



Mobility Data Marketplace

Technical Interface Description

Version 2.4.1 – 28.04.2015

Table of Contents

1	Introduction	6
1.1	Preamble	6
1.2	Structure of the Document.....	6
1.3	Referenced Documents	7
1.4	List of Abbreviations.....	8
2	Overview of the MDM Platform Components	10
3	Formats.....	12
3.1	DATEX II.....	13
3.2	Container Format.....	14
4	Interfaces of the MDM Broker System	15
4.1	Use of Interfaces.....	16
4.1.1	Data supplier	18
4.1.2	Data client.....	18
4.2	HTTPS Interface	19
4.2.1	Data supplier	19
4.2.2	Data client.....	22
4.3	SOAP Interface	26
4.3.1	Data supplier	26
4.3.2	Data client.....	30
4.4	OTS 2 Interface.....	34
4.4.1	Procedure	34
4.4.2	Features.....	34
4.4.3	DATEX II compression for OTS 2	34
4.4.4	OTS 2 Publish	35
4.4.5	OTS 2 Subscribe.....	38
5	Certificate-based M2M Communication.....	41
5.1	Tasks of the Security Component.....	41
5.2	Note on Server Name Indication.....	42
5.3	Applying for a Machine Certificate	42
5.4	Installing a Machine Certificate and Issuer Certificate.....	43
5.5	Authentication of the MDM Platform as Web Client	43
5.6	Authentication of Data Supplier/Data Client Web Clients.....	44

6	Exceptions and Error Messages.....	45
6.1	Exception - Unchanged Data	45
6.2	Error Messages with SOAP Requests.....	45
6.3	Error Messages with HTTPS Requests.....	45
6.4	Error Handling in the Context of OTS 2 Protocol.....	45
6.4.1	Session setup.....	45
6.4.2	Order	46
6.4.3	Data delivery	46
6.4.4	General	46
7	Examples	47
7.1	HTTPS Interface	47
7.1.1	Data Supplier Client Pull HTTPS (Container).....	47
7.1.2	Data Supplier Publisher Push HTTPS (Container)	47
7.1.3	Data client Client Pull HTTPS (DATEX II).....	48
7.1.4	Data client Client Pull HTTPS (Container).....	48
7.2	SOAP Interface	48
7.2.1	Data Supplier Publisher Push SOAP (DATEX II)	48
7.2.2	Data Supplier Client Push SOAP (Container)	49
7.2.3	Data client Client Pull SOAP (DATEX II).....	50
7.2.4	Data client Client Pull SOAP (Container).....	50
7.2.5	Data client Publisher Push SOAP (DATEX II)	50
7.3	OTS 2 Interface.....	51
7.3.1	Protocol example SOAP.....	51
8	Annex A	60
8.1	Use of Client Certificates for HTTPS Communication with Servers That Require a Certificate-based Authentication.....	60
8.2	Use of Client Certificates for the Communication with SOAP Web Services That Require a Certificate-based Authentication	66
9	Annex B	70
9.1	Processing the p12 File for Apache Server Configuration.....	70

List of tables

Table 1: Referenced documents.....	8
Table 2: List of abbreviations.....	9
Table 3: Overview of the MDM platform components	10
Table 4: Overview of the interfaces of the MDM broker system.....	16
Table 5: MDM operation modes.....	17
Table 6: Request/Response between the data supplier system and the MDM platform with the Client Pull HTTPS	21
Table 7: Request/Response between the data supplier system and the MDM platform with the Publisher Push HTTPS	22
Table 8: Request/Response between MDM platform/data client system with Client Pull HTTPS	24
Table 9: Request/Response between the MDM broker system/data client system with Publisher Push HTTPS	25

List of figures

Figure 1: Components of the MDM platform	10
Figure 2: Container Format Overview	14
Figure 3: Interfaces between data supplier, broker system and data client.....	15
Figure 4: Webservice Data supplier system/MDM broker system: DATEX II Client Pull.....	26
Figure 5: Webservice Data supplier system/MDM broker system: Container Client Pull.....	27
Figure 6: Webservice Data supplier system/MDM broker system: DATEX II Publisher Push.....	28
Figure 7: Webservice Data supplier system/MDM broker system: Container Publisher Push.....	29
Figure 8: Web service MDM broker system/Data client system: DATEX II Client Pull.....	30
Figure 9: Web service MDM broker system/Data client system: Container Client Pull.....	31
Figure 10: Web service MDM broker system/Data client system: DATEX II Publisher Push.....	32
Figure 11: Web service MDM broker system/Data client system: Container Publisher Push.....	33
Figure 12: Structure of a binary-coded OTS 2 dataAny-PDU.....	35

Figure 13: Sequence diagram OTS 2 communication between data suppliers and MDM	36
Figure 14: Sequence diagram OTS 2 communication between data clients and MDM	39
Figure 15: Overview of the security architecture	42
Figure 16: File <sammeldatei.pem>	71
Figure 17: File <sammeldatei.pem>	73

1 Introduction

1.1 Preamble

The Mobility Data Marketplace (MDM) aims at supporting the exchange of data between data suppliers and data clients using interfaces. At the same time, it is a central portal with collected information about available online traffic data of individual data suppliers. Thus, the MDM platform allows its users to offer, find and subscribe to online traffic-related data without the necessity of any time-consuming search for relevant data and a complex technical and organisational coordination between data clients and data suppliers. The data exchange is handled via standardized interfaces. In conclusion, the business processes should be simplified for all parties involved and the potential of existing data sources should be exploited.

This interface description is aimed at potential data suppliers and data clients. It is presupposed that knowledge in the implementation and operation of SOAP web services [SOAP] or HTTPS client/server architectures are provided in order to use the interfaces of the MDM system.

The interfaces offered by the MDM platform can be used by the data supplier systems and data client systems to access the services of the platform. These services for data collection or deliveries are provided by using defined and unified URLs [URL] and require a certificate-based client authentication via HTTPS [HTTPS]. For this client authentication, X.509-compliant certificates are used [PKI]. They are issued by the operator of the MDM platform. The data transfer between the MDM platform and the data supplier or data client systems can be supplied via SOAP-based web services or simple HTTPS-GET/POST requests. In addition, the transmission by [OTS2] protocol is provided.

When transmitting data between the MDM platform and the data supplier systems, both GZIP-encoded (i.e. compressed) and uncompressed HTTPS requests and responses are supported. The data transmission between the MDM platform and the data client systems always takes place using GZIP-encoded HTTPS requests and responses. If SOAP is used for transmission, the WS security standards must be adhered to. This includes a transfer of the security token and possibly the signature of the message.

1.2 Structure of the Document

This document is divided into the following sections:

- Section 1 provides a brief overview, the referenced documents and the list of abbreviations.
- Section 2 describes the components of the MDM system.

- Section 3 handles the available data formats.
- Section 4 describes the interfaces of the MDM platform for M2M communications.
- Section 5 describes the measures which secure the M2M communication.
- Section 6 shows possible messages that might occur with faulty requests to the interfaces.
- Section 7 contains XML examples for SOAP and HTTPS requests in DATEX II and container format and an example of the use of OTS 2.

