information. Should always be empty for input messages.

Element **'msg\_mac\_result'** – participant's user digital authentication signature of the block4 element.

Element **'msg\_pac\_result'** – participant digital authentication signature of the entire message. It should always be empty for input messages.

Element **'msg\_pdm'** – 1-character 'possible duplication message' flag. Should always be empty for input messages.

Element **'format'** – contains the main format: MT or MX (could be extended in future, e.g.: XML).

Element **'ref\_msg\_user\_reference'** – this field is optional, in received messages it could be filled with 16-character reference of sent message (**msg\_user\_reference**).

Example of block4 of MT message:

```
20:TRNCRLF:79:Free text
```

Where

*CR* = carriage return

*LF* = line feed

Shown below is an example of a participant's message to be sent to the STPG:

```
<?xml version="1.0"?>
<SWIFT_msg_fields>
  <msg_format>S</msg_format>
  <msg_sub_format>I</msg_sub_format>
  <msg_sender>SNDRCC2AAXXX</msg_sender>
  <msg_receiver>RCVRCC2AAXXX</msg_receiver>
  <msg_type>100</msg_type>
  <msg_priority>N</msg_priority>
  <msg_del_notif_rq>N</msg_del_notif_rq>
  <msg_user_priority>0010</msg_user_priority>
  <msg_mac_result>4CDF324373308B2F8035DE18364</msg_mac_result>
  <format>MT</format>
  <block4>20:TRANSREF
32A:990802CCD10000,00
50:A PAYER INC
59:A PAYEE INC
70:PAYER TO PAYEE INFO</block4>
</SWIFT_msg_fields>
```

### 2.3.2 Fields for messages received from the system

This Section describes the rules for filling message fields received from the STPG:

Element **'msg\_format'** – The format of the message. Will always be 'S' for messages received from the STPG.

Element **'msg\_sub\_format'** – char 'O' for messages received from the STPG.

Element **'msg\_sender'** - the 12-character BIC address of the message sender.

Element **'msg\_receiver'** - the 12-character BIC address of the message recipient.

Element **'msg\_type'** - the message type. It is the 3-character message type as specified in the Application Header Block of S.W.I.F.T. message for MT messages or entire message type for MX messages.

Element **'msg\_priority'** – 1-character transport priority level.

Element **'msg\_del\_notif\_rq'** – 1-character message delivery notification request .

Element **'msg\_user\_priority'** – 4-character indication of any priority the sender assigns for processing by the recipient. It is the banking-priority field (field 113) in the User Header Block of S.W.I.F.T. message. The STPG uses this field to specify the payee's priority status.

Element **'msg\_user\_reference'** – optional 16-character user reference. It is the MUR (Message User Reference) field (field 108) in the User Header Block.

Element **'msg\_copy\_srv\_id'** – optional 3-character identification of network service.

Element **'msg\_fin\_validation'** – optional 8-character network validation services requested for the message.

Element **'msg\_pde'** – optional 1-character 'possible duplication emission' flag.

Element **'msg\_session'** – 4-character session number.

Element **'msg\_sequence'** – 6-character output sequence number.

Element **'msg\_net\_input\_time'** – 4-character remote input time in the HHMM format.

Element **'msg\_net\_output\_date'** – 10-character local output date and time in YYMMDDHHMM format.

Element **'msg\_net\_mir'** – 28-character message input reference.

Element **'msg\_copy\_srv\_info'** – optional 32-character FIN Copy service information.

Element **'msg\_mac\_result'** – participant's user digital authentication signature of the block4 element.

Element **'msg\_pac\_result'** – result of STPG's digital authentication signature check of the entire message.

Element **'msg\_pdm'** – 1-character 'possible duplication message' flag. s mandatory for MT150 message)Element **'block4'** – contains the contents of block 4 of the S.W.I.F.T. message for MT messages or entire message body for MX messages.

Element **'format'** – contains the main format: MT or MX (could be extended in future, e.g.: XML).

Element **'ref\_msg\_user\_reference'** – this field is optional, in received messages it could be filled with 16-character reference of sent message (**msg\_user\_reference**).

Example of block4 of MT message:

```
20:TRNCRLF:79:Free text
```

Where

*CR* = carriage return

*LF* = line feed

Shown below is an example of a message received from the STPG by a participant:

```
<?xml version="1.0"?>
```

```

<!--Example of input message-->
<SWIFT_msg_fields>
  <msg_format>S</msg_format>
  <msg_sub_format>O</msg_sub_format>
  <msg_sender>SNDRCC2AAXXX</msg_sender>
  <msg_receiver>RCVRCC2AAXXX</msg_receiver>
  <msg_type>100</msg_type>
  <msg_priority>N</msg_priority>
  <msg_del_notif_rq>N</msg_del_notif_rq>
  <msg_user_priority>10</msg_user_priority>
  <msg_user_reference/>
  <msg_copy_srv_id/>
  <msg_fin_validation/>
  <msg_pde>N</msg_pde>
  <msg_session>0123</msg_session>
  <msg_sequence>000456</msg_sequence>
  <msg_net_input_time>1200</msg_net_input_time>
  <msg_net_output_date>9908021428</msg_net_output_date>
  <msg_net_mir>990802SNDRCC2AAXXX0123000456</msg_net_mir>
  <msg_copy_srv_info/>
  <msg_mac_result>4CDF324373308B2F8035DE18364</msg_mac_result>
  <msg_pac_result>1</msg_pac_result>
  <msg_pdm>N</msg_pdm>
  <format>MT</format>
  <block4>20:TRANSREF
32A:990802CCD10000,00
50:A PAYER INC
59:A PAYEE INC
70:PAYER TO PAYEE INFO</block4>
</SWIFT_msg_fields>

```

### 2.3.3 Positive acknowledgements (ACK) Fields

This Section describes the fields of positive acknowledgements received from STPG:

Element '**DateTime**' – 6-character date and time of acknowledgements in YYMMDDformat

Element '**MIR**' – 28-character message input reference of affected message.