1.3 Referenced Documents

[Source]	Publisher
[BHB]	MDM User Manual, V1.2 http://service.mdm-portal.de/doc/MDM-Benutzerhandbuch.pdf
[DatexIIPSM]	DATEX II V2.0 Exchange Platform Specific Model
[DatexIIPull]	DATEX II V2.0 Pull wsdl
[DatexIIPush]	DATEX II V2.0 Push wsdl
[DatexIISchema]	DATEX II XML Schema 2.0
[DatexIISDG]	DATEX II v2.0 Software Developers Guide, Version v.1.2
[DatexIISpec]	Includes the following documents, which are available to all registered users for download under http://www.datex2.eu : [DatexIIPSM], [DatexIISDG], [DatexIIUserGuide]
[DatexIIUserGuide]	DATEX II v2.0 User Guide v.1.2
[GZIP]	RFC 1952 (Mai 1996) GZIP File Format Specification Version 4.3, http://tools.ietf.org/rfc/rfc1952.txt
[HTTP/1.1]	RFC 2616 (Juni 1999) Hypertext Transfer Protocol -- HTTP/1.1 http://www.ietf.org/rfc/rfc2616.txt
[HTTPS]	RFC 2818 (Mai 2000) HTTP over TLS http://www.ietf.org/rfc/rfc2818.txt
[MCS]	MDM Container format specification http://www.mdm-portal.de

[Source]	Publisher
[OTS2]	OTS 2 Specification OTS Communication Version 02-02-09 http://www.opentrafficsystems.org
[OTS2DIN]	DIN SPEC 91213-1 Open Traffic Systems – OTS 2 Interface Specification – Part 1: Introductory remarks for decision makers; January 2011 DIN SPEC 91213-2 Open Traffic Systems – OTS 2 Interface Specification – Part 2: Technical specification for implementers; February 2011
[PKI]	RFC 2459 (January 1999) Internet X.509 Public Key Infrastructure Certificate and CRL Profile http://www.ietf.org/rfc/rfc2459.txt
[SOAP]	SOAP Version 1.2 http://www.w3.org/TR/soap12-part1/
[URL]	RFC 1738 (December 1994) Uniform Resource Locators (URL) http://www.ietf.org/rfc/rfc1738.txt
[X.509v3]	ITU-T Recommendation X.509 (1997 E): Information Technology - Open Systems Interconnection – The Directory: Authentication Framework, June 1997. http://www.itu.int/rec/T-REC-X.509-199708-S/en

Table 1: Referenced documents

1.4 List of Abbreviations

Abbreviation	Explanation
BASE64	BASE64 describes a method of encoding 8-bit binary data into a string that consists only of readable code page-independent ASCII characters.
BAST	Bundesanstalt für Straßenwesen (Federal Highway Research Institute)
DE	German
GMT	Greenwich Mean Time
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure

Abbreviation	Explanation
ID	Identifier
IIS	Microsoft Internet Information Services
IT	Information Technology
JSSE	Java Secure Socket Extension
M2M	Machine-to-Machine
MDM	Mobility Data Marketplace
MDV	Metadatenverzeichnis (metadata directory)
OTS	Open Traffic Systems
PAS	Publicly Available Specification
PKI	Public Key Infrastructure
PSM	Platform Specific Model
RC	Release Candidate
RFC	Request for Comments
SDG	Software Developers Guide
SNI	Server Name Indication
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator
UTF	UCS Transformation Format
WS	Web Server
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Table 2: List of abbreviations

2 Overview of the MDM Platform Components

The MDM platform consists of four components that fulfill different roles.

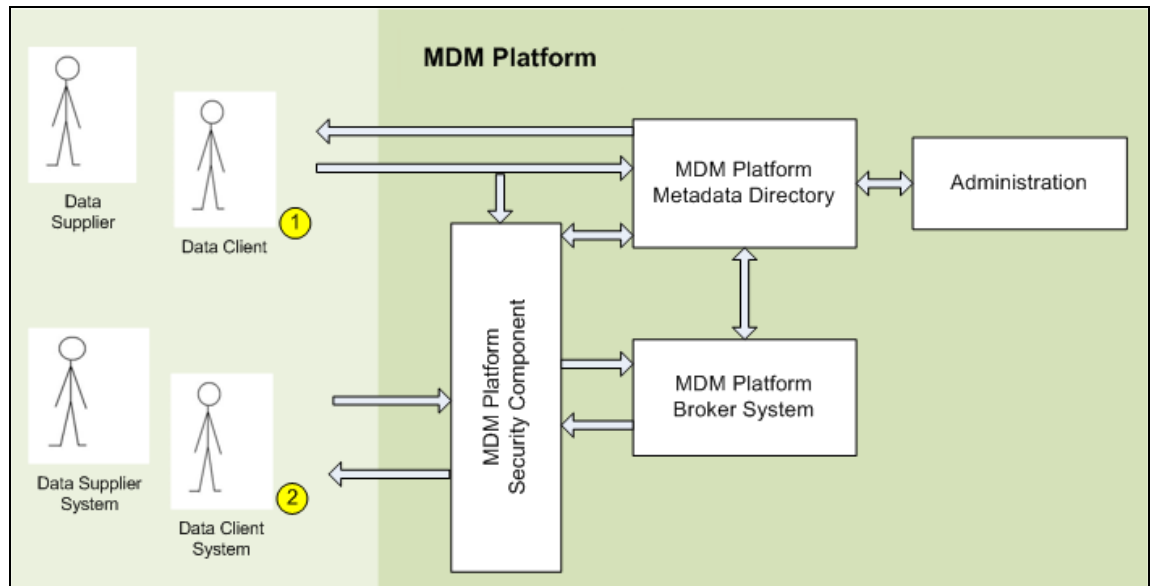


Figure 1: Components of the MDM platform

Component	Description
Security component	Via the security component, the data client system/data supplier system can be authenticated to use the services.
Metadatenverzeichnis (metadata directory)	The metadata directory is used to manage all information relevant to MDM platform and provides a number of organisational services.
Broker system	The broker system handles the actual processing of the data packets and it is, therefore, the focus of this interface description.
Administration	The administration is realized by means of a web-based user interface (GUI), see [BHB]

Table 3: Overview of the MDM platform components

The following communication and application scenarios are supported by the MDM platform:

- Interested parties as well as data clients and data suppliers can communicate with the metadata directory by using the web GUI, in order to access services, such as researching or registering. To view or edit certain content of the metadata directory, an authentication must first be run throughout the MDM platform security component.
- Following an authentication via the security component, the data client and data supplier systems can establish an M2M communication with the broker system to deliver or request data.

3 Formats

In order to exchange mobility data between the broker system and the data supplier and data client systems, the following data formats are specified:

- The MDM platform supports the XML-based format DATEX II by means of native interfaces to allow rapid utilization of the platform using standard-compliant DATEX II implementations.
- In order to create a generic interface independent from any specific format, a new data format is provided for transmission. With the so-called container format any XML and binary data can be transmitted.

The validity of the data is checked and logged upon delivery of a data packet to the MDM broker interface. For this purpose, the schema file is based on the URL that is stored in the publication description. For publications in DATEX II format, it is the responsibility of the data supplier to provide the correct file schema. For publications in container format, the standard schema is already made available under a generally valid URL. Please, reference this URL in the "schemaLocation" attribute of your XML data packets to provide data clients with an automatic validation of the packets. The MDM accepts the data packets independent of the validation result and delivers them to the data clients even if the result is negative.

3.1 DATEX II

DATEX II is a European standard for exchanging mobility data. Basic knowledge of DATEX II specification is required for this section [DatexIISpec]. For the MDM interface, the DATEX II specification is used in version 2.0.

DATEX II defines XML structures for the exchange of mobility data. The underlying scheme can be viewed under <http://www.datex2.eu/>. The payload must be defined on the basis of this scheme. DATEX II determines not only a standard for the structure of the payload, but also regulates the exchange process. The latter is described in detail in chapter 4.

The underlying documents on which DATEX II is based are listed in chapter "Referenced Documents"[DatexIISpec]. The structure of the DATEX II payload is not relevant to the MDM platform, as the latter transfers the data unchanged and does not process the data in any way.

DATEX II not only provides for the dispatch of complete data packets, but also for sending updates to previous versions. This DATEX II option is **not** supported by the MDM platform: Both the data supplier system and the MDM broker system must always send complete data packets.

This means that each packet contains all records of the relevant publication that are known to the data supplier. These records are valid at the time of packet sending. It is therefore not possible to send only changes to the "last known" state. This may seem to be a disadvantage, but it is a requirement that is essential to the preservation of the MDM system scalability. The disadvantage of this partial redundancy is tacitly accepted, as it is taken into account by the scalable architecture of the platform and the performance of modern ICT infrastructure. It should be borne in mind that the MDM platform diminishes the burden of scalability of the data suppliers.

3.2 Container Format

In addition to the DATEX II standard mentioned in the previous section, another XML-based model for the transmission of data is supported by the MDM platform. This container format called data format has been specially created for the exchange of data via the MDM. The schema of the data format is found in the container format specification [MCS]. In addition to the actual payload that is contained in a body element, the data format allows more structural information to be transmitted in a header element. This information is particularly used to control the communication process.

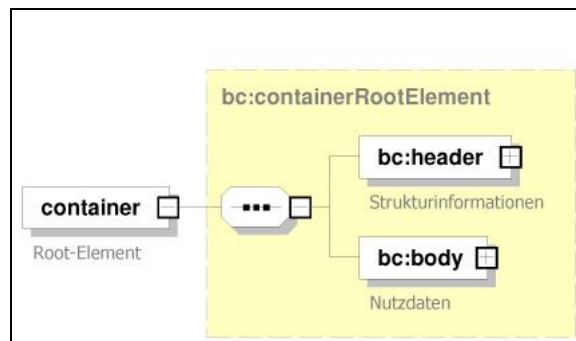


Figure 2: Container Format Overview

In order to keep the model flexible, the format and content of the body element is not specified. Thus, not only data in XML format can be transported in containers, but also binary data.

4 Interfaces of the MDM Broker System

The MDM broker system takes the role of the client or the role of the server as an intermediary between the data supplier system and the data client system, depending on the situation:

- As a client, the broker system can request data from the data supplier or the data supplier can - on his own initiative - send the data to the broker system.
- As a client, the data client can on its part request data from the broker system or the broker system can send - on its own initiative - the data to the data client.

Figure 3 shows the possible paths that are available for data packet transmission between the data supplier and the broker system on the one hand and the broker system and the data client on the other.

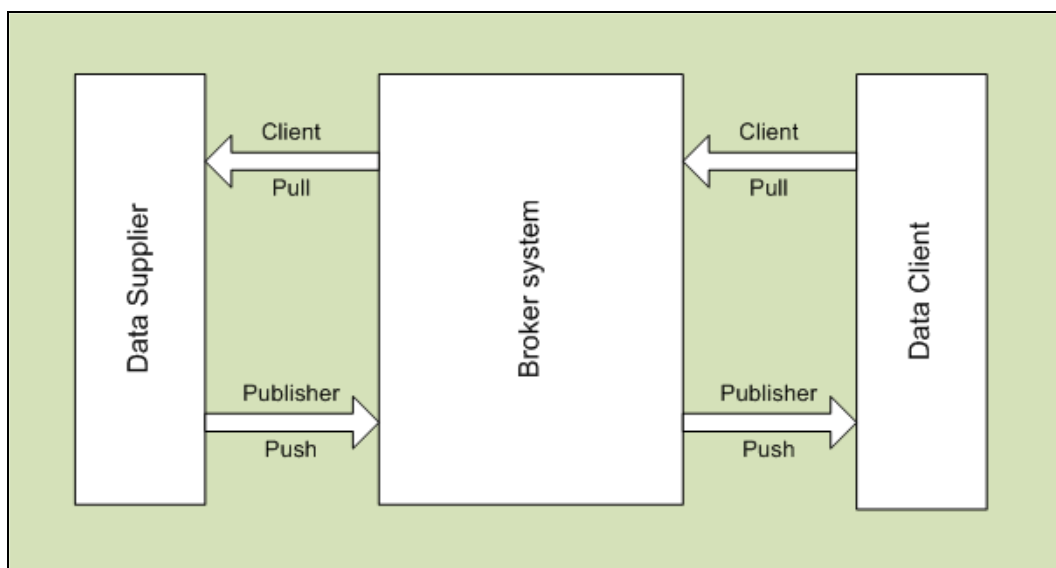


Figure 3: Interfaces between data supplier, broker system and data client

The data packets received or sent by the broker system must be in DATEX II format or in self-defined container format.

The transmission protocols HTTPS and SOAP via HTTPS are supported for each format. For the format DATEX II, the OTS 2 protocol is also supported.

The following table shows what communications are supported. The section in which the relevant communication is described - distinguished by the data supplier and data client systems - is mentioned for each data format (DATEX II / container), communication pattern (Client Pull / Publisher Push) and protocol (HTTPS, SOAP, OTS 2), if supported.

It is additionally indicated whether the data supplier or data client system acts as a client or as a server towards the MDM. Client here means that the system makes enquiries to the MDM or actively establishes the connection to it.

On the other hand, server means that the system is contacted by the MDM and must answer its enquiries. In this case, an external network access to the system to be connected to the MDM, must be allowed.

		Data supplier system			Data client system		
		HTTPS	SOAP	OTS 2	HTTPS	SOAP	OTS 2
DATEX II	Client Pull	4.2.1.1 Server	4.3.1.1 Server	-	4.2.2.1 Client	4.3.2.1 Client	-
	Publisher Push	-	4.3.1.3 Client	4.4.4 Client	-	4.3.2.3 Server	4.4.5 Client
Container	Client Pull	4.2.1.2 Server	4.3.1.2 Server	-	4.2.2.2 Client	4.3.2.2 Client	-
	Publisher Push	4.2.1.3 Client	4.3.1.4 Client	-	4.2.2.3 Server	4.3.2.4 Server	-

Table 4: Overview of the interfaces of the MDM broker system

If the SOAP method is used, the WSDL of the broker service can generally be queried at the service endpoint that is specific to the relevant publication or subscription using the ?wsdl request.