Element '**REF**' – 16-character message user reference of affected message.

Element '**Signature**' – digital authentication signature for acknowledgement.

Shown below is an example of positive acknowledgements received from STPG:

```

<?xml version="1.0"?>
<ack_nak>
  <type>ACK</type>
  <Data>
    <DateTime>150115</DateTime>
    <MIR>990802SENDER22XXXX0123000456</MIR>
    <REF>FT1331669663</REF>
    <Signature>SIGNATURE</Signature>
  </Data>
</ack_nak>

```

### 2.3.4 Negative acknowledgements (NAK) Fields

This Section describes the fields of positive acknowledgements received from STPG:

Element **'DateTime'** – 6-character date and time of acknowledgements in YYMMDDformat

Element **'MIR'** – 28-character message input reference of affected message.

Element **'REF'** – 16-character message user reference of affected message.

Element **'Code'** – error code describing the cause of message rejection.

Element **'Description'** – user friendly description of the cause of message rejection.

Element **'Info'** – additional information related to the cause of message rejection.

Shown below is an example of negative acknowledgements received from STPG:

```
<?xml version="1.0"?>
<ack_nak>
  <type>NAK</type>
  <Data>
    <DateTime>1501151112</DateTime>
    <MIR>990802SENDER22XXXX0123000456</MIR>
    <REF>FT1331669663</REF>
    <Signature>SIGNATURE</Signature>
    <Code>EL26</Code>
    <Description>Invalid value of message user priority</Description>
    <Info>AA</Info>
  </Data>
</ack_nak>
```

## 2.4 Correspondence between bare SWIFT and XML message header and footer fields

### 2.4.1 Fields for messages to be sent to the system

#### Block 1:

No.	SWIFT field	XML	Comment
1.	Block opening (constant)		
2.	Block number		
3.	System constant		
4.	Sender	msg_sender	8-character BIC of the sender's bank, terminal code, for example, A or B, three terminating BIC characters if any or «XXX»

No.	SWIFT field	XML	Comment
5.	Number of communication session and sequence	msg_session and msg_sequence	
6.	Block closing (constant)		

**Block 2:**

No.	SWIFT field	XML	Comment
1.	Block opening (constant)		
2.	Block number		
3.	"Incoming/ outgoing" flag	msg_sub_format	
4.	Message type	msg_type	
5.	Recipient	msg_receiver	System BIC
6.	Constant		
7.	Block closing (constant)		

**Block 3:**

No.	SWIFT field	XML	Comment
1.	Block opening (constant)		
2.	Block number		
3.	Operational priority	msg_user_priority	
4.	Recipient reference	msg_user_reference	
5.	Block closing (constant)		

**Block 5:**

No.	SWIFT field	XML	Comment
1.	Block opening (constant)		
2.	Block number		

3.	TNG field		Present for T&T BICs
4.	MAC field	msg_mac_result	
5.	Block closing (constant)		

## 2.4.2 Fields for messages received from the system

### Block 1:

No.	SWIFT field	XML	Comment
1.	Block opening (constant)		
2.	Block number		
3.	System constant		
4.	Recipient field	msg_receiver	while working through the SWIFT network 9 <sup>th</sup> character contains actual LT code
5.	Number and sequence of the communication session	msg_session and msg_sequence	
6.	Block closing (constant)		

### Block 2:

No.	SWIFT field	XML	Comment
1.	Block opening (constant)		
2.	Block number		
3.	"Incoming/ outgoing" flag	msg_sub_format	
4.	Message type	msg_type	
5.	Sending time	msg_net_input_time	

6.	Sending date	msg_net_mir	
7.	Message sender	msg_sender	
8.	Number and sequence of the session	msg_session and msg_sequence	
9.	Message receiving date	msg_net_output_date	
10.	Receiving time	msg_net_output_date	
11.	Constant		
12.	Block closing (constant)		

**Block 3:**

No.	SWIFT field	XML	Comment
1.	Block opening (constant)		
2.	Block number		
3.	Operational priority	msg_user_priority	
4.	Recipient reference	msg_user_reference	
5.	Block closing (constant)		

**Block 5:**

No.	SWIFT field	XML	Comment
1.	Block opening (constant)		
2.	Block number		
3.	TNG field		Present for T&T BICs
4.	MAC field	msg_mac_result	
5.	PAC field	msg_pac_result	'1' – the signature is verified successfully, '0' – signature verification error
6.	PDE field	msg_pde	
7.	Block closing (constant)		

---

## 3. STPG File Adapter Client Description

---

### 3.1 General Description

The STPG File Adapter Client (hereafter referred to as the File Adapter) is a software product intended to provide connectivity to the STPG by means of file exchange. The File Adapter is based on an STP Adapter technology. The File Adapter implements two interfaces. On the one hand it establishes and maintains a network client connection to the STPG, and on the other hand it monitors the given directory for incoming files to be sent to the STPG. Every file placed into the incoming directory is processed by the File Adapter and sent to the STPG. Every message received by the File Adapter from the STPG is processed and placed into the outgoing directory.

### 3.2 Common Configuration Parameters

The File Adapter configuration parameters are located in the configuration file called 'config\_gw\_file.properties'. It is located in props directory in File Adapter installation folder. The name of this file is related with Adapter startup configuration file which is used as one of argument to command line to start File Adapter. For convenient the command line placed in command file with corresponding name - start\_stpa\_gw\_file.cmd, start\_stpa\_gw\_file\_pp.cmd (the same for relative scripts(.sh) for \*NIX operating systems). Command file is located in File Adapter installation folder. This command file normally uses to start File Adapter

```
>start_stpa_gw_file.cmd
>start_stpa_gw_file_pp.cmd

>start_stpa_gw_file.sh
>start_stpa_gw_file_pp.sh
```

The File Adapter configuration file is valid simple properties file. Typical configuration parameter located in this file presented as equality like name = value. For example:

```
processedDir = processed
```

The configuration parameters that were not mentioned in this documentation affect internal STP Adapter functionality and should not be changed by the user during normal operation. The detailed description of the configuration parameters is given below in this chapter.

### 3.2.1 File System Configuration

Affected parameters specify the file interface of the File Adapter for messages to be sent to STPG and for messages to be received from STPG. There are six parameters to be set according to the required directory structure (in bold presented short name of parameter for future references):

- **inputDir** - sets the path to the directory in the file system wherefrom the Adapter will read out data files.
- **processedDir** - sets the path to the directory in the file system where successfully delivered files will be moved. Under “successfully delivery” means that the message successfully delivered to STPG and File Adapter got positive (ACK) acknowledgement from the System. This is optional parameter – if this parameter missing or empty in the configuration file than successfully delivered files will be deleted.
- **unprocessedDir** - sets the path to the directory in the file system where unsuccessfully delivered files will be moved. Under “unsuccessfully delivery” means that the message delivered to STPG but File Adapter got negative (NAK) acknowledgement from the System. This is optional parameter – if this parameter missing or empty in the configuration file than unsuccessfully delivered files will be deleted.
- **errorDir** - sets the path to the directory in the file system where unsuccessfully processed files will be moved. Under “unsuccessfully processed” means that the message was not delivered to STPG due to some problems with message preparation or communication problems. This is optional parameter – if this parameter missing or empty in the configuration file than unsuccessfully processed files will be deleted.
- **rejectedDir** - sets the path to the directory in the file system where negative acknowledgment (NAK) messages from STPG will be stored. This is optional parameter – if this parameter missing or empty in the configuration file than negative acknowledgment messages will not be stored.
- **acceptedDir** - sets the path to the directory in the file system where positive acknowledgment (ACK) messages from STPG will be stored. This is optional parameter – if this parameter missing or empty in the configuration file than positive acknowledgment messages will not be stored.
- **outputDir** - sets the path to the directory in the file system where messages received by adapter from STPG will be stored.
- **outputType** – specifies the format how the messages received by adapter from STPG will be stored in the corresponding directory. There are two options available:
  - XML – messages will be stored in XML format
  - MT – messages will be stored in bare MT format.

Below is an example of a typical file system configuration:

```
inputDir =input
processedDir =processed
unprocessedDir=unprocessed
acceptedDir =accepted
rejectedDir =rejected
errorDir =error
outputDir =output
```

### 3.2.2 STPG Connectivity Configuration

Affected parameters specify the STPG connectivity configuration:

- **user** - sets user identifier for connection to STPG.
- **password** - sets password for connection to STPG.
- **url** - sets URL identifier for connection to STPG.
- **reConnectionTimeout** – sets the timeout in seconds between reconnections.
- **connectRetries** – the number of reconnection attempts to be performed in case of unexpected disconnection from STPG until Adapter will stop with connection fail diagnostics. In case of successful reconnection, the counter of reconnection attempts will be reset to this parameter value. By default no reconnection will be performed. Special parameter values:
  - 0 – no reconnections will be performed (this is default)
  - -1 – reconnection attempts will be continued until successful reconnection or manual stop of Adapter.

Below is an example of a typical STPG connectivity configuration.

```
user =CITIIDJAXXX  
password =1  
url =https://192.168.19.106:8443/SSYSGw/gw  
connectRetries =10  
reConnectionTimeout=5
```

### 3.2.3 Cryptography settings

Cryptography settings, required for digital signatures in the File Adapter, are located in the configuration file named 'crypto.conf'. It is located in File Adapter's installation folder.

Settings in the file look like a set of pairs `name = value`. For example:

```
keystore=windows
```

#### 3.2.3.1 Private key

In order to create digital signatures, File Adapter needs to have a private key with corresponding X.509 certificate, issued by designated Certification Authority (referred to as "CA" in current document).

CMA Java Crypto API supports private key stored in any of the following keystore types:

- Java keystore file (.jks)
- Windows certificate store

Selection of keystore is configurable with the following setting in 'crypto.conf':

- **keystore** (either "file" or "windows") – specifies if the application uses a file or Windows certificate store for private key and its corresponding certificate.

In case of **keystore=file** the following additional settings are available:

- **keystoreType** (default: JKS) – type of keystore file
- **keystoreFile** – keystore file name (and path)
- **keystorePass** – password for the keystore file
- **keyAlias** – key alias (useful if there are many keys in the keystore); in case if key alias is not set, File Adapter iterates through all the aliases until it finds the first entry with a private key and corresponding public certificate.