4.1 Use of Interfaces

When using the HTTPS or SOAP protocol, there are three different modes of operation for the exchange of data, all of which are supported by the MDM platform:

Mode	Description
Client Pull	The communication is initiated by the client (MDM broker system to data supplier or data client system to MDM platform) and the data is sent as a response.
Publisher Push Periodic	The communication is initiated by the publisher (data supplier system to MDM platform) at timed intervals.
Publisher Push on Occurrence	The communication will always be initiated by the publisher (data supplier system to MDM platform or MDM broker system to data client) if the data changes.

Table 5: MDM operation modes

The OTS 2 protocol [OTS2] works session-based and by publish-subscribe (data subscription). It differs from the other interfaces provided by MDM.

The valid specification of the OTS 2 protocol including associated schema and WSDL files can be obtained from the OTS website [OTS2]. The available content complies with the DIN SPEC (PAS) 91213 [OTS2DIN].

The use of the MDM OTS 2 interface requires a client that implements the OTS 2 protocol stack. Since OTS 2 has a higher complexity than the other interfaces provided by MDM due to its extensive capabilities (only partially required for the MDM interface) its use must be weighed carefully.

The use will particularly be rewarding if other OTS 2 interfaces are operated or planned in the client system or if the advantage of the push mode is used for data clients. This advantage consists of the ability to transfer - without delays caused by additional latency by polling - MDM data in push mode, without having to implement a server that makes an opening of its own network necessary for external access (for the MDM) (see chapter 4.4.5). Thus, the data client is supplied as soon as possible. It does however not open its network to the outside as in the corresponding (Push) HTTPS and SOAP protocol options of MDM.

OTS 2 uses only DATEX II as data model, as described in this document or in [DatexIISpec]. Regarding the dispatch of complete data packets (as opposed to the dispatch of changes), the descriptions in chapter 4.4.2 shall apply. Compression is applied only for the actual payload, not for the OTS 2 protocol part (see chapter 4.4.3).

4.1.1 Data supplier

Towards the data supplier (the publisher), the MDM broker system appears as a subscriber who receives the data packets. The broker system can assume the role of a server or a client, depending on the procedure.

When using the OTS 2 protocol, the broker system takes the role of an OTS 2 distributor, the data supplier system takes the role of an OTS 2 publisher.

4.1.2 Data client

Towards the data client (the subscriber), the MDM broker system appears as a publisher who provides the data packets. The broker system can assume the role of a server or a client, depending on the procedure.

When using the OTS 2 protocol, the broker system takes the role of an OTS 2 distributor, the data client system takes the role of an OTS 2 subscriber.

4.2 HTTPS Interface

4.2.1 Data supplier

4.2.1.1 Client Pull HTTPS (DATEX II)

As with the client pull exchange process, the MDM broker system requests the data supplier system periodically to deliver its data to the MDM platform. The time interval used must be configured in the metadata directory when configuring the data services. For this exchange, the points C1-C12 from the Simple HTTP Server Profile of the [DtexIIPSM] shall apply.

It should be noted that the additional, optional rules do not apply. The options for authentication (C13, C14, C17) do not apply, as they are obsolete when using the HTTPS method that is compulsory for MDM. C18-C27 do no longer apply, since the options relate only to the optional provision of DATEX II data in file format, which is not applicable to MDM.

4.2.1.1.1 Request to data supplier

The MDM broker system sends an HTTPS GET request to the data supplier system from which the data is to be collected. The MDM platform is able to identify the data supplier systems that have subscribed to a pull method, and to send requests to them at defined intervals.

Via the MDM administration component, the data supplier must enter the publication-specific server URL in the publication configuration.

The broker system sends the request with an "If-Modified-Since" header field whenever the data supplier system had set the header field "Last-Modified" in its response (see [HTTP/1.1]). The data supplier system should always set this header field to enable the MDM platform to use this feature. As a result, the transfer of already collected data packets can be prevented.

Example:

If the response of the previous data packet contains the following header line

`Last-Modified: Sat, 29 Oct 1994 19:43:31 GMT`

the next data packet will be requested with a request, which contains the following header line:

`If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT`

4.2.1.1.2 Response to the MDM platform

After receipt of the request, the data supplier system must generate an HTTPS response whose message body consists of the requested DATEX II data. Pursuant to [DatexIIPSM] section 4, the response has the content type "text/xml; charset=utf-8" and should be available as GZIP encoding.

The MDM broker system accepts this data and stores it in a packet buffer. A previous data packet, if it still exists, will be replaced.

4.2.1.2 Client Pull HTTPS (Container)

The MDM broker system prompts the data supplier system to periodically deliver a packet for a publication to the MDM platform. The time interval used must be configured in the metadata directory when configuring the data services.

4.2.1.2.1 Request to data supplier

The broker system sends an HTTPS GET request to the data supplier system. As a parameter, the publication ID of the publication for which a data packet has to be delivered is handed over to the MDM.

Via the MDM administration component, the data supplier must enter its URL in the publication configuration.

The URL of the data supplier system from the publication configuration is complemented by appending the publication ID:

Example:

Data supplier configured in metadata directory as URL for pickup:

```
https://<DG-Maschine>/<context>
```

The ID of the associated publication is 2000002. This results in the following URL for the request by the MDM broker system:

```
https://<DG-Maschine>/<context>?publicationID=2000002
```

4.2.1.2.2 Response to the MDM platform

The data supplier system must respond to the request with an HTTPS response. The content type of the response must be of the type "text/xml" and should be available as GZIP encoding. Non-compressed content can also be processed by the MDM platform. The message body has to include the requested data packet. The standard HTTP status codes [HTTP/1.1] must be used, whereby the explanations described in Table 6 shall apply.

Description	
Request	GET /anfrageServlet?publicationID=2000002 HTTP/1.1 Host: Data supplier host Accept-Encoding: GZIP
Response	HTTP/1.1 200 OK Content-Type: text/xml Content-Length: xx <container> ... </container>
Statuscodes	Standard HTTP1.1 Statuscodes [HTTP/1.1] The following status codes have a particular meaning: - 400: No publication parameter has been given - 404: Publication parameters could not be assigned

Table 6: Request/Response between the data supplier system and the MDM platform with the Client Pull HTTPS

4.2.1.3 Publisher Push HTTPS (Container)

The data supplier system has to send a data packet for a publication to the MDM broker system.

4.2.1.3.1 Request to the MDM broker system

The data supplier system must send an HTTPS POST request with a message in container format to the MDM broker system. In this process, the publication ID in the header element and the payload in the body element of the container message must be delivered .

The URL of the broker system is constructed as follows:

```
https://<BAST-MDM-Broker-Server>/BAST-MDM-Interface/srv/container/v1.0
```

Example:

```
https://service.mac.mdm-portal.de/BAST-MDM-Interface/srv/container/v1.0
```

4.2.1.3.2 Request to data supplier

In response to the request, the data supplier system receives an HTTPS response. The message body is empty. The standard HTTP status codes [HTTP/1.1] may be used as status codes, whereby the explanations described in Table 7 shall apply.

Description	
Request	Request POST/data delivery HTTP/1.1 Host: mdmhost Content-Type : text/xml Accept-Encoding: GZIP <container> ... </container>
Response	Response HTTP/1.1 200 OK
Status codes	Standard HTTP1.1 Status Codes [HTTP/1.1] The following status codes have a particular meaning: - 400: No publication parameter or no data has been given - 404: The publication parameter could not be assigned or the publication is no longer valid

Table 7: Request/Response between the data supplier system and the MDM platform with the Publisher Push HTTPS

4.2.2 Data client

4.2.2.1 Client Pull HTTPS (DATEX II)

With the client pull exchange process, the data client system must prompt the MDM broker system to transfer the data.