In case of **keystore=windows**, current Windows user's personal certificate store is searched for certificates with private keys (expired certificates are skipped). If there are several certificates found, File Adapter shows GUI dialog to select a certificate. If there were no certificates found, File Adapter shows another GUI prompt for a user to insert token/smart card.

### 3.2.3.2 Examples of crypt.conf

An example for the case with private key stored in a Java keystore:

```
keystore=file
keystoreFile=mykey.jks
keystorePass=changeme
```

Another example is for the case with private key stored current Windows user's personal certificate store:

```
keystore=windows
```

## 3.3 Running File Adapter

After the File Adapter has been launched it starts establishing connection with the STPG. When the connection is established normal operations begin.

The File Adapter monitors the directory specified for the **inputDir** parameter. When a new file appears there the File Adapter takes that file, assuming it is in the XML format, and sends to the STPG Server.

If the file has any kind of errors (contains syntactic errors or whatever) that leads to its rejection by the File Adapter itself it is moved to the directory specified by the **errorDir** parameter. The cause of the problem will be clearly described in the special error file, putted in the same directory together with rejected file. The name of error file makes from original message file name plus extension ".err". Also cause of the problem will be fixed in the console and log files.

If the file has any kind of errors (contains syntactic errors or whatever) that leads to its rejection by the STPG it is moved to the directory specified by the **unprocessedDir** parameter. The cause of the problem will be clearly described in the negative acknowledgement file (NAK), putted in the directory specified by the **rejectedDir** parameter. The name of negative acknowledgement file makes from original message file name plus extension “.nak.xml”. Also cause of the problem will be fixed in the console and log files.

If the file was successfully accepted by STPG it is moved to the directory specified by the **processedDir** parameter. The positive acknowledgement file (ACK), putted in the directory specified by the **acceptedDir** parameter. The name of positive acknowledgement file makes from original message file name plus extension “.ack.xml”.

All data messages, sent to the File Adapter by the STPG are placed into the directory specified by **outputDir** parameter. Upon receiving the message from STPG and successful delivering it to output directory, positive acknowledgement (ACK) will be sent to STPG automatically.

Stopping the adapter can be done by pressing Ctrl-C

### 3.3.1 Logging subsystem

During startup and production run all major information regarding functioning of Adapter is outputted into system console. Additionally for protocol and eventually problems solving reasons file logging subsystem is implemented. This subsystem preconfigured upon Adapter delivery and creates following log files in log directory:

- STPAdapter\_INFO.log – log file generated on INFO level
- STPAdapter\_DEBUG.log – log file generated on DEBUG level
- STPAdapter\_ALL.log – all available log information
- console.log – console information written to the file
- exceptions.log – error log

All logging files placed into log directory and supplied by packing and renaming capabilities.

### 3.3.2 Troubleshooting

The symptom of successful STPAdapter startup is following line in the console and logs:  
adaptor.Adaptor (Adaptor.java:346)|waiting for runnables to stop

This line means that adapter successfully initialized and established connection to STPG - adapter is ready to send and receive messages. Adapter will be running until manual stop or some unrecoverable error occurred.

In case of some problems occurred, adapter stops and clear diagnostic messages places into console (and into log files if logging subsystem successfully initialized).

Examples of typical startup problems:

1. Some dependency for some component is missing in the classpath.

Diagnostics example:

Error creating bean with name 'GwConnection' defined in URL [file:cfg/gw\_config.xml]: Instantiation of bean failed

Decision: provide all necessary libraries and its dependencies in the bin directory or reinstall adapter eventually preserving configuration file (cfg directory).

2. Some property has invalid value.

Diagnostics example:

PgDirWriter : **inputDir** pg does not exist or is not a directory

Decision: provide correct value of property in the configuration file.

3. Some obligatory property is missing.

Diagnostics example:

PgDirWriter : **inputDir** has to be set

Decision: provide value of property in the configuration file.

4. Initialization procedure for some component is failed.

Diagnostics example:

PgGwWriter : Failed to open gw connection

Decision: is the example above opening connection to STPG system was failed. There can be a lot of different causes. For example wrong property **url** for connection or absence of available STPG server.

---

## 4. Security

---

Current chapter specifies security mechanisms employed to secure communication and protect integrity of message exchange for STP File Adapter and states security requirements to client application.

- 1) Encryption:

- a. All the data transferred between STP Adapter (on remote Participant site) and STPG (in Central Institution) is encrypted with TLS/SSL, because HTTPS is the protocol used for communication between STP Adapter and STPG.

- 2) Access control:

- a. Connection of STP Adapter to STPG is authenticated using login/password (assigned in the Core System in Central Institution) and with digital certificate (HTTPS client authentication is required).

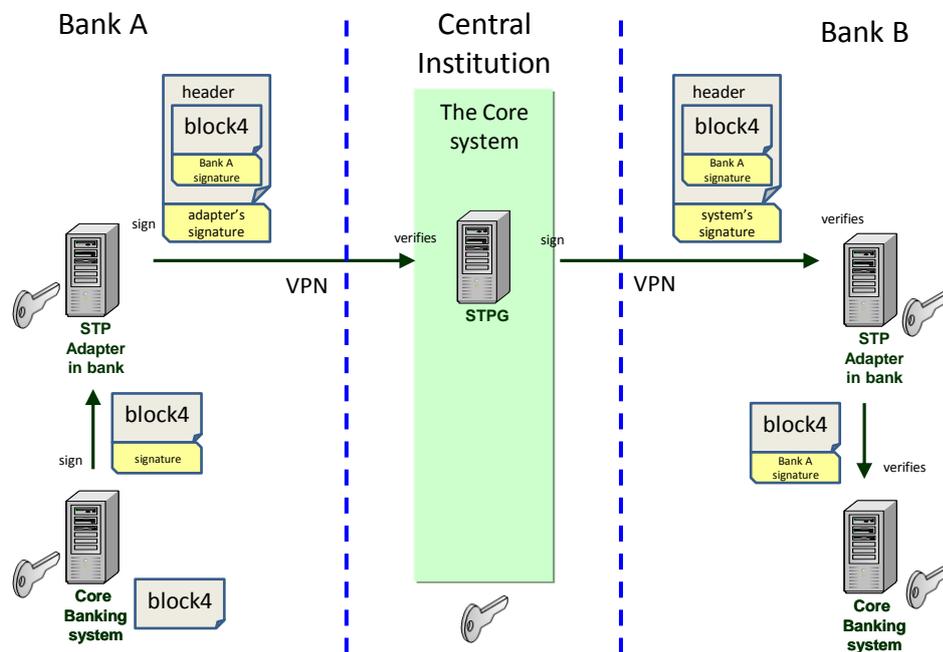
- 3) Message integrity, non-repudiation:

- a. Participant's client application digitally signs contents of each message sent to STP Adapter; STP Adapter optionally adds another signature and transmits the data to STPG. The Core System validates digital signature(s), stores them in central database (to allow verification in future, such as to prove non-repudiation in case of a dispute or for audit).

- b. The Core System digitally signs messages, client application receives content with signature and can validate the signature.
- c. In case if message content was prepared and digitally signed by another participant, and the content is transmitted through the Core System unmodified (e.g. copy of a payment message), the Core System also copies original sender Participant's digital signature. So, the Recipient Participant's application can verify Sender Participant's digital signature.

This is illustrated on the following diagram.

*Diagram 1. Digital signatures exchanged between two Participants (example with MT messages)*



## 5. Digital signatures – developer's guide

### 5.1 General requirements and recommendations

Participant application needs to create digital signatures of outgoing messages and verify digital signatures of incoming messages.

### 5.1.1 Private key with the corresponding certificate

In order to create digital signatures, the application needs to have a private key with corresponding X.509 certificate, issued by designated Certification Authority.

The private key can be stored in any of the following locations:

- Java keystore file (.jks),
- Windows certificate store (which is actually a file, just secured by Windows domain security) - Java applications may use MSCAPI provider to access this type of stores;
- Hardware device (such as cryptography token, or smart card, or even HSM). Usually manufacturers of such devices provide MSCAPI security provider and/or PKCS#11 library that allow standards-compliant applications to access these devices. Java allows access to devices with either of these interfaces.

### 5.1.2 Public certificates

In order to verify digital signatures of incoming messages, client applications need to have

- 1) certificate of the central system (to verify any signatures generated by the system);
- 2) certificates of all other participants (to verify signatures generated by other participants on their payments).

All the certificates are kept centrally in the system in LDAP directory service.

Participant applications should retrieve certificates from LDAP and use them to verify digital signatures.

### 5.1.3 Digital signatures of block4

Messages incoming from the system have block4 (string content of block4 field) digitally signed, and the signature is stored in field msgMacResult.

Similarly, for those messages that participant is going to send to the system, client application has to digitally sign block4 with detached signature, encode the signature using base64 and put it into field msgMacResult.

### 5.1.4 Digital signatures of MX (and other XML) messages

In case of using MX or other XML message types, messages incoming from the system have XML as string content of block4 field. It is digitally signed, and the signature is stored in field msgMacResult.

Similarly, for those messages MX or XML that participant is going to send to the system, client application has to digitally sign MX or XML with detached XML signature, and store it in the field msgMacResult.

### 5.1.5 Checks required during verification of digital signatures

As part of verifying digital signature, participant application has to check if the signer's certificate was revoked, or not. This is achieved through the use of Certificate Revocation List (CRL) that is also published by

designated Certification Authority to central LDAP or available through other URL(s) – specified in “CRL Distribution Point” (CDP) extension of signer’s certificate.

Client application should be able to download CRL either from LDAP or from any URLs specified in CDPs of signers’ certificates and should verify validity of signers’ certificates.

CRLs may be updated frequently (in certain environments as often as every hour), so client applications need to take that into account and should regularly check LDAP for fresh CRL (updated CRL has sequence number incremented as compared to older CRL).

## 5.2 Digital signing of MT message (block4)

Block4 of an MT message has to be digitally signed prior to sending to the system. In order to sign it, block4 string has to be converted to buffer (array) of bytes using the following algorithm:

- 1) Translate all line endings in block4 string from CR+LF (if any) to LF
- 2) Convert (encode) the string into array of bytes using UTF-16LE encoding (UTF16 Little Endian)

Once this is done, the array of bytes has to be digitally signed using private key and corresponding certificate. Signature format is **PKCS#7 Enveloped Data, detached signature**. Signature buffer should be finally base64-encoded into a string, prior to passing it to any API or sending to the system.

Detached signature means that the signature buffer produced is separate from the signed data buffer. Attached signature format SHOULD NOT be used as the MT message is transmitted and stored anyway, and duplicating it inside signature buffer is a waste of valuable disk space and network traffic .