4.2.2.1.1 Request to the MDM platform

The data client system must send an HTTPS GET request to the URL of the MDM platform. Due to the subscription ID, the associated packet buffer and the data packet are determined.

The URL of the broker system is constructed as follows:

```
https://<BAST-MDM-Broker-Server>/BAST-MDM-  
Interface/srv/<Subskriptions-  
ID>/clientPullService?subscriptionID=<Subskriptions-ID>
```

Example:

```
https://service.mac.mdm-portal.de/BASSt-MDM-Interface/srv/2000000/clientPullService?subscriptionID=2000000
```

The broker system supports requests that have an "If-Modified-Since" header field. For this purpose, the responses of the broker system always contain the header field "Last-Modified" (see [HTTP/1.1]). If the data client system wants to use this feature, it must always transmit the value from the last Last-Modified header field. As a result, the transfer of already collected data packets can be prevented. It is strongly recommended that you implement this feature on the data client side.

Example:

If the response of the previous data packet contains, for example, the following header field

```
Last-Modified: Sat, 29 Oct 1994 19:43:31 GMT
```

the next data packet must be requested with a request that contains the following header field:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

4.2.2.1.2 Response to data client

The MDM broker system generates an HTTPS response after receipt of the request. For this purpose, the associated packet buffer and the appropriate data packet will be determined on the basis of the subscription ID. The content of the data packet is sent to the data client in the body of the response. Pursuant to DATEX II Client Pull HTTP profile [DatexIIPSM] section 4, the response has the content type "text/xml; charset=utf-8".

4.2.2.2 Client Pull HTTPS (Container)

With the client pull exchange process, the data client system must prompt the MDM broker system to transfer the data. Which subscription is affected has to be specified by a request parameter.

4.2.2.2.1 Request to the MDM platform

The data client system must send an HTTPS GET request to the MDM platform. As a parameter, the subscription ID of the subscription for which a data packet has to be delivered is handed over to the MDM.

The URL of the broker system is constructed as follows:

```
https://<BASSt-MDM-Broker-Server>/BASSt-MDM-Interface/srv/container/v1.0?subscriptionID=<Subskriptions-ID>
```

Example:

```
https://service.mac.mdm-portal.de/BASSt-MDM-  
Interface/srv/container/v1.0?subscriptionID=2000000
```

4.2.2.2.2 Response to the data client system

The MDM broker system generates an HTTPS response after receipt of the request. The standard HTTP status codes [HTTP/1.1] can be used, whereby the explanations described in Table 8 shall apply. The content type of the response is of the type "text/xml" and is sent GZIP-compressed. The message body of the response consists of the requested data packet.

Description	
Request	Request GET /BASSt-MDM- Interface/srv/container/v1.0?subscriptionID=2000000 HTTP/1.1 Host: mdmhost Accept-Encoding: GZIP
Response	Response HTTP/1.1 200 OK Content-Type: text/xml Content-Length: xx <container> ... </container>
Statuscodes	Standard HTTP1.1 Statuscodes [HTTP/1.1] The following status codes have a particular meaning: <ul style="list-style-type: none">- 204: No data packet in the packet buffer for subscription- 400: No subscription parameter- 404: None or no longer valid subscription to the subscription parameter found

Table 8: Request/Response between MDM platform/data client system with Client Pull HTTPS

4.2.2.3 Publisher Push HTTPS (Container)

The MDM broker system sends a data packet of a subscription to a data client system.

4.2.2.3.1 Request to the data client system

The MDM broker system sends an HTTPS POST request to the data client system in which the subscription ID is delivered in the header element and the payload in the body element of the container message.

Via the MDM administration component, the data client must enter its URL in the subscription configuration.

4.2.2.3.2 Response to the MDM platform

The data client system must respond to the request with an HTTPS response.

The message body is empty. The standard HTTP status codes [HTTP/1.1] may be used as status codes, whereby the explanations described in Table 9 shall apply.

Description	
Request	Request POST/data delivery HTTP/1.1 Host: Data client host Content-Type : text/xml Accept-Encoding: GZIP <container> ... </container>
Response	Response HTTP/1.1 200 OK
Status codes	Statuscodes Standard HTTP1.1 Statuscodes [HTTP/1.1] The following status codes have a particular meaning: - 400: No subscription parameter or no data has been given - 404: Subscription parameters could not be assigned

Table 9: Request/Response between the MDM broker system/data client system with Publisher Push HTTPS

4.3 SOAP Interface

4.3.1 Data supplier

4.3.1.1 Client Pull SOAP (DATEX II)

As with the Client Pull SOAP exchange process, the MDM broker system requests the data supplier system to deliver its data to the MDM platform.

4.3.1.1.1 Offering a web service

The data supplier system must provide a web service that is defined according to the DATEX II Pull WSDL [DatexIIPull]. Null is thereby expected as input. As output, the MDM broker system gets in return the requested data in DATEX II format.

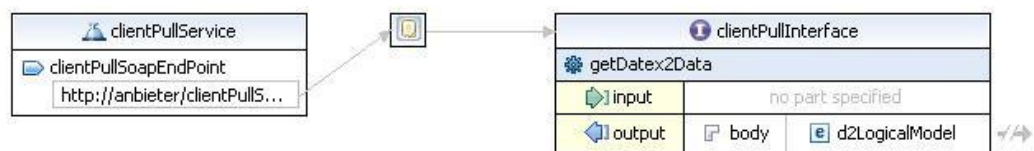


Figure 4: Webservice Data supplier system/MDM broker system: DATEX II Client Pull

Via the MDM administration component, the data supplier must enter its URL in the publication configuration.

4.3.1.1.2 Calling up a web service

The MDM broker system provides a web service client that is defined according to the DATEX II pull WSDL [DatexIIPull] to invoke web services. This web service must return data according to the schema [DatexIISchema].

The broker system identifies the data supplier systems that have subscribed to a pull method, and the associated service endpoints in the metadata directory and periodically calls them up according to the configured publication frequency. The data received after the call is cached in corresponding packet buffers for delivery to potential data clients. A previous data packet, if it still exists, will be replaced.

4.3.1.2 Client Pull SOAP (Container)

As with the Client Pull SOAP exchange process, the MDM broker system requests the data supplier system periodically to deliver its

data to the MDM platform. The time interval used must be configured in the metadata directory when configuring the data services.

4.3.1.2.1 Offering a web service

The data supplier system has to offer a web service that expects as input the parameters publication ID and time stamp with the date of creation according to the elements of the container model schema. The data supplier system must generate and return a data packet in the container format for the transferred publication ID.