The signature SHOULD include

- SignerInfo - Signer’s certificate issuer name and serial number
- Date/time stamp as signed attribute

The signature SHOULD NOT include Signer’s certificate. Every certificate used in the system is stored in central LDAP directory server and any software that verifies digital signatures retrieves certificates from LDAP. So, including signer certificate into every signature is a huge waste of valuable database space and also network traffic. SignerInfo (issuer name and serial number) is enough to verify the signature.

The signature SHOULD NOT include any other certificates or certificate revocation lists (CRL) for the same reason – that’s a waste of space and network traffic.

Example in Java language, with simplified (no) exception handling and based on open source code:

- 1) Bouncy Castle Crypto APIs (<https://www.bouncycastle.org/java.html> )
- 2) Base64 (<http://iharder.sourceforge.net/current/java/base64/>)

```
import java.security.PrivateKey;
import java.security.cert.X509Certificate;

import org.bouncycastle.cms.CMSProcessableByteArray;
import org.bouncycastle.cms.CMSSignedData;
import org.bouncycastle.cms.CMSSignedDataGenerator;
import org.bouncycastle.cms.CMSTypedData;
import org.bouncycastle.cms.jcajce.JcaSignerInfoGeneratorBuilder;
import org.bouncycastle.operator.ContentSigner;
```

```

import org.bouncycastle.operator.jcajce.JcaContentSignerBuilder;
import org.bouncycastle.operator.jcajce.JcaDigestCalculatorProviderBuilder;

public class CMSSignedDataSigner {

    final PrivateKey privateKey;
    final X509Certificate signerCert;
    final String providerName;
    final String algorithm;

    public CMSSignedDataSigner(PrivateKey privkey, X509Certificate cert, String provider, String
alg)
    {
        privateKey = privkey;
        signerCert = cert;
        providerName = provider;
        algorithm = alg;
    }

    private byte[] signSep(byte[] data) throws Exception
    {
        CMSTypedData typedData = new CMSPprocessableByteArray(data);
        CMSSignedDataGenerator gen = new CMSSignedDataGenerator();
        ContentSigner signer =
            new JcaContentSignerBuilder(algorithm)
                .setProvider(providerName)
                .build(privateKey);
        JcaSignerInfoGeneratorBuilder builder = new JcaSignerInfoGeneratorBuilder(
            new JcaDigestCalculatorProviderBuilder()
                .setProvider("BC")
                .build() );
        gen.addSignerInfoGenerator(builder.build(signer, signerCert));
        CMSSignedData signed = gen.generate(typedData, false);
        byte[] der = signed.getEncoded();
        return der;
    }

    public String signBlock4(String block4) throws Exception {

        final String stringToSign = block4.replaceAll("\r\n", "\n");
        byte[] dataToSign = stringToSign.getBytes("UTF-16LE");
        return Base64.encodeBytes(signSep(dataToSign));
    }
}

```

In the example above an instance of CMSSignedDataSigner has to be constructed with the following arguments:

- 1) Signer's private key from a keystore
- 2) Signer's X.509 certificate from the same keystore
- 3) Cryptography provider - should be
  - a. either "BC" in case of java keystore (JKS file),
  - b. or another value in case of other types of keystores, such as MS CAPI  
– should be extracted from KeyStore object "ks" as follows:

```
final String provider = ks.getProvider().getName();
```

- 4) Signature algorithm – usually "SHA256withRSA" if private key is in java keystore or similar software-based store, or "SHA1withRSA" in case if private key is on smartcard/token or similar device (reason: some popular smartcards/tokens still don't support signing with SHA256 hash, they only support SHA1);

### 5.3 Verifying digital signature of MT message (block4)

Block4 of an MT messages received from the central system are digitally signed by the system. In order to verify digital signature, block4 string has to be converted to buffer (array) of bytes using the same algorithm which is used while signing block4:

- 1) Translate all line endings in block4 string from CR+LF (if any) to LF
- 2) Convert (encode) the string into array of bytes using UTF-16LE encoding (UTF16 Little Endian)

Digital signature string is a base64 encoded buffer of PKCS#7 Enveloped Data, detached signature. The signature string has to be base64-decoded into a byte buffer in order to proceed with verification.

Once the steps above are done, the two byte arrays (signed data and digital signature) can be used to verify the signature.

Example in Java language, with simplified (no) exception handling and based on open source code:

- 1) Bouncy Castle Crypto APIs (<https://www.bouncycastle.org/java.html> )
- 2) Base64 (<http://iharder.sourceforge.net/current/java/base64/>)

```
import java.math.BigInteger;
import java.security.cert.X509Certificate;
import java.util.Date;
import java.util.Enumeration;
import java.util.Iterator;

import org.bouncycastle.asn1.ASN1UTCTime;
import org.bouncycastle.asn1.DERUTCTime;
import org.bouncycastle.asn1.cms.Attribute;
import org.bouncycastle.asn1.cms.AttributeTable;
import org.bouncycastle.asn1.cms.CMSAttributes;
import org.bouncycastle.asn1.x500.X500Name;
import org.bouncycastle.cms.CMSException;
import org.bouncycastle.cms.CMSProcessableByteArray;
import org.bouncycastle.cms.CMSSignedData;
import org.bouncycastle.cms.CMSTypedData;
import org.bouncycastle.cms.SignerInformation;
import org.bouncycastle.cms.SignerInformationStore;
import org.bouncycastle.cms.jcajce.JcaSimpleSignerInfoVerifierBuilder;

public class CMSSignedDataVerifier {

    private boolean checkSep(byte[] data, byte[] sign) throws Exception {

        CMSTypedData typedData = new CMSProcessableByteArray(data);
        CMSSignedData s = new CMSSignedData(typedData, sign);
        SignerInformationStore signers = s.getSignerInfos();
        @SuppressWarnings("unchecked")
        Iterator<SignerInformation> iter = signers.getSigners().iterator();
        while (iter.hasNext()) {
            SignerInformation signer = iter.next();
            X500Name issuer = signer.getSID().getIssuer();
            BigInteger serial = signer.getSID().getSerialNumber();
            System.out.println("Serial: " + serial + ", issuer: " + issuer);

            // retrieve signer's certificate from certificate database (LDAP)
            final X509Certificate signerCert =
                selectCertByIssuerSerial(issuer, serial);

            // verify digital signature
            boolean verified = signer.verify(
                new JcaSimpleSignerInfoVerifierBuilder()
                    .setProvider("BC").build(cert));

            if (verified)
        }
    }
}
```

```
        // extract signing time - just in case if you need it
        Date signingTime = extractSigningTime(signer);
    }
}
return verified;
}

private Date extractSigningTime(SignerInformation signer) throws Exception {
    Date signingTime = null;
    AttributeTable signedAttrs = signer.getSignedAttributes();
    if (signedAttrs == null)
        return null;

    Attribute signingTimeAttr = signedAttrs.get(CMSAttributes.signingTime);
    if (signingTimeAttr == null)
        return null;

    Enumeration<?> en = signingTimeAttr.getAttrValues().getObjects();
    while (en.hasMoreElements()) {
        Object obj = en.nextElement();
        if (obj instanceof ASN1UTCTime) {
            ASN1UTCTime asn1Time = (ASN1UTCTime) obj;
            signingTime = asn1Time.getDate();
            break;
        } else if (obj instanceof DERUTCTime) {
            DERUTCTime derTime = (DERUTCTime) obj;
            signingTime = derTime.getDate();
            break;
        }
    }

    return signingTime;
}

public X509Certificate selectCertByIssuerSerial(X500Name issuer,
        BigInteger serialNumber) throws Exception
{
    // THIS HAS TO BE IMPLEMENTED BY THE APPLICATION.

    // Retrieve signer's certificate, for instance from LDAP server,
    // or from some local storage.
    // Essentially there should be a map with issuer+serial as a key and
    // X509Certificate as value.
}

public boolean checkBlock4(final String block4, final String signature)
        throws Exception {
    final String stringToVerify = block4.replaceAll("\r\n", "\n");
    byte[] signedData = stringToVerify.getBytes("UTF-16LE");
    byte [] sign = Base64.decode(signature);
    return checkSep(data, sign);
}
}
```

## 5.4 Digital signing and verifying signatures of MX or other XML

In case of MX or other XML format messages, standard XML syntax for digital signatures is used, as defined in XML Signature standard (also called XMLDSig, XML-DSig, XML-Sig). Functionally, it has much in common with PKCS#7 (used for signing MT block4) but is more extensible and geared towards signing XML documents. It is used by various Web technologies such as SOAP, SAML, and others.

XML signatures can be used to sign XML documents. An XML signature used to sign a document outside its containing XML document is called a detached signature; if it is used to sign some part of its containing document, it is called an enveloped signature; if it contains the signed data within itself it is called an enveloping signature.

The system uses detached signatures to simplify processing, and minimize network traffic and valuable database space.

An XML Signature consists of a `Signature` element in the `http://www.w3.org/2000/09/xmldsig#` namespace. The basic structure is as follows:

```
<Signature>
  <SignedInfo>
    <CanonicalizationMethod />
    <SignatureMethod />
    <Reference>
      <Transforms>
      <DigestMethod>
      <DigestValue>
    </Reference>
    <Reference /> etc.
  </SignedInfo>
  <SignatureValue />
</Signature>
```

- The `SignedInfo` element contains or references the signed data and specifies what algorithms are used.
- The `SignatureMethod` and `CanonicalizationMethod` elements are used by the `SignatureValue` element and are included in `SignedInfo` to protect them from tampering.
- One or more `Reference` elements specify the resource being signed by URI reference; and any transforms to be applied to the resource prior to signing. A transformation can be a XPath-expression that selects a defined subset of the document tree.
- `DigestMethod` specifies the hash algorithm before applying the hash.

- `DigestValue` contains the result of applying the hash algorithm to the transformed resource.
- The `SignatureValue` element contains the Base64 encoded signature result - the signature generated with the parameters specified in the `SignatureMethod` element - of the `SignedInfo` element after applying the algorithm specified by the `CanonicalizationMethod`.

XML Signature standard also has optional `KeyInfo` element that allows the signer to include signer's certificate that validates the signature. However `KeyInfo` SHOULD NOT be used with the system to save valuable space and traffic. All certificates used in the system are available in central LDAP directory server.

XML Signature standard also has optional `Object` element that allows the signer to include the signed data if this is an enveloping signature. Detached signatures SHOULD be used in the system to avoid duplication (as the signed XML documents are anyway transmitted over network stored in databases) and save valuable database space and network traffic.

So, there are two agreements used throughout the system for XML signatures:

- 1) No certificates in signature (i.e. no `KeyInfo` element)
- 2) Detached signature (i.e. no `Object` element)

No examples are provided for the XML signatures as there are no vendor-specific conventions used in the system except for the two agreements stated above – detached signature, certificates not included into signature.