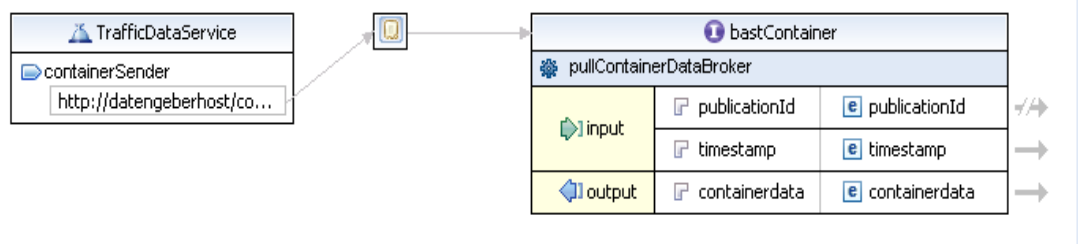


Figure 5: Webservice Data supplier system/MDM broker system: Container Client Pull

Via the MDM administration component, the data supplier must enter the service endpoint in the URL attribute of the publication configuration.

4.3.1.2.2 Calling up a web service

The MDM broker system provides a web service client that is defined according to the container format specification [MCS] to invoke web services.

The broker system identifies the data supplier systems that have subscribed to a pull method, and the associated service endpoints in the metadata directory and periodically calls them up according to the configured publication frequency. The data received after the call is cached in corresponding packet buffers for delivery to potential data clients. A previous data packet, if it still exists, will be replaced.

4.3.1.3 Publisher Push SOAP (DATEX II)

With the Publisher Push exchange process, the data supplier system must deliver the data to the MDM platform on its own initiative. In this process, an appropriate SOAP interface must be used. Whether the data is event-based (on occurrence) or periodically generated and delivered to the MDM platform is irrelevant to the operation of the

MDM broker system. The mechanism for the exchange is the same in both cases.

4.3.1.3.1 Offering a web service

The MDM broker system provides a web service that is defined based on the specification DATEX II Push WSDL [DatexIIPush]. The data to be supplied is expected as input. As output, the data supplier system gets in return confirmation data in DATEX II format.

The output consists of an acknowledgement of receipt.

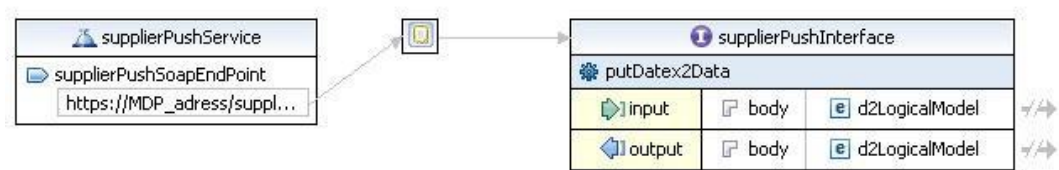


Figure 6: Webservice Data supplier system/MDM broker system: DATEX II Publisher Push

In the broker system, the ID of the publication, the data packets belong to, is entered in the URL of the service endpoint.

The URL is structured as follows:

```
https://<BASt-MDM-Broker-Server>/BASt-MDM-Interface/srv/<publication ID>/supplierPushService
```

Example:

```
https://service.mac.mdm-portal.de/BASt-MDM-Interface/srv/2000002/supplierPushService
```

4.3.1.3.2 Calling up the web service

The data supplier system has to provide a web service client that is defined according to DATEX II Push WSDL [DatexIIPush] to call up the web service. The web service must deliver the data to the publication-specific service endpoint of the MDM broker system. The MDM broker system accepts this data and stores it in a packet buffer. A previous data packet, if it still exists, will be replaced.

4.3.1.4 Publisher Push SOAP (Container)

With the Publisher Push exchange process, the data supplier system must deliver the data to the MDM platform on its own initiative. In this process, an appropriate SOAP interface must be used. Whether the

data is event-based (on occurrence) or periodically generated and delivered to the MDM platform, is irrelevant to the operation of the MDM broker system. The mechanism for the exchange is the same in both cases.

4.3.1.4.1 Offering a web service

The MDM broker system provides a web service, which expects - as input - the data structure of the container format filled with the publication ID in the header element and a data packet in the body element, and returns a status message as output.

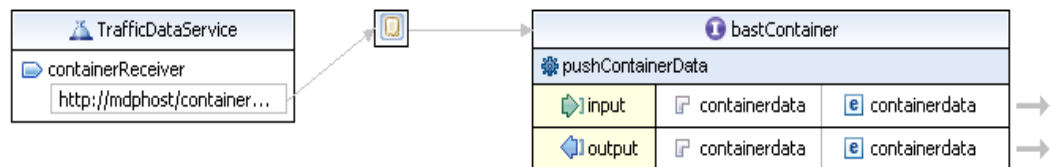


Figure 7: Webservice Data supplier system/MDM broker system: Container Publisher Push

4.3.1.4.2 Calling up the web service

The data supplier system must provide a web service client in accordance with the container format specification [MCS]. This client serves to launch the web service.

The SOAP endpoint of the broker system is as follows:

```
https://<BAST-MDM-Broker-Server>/BAST-MDM-Interface/srv/container/v1.0
```

Example:

```
https://service.mac.mdm-portal.de/BAST-MDM-Interface/srv/container/v1.0
```

4.3.2 Data client

4.3.2.1 Client Pull SOAP (DATEX II)

With the Client Pull SOAP exchange process, the data client system must prompt the MDM platform to transfer the data to the data client system.

4.3.2.1.1 Offering a web service

The MDM broker system provides a web service that is defined based on the specification [DatexIIPull]. As input, the subscription ID is expected here in the URL, as output, the data client gets in return the requested data in DATEX II format. Based on the transmitted subscription ID, the MDM platform can identify the corresponding packet buffer and the data packet.

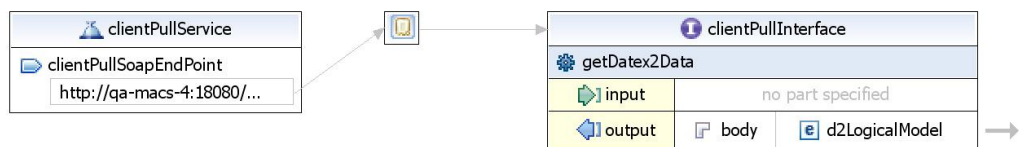


Figure 8: Web service MDM broker system/Data client system: DATEX II Client Pull

4.3.2.1.2 Calling up the web service

The data client system must provide a web service client that is defined according to the specification [DatexIIPull] to invoke web services. The corresponding subscription ID must be carried in the URL as input parameter.

The SOAP endpoint of the broker system is as follows:

```
https://<BAsT-MDM-Broker-Server>/BAsT-MDM-Interface/srv/<Subskriptions-ID>/clientPullService
```

Example:

```
https://service.mac.mdm-portal.de/BAsT-MDM-Interface/srv/2000000/clientPullService
```

4.3.2.2 Client Pull SOAP (Container)

With the Client Pull SOAP exchange process, the data client system must prompt the MDM platform to transfer the data to the data client system.

4.3.2.2.1 Offering a web service

The MDM broker system provides a web service, which expects - as input - a subscription ID and a timestamp (includes the creation time of the request). The data is returned as output in the container format.

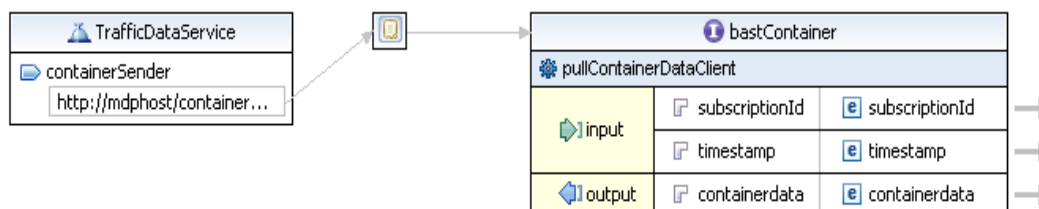


Figure 9: Web service MDM broker system/Data client system: Container Client Pull

4.3.2.2.2 Calling up the web service

The data client system must provide a web service client in accordance with the container format specification [MCS]. This client serves to launch the web service.

The SOAP endpoint of the broker system is as follows:

```
https://<BAST-MDM-Broker-Server>/BAST-MDM-Interface/srv/container/v1.0
```

Example:

```
https://service.mac.mdm-portal.de/BAST-MDM-Interface/srv/container/v1.0
```

4.3.2.3 Publisher Push SOAP (DATEX II)

With the Publisher Push exchange process, the MDM broker system delivers the data to the data client systems on its own initiative. In this process, an appropriate SOAP interface must be used. Whether the data is event-based (on occurrence) or periodically generated and delivered to the MDM platform is in this case irrelevant; the mechanism for the delivery to the data client is identical.

4.3.2.3.1 Offering a web service

The data client system must provide a web service that is defined according to the specification [DatexIIPush]. The data to be supplied is expected as input. As output, the MDM platform gets in return

confirmation data in DATEX II format .The format of the input parameter corresponds to the DATEX II scheme [DatexIISchema].

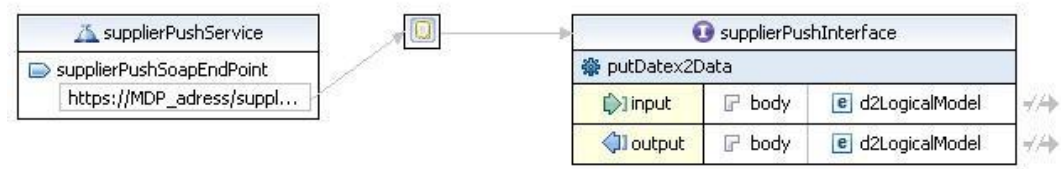


Figure 10: Web service MDM broker system/Data client system: DATEX II Publisher Push

4.3.2.3.2 Calling up the web service

The MDM broker system provides a web service client that is defined according to the [DatexIIPush] to invoke the web services of the data client system. Via the MDM administration component, the data client must enter its service endpoint in the subscription configuration.

The broker system identifies the data client systems and launches a corresponding web service call.

If the data transfer could be successfully completed, the broker system would then expect a confirmation message from the data client system:

```
<D2LogicalModel:d2LogicalModel modelBaseVersion="2"
xsi:schemaLocation="http://datex2.eu/schema/2/2_0/
DATEXIISchema_2_2_0.xsd"
xmlns:D2LogicalModel="http://datex2.eu/schema/2/2_0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <D2LogicalModel:exchange>
    <D2LogicalModel:response>acknowledge</D2LogicalModel:response>
  </D2LogicalModel:exchange>
  ...
</D2LogicalModel:d2LogicalModel>
```

4.3.2.4 Publisher Push SOAP (Container)

With the Publisher Push exchange process, the MDM broker system delivers the data to the data client systems on its own initiative. In this process, an appropriate SOAP interface must be used. Whether the data is event-based (on occurrence) or periodically generated and delivered to the MDM platform is in this case irrelevant; the mechanism for the delivery to the data client remains identical.

4.3.2.4.1 Offering a web service

The data client system must provide a web service that is defined on the basis of the specification [MCS]. A data packet of the type container format must be accepted as input and, as output, a status message must be delivered.

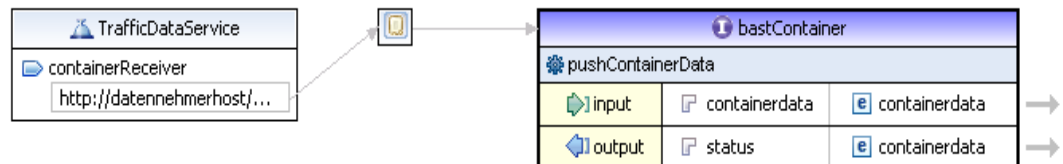


Figure 11: Web service MDM broker system/Data client system: Container Publisher Push

4.3.2.4.2 Calling up the web service

The MDM broker system provides a web service client that is defined according to the container format specification [MCS] to invoke the web services of the data client system. Via the MDM administration component, the data client must enter its service endpoint in the URL attribute of the subscription configuration.

The broker system identifies the data client systems and launches a corresponding web service call.

If the data transfer could be successfully completed, the broker system would then expect a status message from the data client system:

4.4 OTS 2 Interface

4.4.1 Procedure

For delivering or retrieving data, the external client must initially establish a session with the OTS 2 server of the MDM platform.

In response, a data order (subscription) takes place within the session by the intended recipient (data client or MDM). The data transmitter (MDM or data supplier) will then transmit the required data within the existing session automatically and continuously (without any further queries) until the order is completed.

4.4.2 Features

The methods of the OTS 2 activity layer used for communication with the MDM include *ATie*, *ASubscribe* (for data clients) / *ASnippet* (for data suppliers), *AUnsubscribe* and *AUntie*. The used method calls comprise *onATied*, *onASnippet* (for data clients) / *onASubscribe* (for data suppliers), *onAUnsubscribe* and *onAUntied*. The data contents *application*, *subscriptionAny* and *dataAny* are transported.

Only the OTS 2 protocol range, which enables the transmission of any up-to-date DATEX II packages by *subscriptionAny*, will therefore be needed. Singular queries and commands are not used. Historical data cannot be obtained from the MDM. In addition, special requests to read out the current state and then subsequent changes are not supported. Therefore, data packets coming from the data suppliers must always describe the complete current state.

The data clients cannot retrieve any selection of objects belonging to a service. The data packets are always completely submitted (MDM subscription). If necessary, data suppliers should split their data offering into several publications, in order to enable the data clients to subscribe to only a subset of the available data services.

4.4.3 DATEX II compression for OTS 2

On the data client side, the MDM basically provides compressed data. In contrast to the use of HTTPS and SOAP protocols, only the DATEX II payload is compressed in the case of OTS 2 and not the complete OTS 2 package. If the package is considered at HTTP level, it will then constitute an uncompressed packet. Given the specific requirements regarding the processing time of data packets through the MDM and the relatively long computation time required for the compression of a single data packet, it has been decided that only the DATEX II payload would be transmitted compressed and the subscription-specific individual OTS 2 frame would be transmitted uncompressed. Thus, the

MDM can also submit a once compressed data packet to a number of data clients using OTS 2.

For this purpose, an OTS 2 snippet of the type *acDataAnyType* (other types are not used) includes a BASE64Binary-encoded binary package, which contains the DATEX II package in GZIP-compressed form.

The binary package is embedded in a <binary> element. The attribute *type* of the <binary> element identifies the type of data transferred and it is here provided with "base64BinaryDatex2Gzip".