## 5.5 Digital signing for acknowledgments

STPA File Adapter automatically signs acknowledgments for the messages received from the Client System. For digital signing of these acknowledgments the following rules are used:

- **In case of ACK**, element `<Signature>` contains the digital signature of the string:

```
Data<DateTime<=%1=>MIR<=%2=>REF<=%3=>Signature<>>
```

Where:

- %1 – the string representation of the current date in the format “YYMMDDHHMM”;
- %2 – message input reference of the message that is going to be acknowledged;
- %3 – message user reference of the message that is going to be acknowledged;

**Example:**

```
Data<DateTime<=1501201112=>MIR<=141107SYSTEM22XXX0001012773=>REF<=FT1331669663=>Signature<>>
```

>

- **In case of NAK**, element `<Signature>` contains the digital signature of the string:

```
Data<DateTime<=%1=>MIR<=%2=>REF<=%3=>Signature<>Code<=%4=>Description<=%5=>Info<=%6=>
```

>

Where:

- %1 – the string representation of the current date in the format “YYMMDDHHMM”;
- %2 – message input reference of the message that is going to be acknowledged;
- %3 – message user reference of the message that is going to be acknowledged;
- %4 – error code;
- %5 – error description;
- %6 – additional error info(string);

**Example:**

```
Data<DateTime<=1501201112=>MIR<=141107SYSTEM22XXX0001012773=>REF<=FT1331669663=>Signature<>
Code<=EL26=>Description<=Invalid value of message user priority=>Info<=AA=>>
```

If the value of <any\_element> is empty string then it should be represented in the following way:

**any\_element<>**

Example of string for signing for acknowledgment with empty <Info> and <REF>:

```
Data<DateTime<=1501201112=>MIR<=141107SYSTEM22XXX0001012773=>REF<>Signature<>Code<=
EL26=>Description<=Invalid value of message user priority=>Info<>>
```

---

## 6. Installation notes

---

### 6.1 System requirements

File Adapter can be run on following operation systems:

- Windows 10 (both 32 and 64 bit version, starting from JDK 1.8.0\_51)
- Windows 8 (both 32 and 64 bit version)
- Windows 7 SP1 (both 32 and 64 bit version)
- Windows Vista SP2 (both 32 and 64 bit version)
- Windows Server 2008 SP1 (64 bit version)
- Windows Server 2012 (64 bit version)
- Solaris 11.x (x64 and SPARC, only 64 bit version)
- Solaris 10 Update 9+ (x64 and SPARC, only 64 bit version)
- Oracle Linux 7.x (64 bit version)
- Oracle Linux 6.x (both 32 and 64 bit version)
- Oracle Linux 5.5+ (both 32 and 64 bit version)
- Red Hat Enterprise Linux 7.x (64 bit version, starting from JDK 1.8.0\_20)
- Red Hat Enterprise Linux 6.x (both 32 and 64 bit version)
- Red Hat Enterprise Linux 5.5+ (both 32 and 64 bit version)
- Suse Linux Enterprise Server 12.x (64 bit version, starting from JDK 1.8.0\_31)
- Suse Linux Enterprise Server 11.x (both 32 and 64 bit version)
- Suse Linux Enterprise Server 10 SP2+ (both 32 and 64 bit version)
- Suse Linux Enterprise Server 10 SP2+ (both 32 and 64 bit version)
- Ubuntu Linux 14.x (both 32 and 64 bit version, starting from JDK 1.8.0\_25)
- Ubuntu Linux 13.x (both 32 and 64 bit version)
- Ubuntu Linux 12.04 - LTS (both 32 and 64 bit version)
- OS X 10.9 (64 bit version)
- OS X 10.8.3+ (64 bit version)

Processor: Minimum Pentium 2 266 MHz

RAM: Minimum requirement: 1 GB

Disk space: 124 MB for Java runtime environment; 50 MB for File Adapter

## 6.2 Java requirements

File adapter is pure java application, so Java SE Runtime Environment or Java SE Development Kit should be installed in the computer.

Minimum required version of Java is 8.

Additional information regarding Java installation can be found in official Java website: <http://www.java.com>

## 6.3 Installation procedure

File Adapter is delivered as simple ZIP archive. The content of this archive should be unpacked in some pre created folder. After unpacking following directory structure will be created:

- bin – directory for File Adapter libraries
- oalib – Open Adaptor framework libraries
- cfg – directory for configuration files
- props – directory for .properties files
- log – directory for log files
- input – directory for input files
- processed – directory for successfully processed files
- accepted – directory for positive acknowledgements files (ACK)
- rejected – directory for negative acknowledgements files (NAK)
- error – directory for unsuccessfully processed files
- output – directory for output processed files
- start\_stpa\_gw\_file.cmd – main command file to start adapter.
- start\_stpa\_gw\_file\_pp.cmd – command file to start adapter in batch mode.
- start\_stpa\_gw\_file.sh – \*NIX script to start adapter.
- start\_stpa\_gw\_file\_pp.sh – \*NIX script to start adapter in batch mode.

After installation adapter should be configured as described above.

To start adapter use command files start\_stpa\_gw\_file.cmd, start\_stpa\_gw\_file\_pp.cmd for windows or start\_stpa\_gw\_file.sh, start\_stpa\_gw\_file\_pp.sh for \*NIX environment.