This results in the following structure within an OTS 2 snippet:

```
<dataAny>
  <binary type="base64BinaryDatex2Gzip">
    PGQyTG9naWNhbE1vZGVsIHhtbG5zPSJodHRwOi8vZGF0ZXgyLmV1L3NjaGVt...
  </binary>
</dataAny>
```

Figure 12: Structure of a binary-coded OTS 2 dataAny-PDU

To restore the original DATEX II package, a data client must first decode the content of the <binary> element from the *dataAny* snippet using BASE64Binary and then decompress it using GZIP.

In addition to classic compression at HTTP level, OTS 2 data suppliers can also use this variant of compression during delivery; the use of this method is recommended.

4.4.4 OTS 2 Publish

When using the OTS 2 protocol, the data supplier takes the role of an OTS 2 publisher.

Depending on the MDM publication, a separate connection has to be established to the designated service endpoint (OTS 2 method *ATie*).

The URL of the service endpoint is structured as follows:

```
soap.tls://<BASt-MDM-Broker-Server>/BASt-MDM-OTS2-
Interface/pub/<publicationID>
```

Example:

```
soap.tls://service.mac.mdm-portal.de/BASt-MDM-OTS2-Interface/pub/2004000
```

The connections are usually maintained - as long as possible - permanently (and not rebuilt, e.g. every minute). The authentication takes place only when establishing a connection (see below).

The following Figure 13 shows an example of a sequence with a connection establishment, order by the MDM, data delivery by the client (here are just two deliveries indicated) and a connection terminated by the client (a disconnection in the reverse direction by the MDM would also be possible). The data supplier (client) is located on the left side.

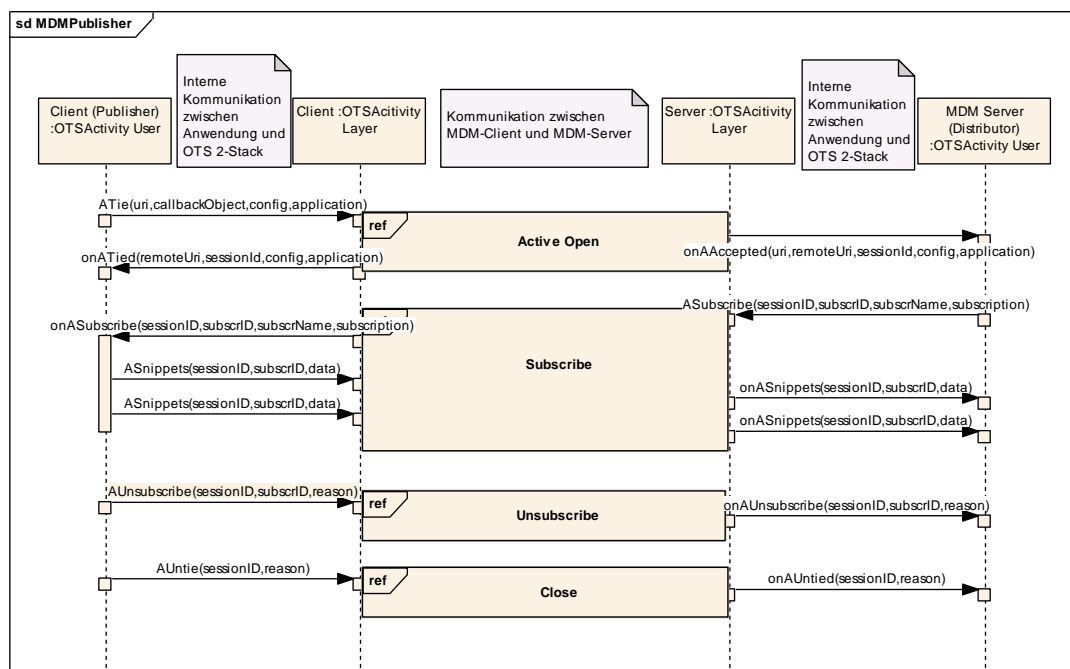


Figure 13: Sequence diagram OTS 2 communication between data suppliers and MDM

4.4.4.1 Connection establishment

By using its machine certificate, the data supplier system has to establish a TLS connection in the direction of MDM with the OTS 2 protocol connection "SOAP/HTTP with TLS encryption" (OTS 2 method *ATie*, "soap.tls:" is in the URL schema field).

For the implementation of the certificate transmission, please refer to chapter 8.1 (Use of Client Certificates for HTTPS Communication with Servers That Require a Certificate-based Authentication).

The data supplier must set the following special features in the configuration information:

- "a_publisher=1" (indicates that it is a data supplier)
- "a_c_datex_any_mdm=1" (encodes the specific conditions for the use of OTS 2 protocol in MDM)
- "t_targetURI" with the complete target URI, e.g.
"t_targetURI=soap.tls://service.mac.mdm-portal.de/BASSt-MDM-OTS2-Interface/pub/2004000" (is internally required by the MDM server OTS 2)

To establish a connection, see also [OTS2] chapter 7.6.4.1 and 7.6.4.2.

4.4.4.2 Order

Pursuant to OTS 2 protocol, after a successful connection establishment (OTS 2 method call *onATied*) the data supplier has to wait until the MDM places an order (OTS 2 method call *onASubscribe*).

This order has the type *acSubscriptionAnyType*. The data to be provided is already determined by the selection of the service endpoint in MDM and it is therefore not specified in the order.

For the order, please see also [OTS2] chapter 7.6.7.1 and 7.6.7.2.

4.4.4.3 Data delivery

The data supplier regularly delivers his data after receipt of the order (OTS 2 method *ASnippet*). The data supplier is responsible for the transmission of data packets to MDM in the agreed delivery frequency (pursuant to the configuration of the data services in the metadata directory).

The data packets must use the type *acDataAnyType*. The data therein is expected to be a DATEX II package, or (recommended) a BASE64-encoded and GZIP-compressed DATEX II binary package (see chapter 4.4.3).

For the data delivery, please see also [OTS2] chapter 7.6.7.3 and 7.6.7.4.

4.4.4.4 Connection termination

An existing connection can be again terminated by either side (OTS 2 methods *AUnSubscribe* to terminate the order or data delivery and, then, *AUntie* to terminate a connection).

Connections will be terminated by the MDM only if a publication or a subscription is set out in an inactive mode from the administration (metadata directory), or if the MDM server is shut down (e.g. for maintenance purposes).

In the case of a connection termination (OTS 2 method call *onAUntied*), the client receives, in the first case, the reason "MDM Service Disabled" in the field "reason" and, in the second case, the reason "MDM Server Shutdown".

If the data supplier wants to terminate the connection, it will have to provide an appropriate justification in the "reason" field, e.g. "MDM Client Shutdown" or "MDM Client Restart" to create detailed log messages.

To establish a connection, see also [OTS2] chapter 7.6.5 and 7.6.8.

4.4.5 OTS 2 Subscribe

When using the OTS 2 protocol, the data client takes the role of an OTS 2 subscriber.

Depending on the MDM subscription, a separate connection has to be established to the designated service endpoint (OTS 2 method *ATie*).

The URL of the service endpoint is structured as follows:

```
soap.tls://<BASt-MDM-Broker-Server>/BASt-MDM-OTS2-  
DeliveryService/sub/<subscriptionID>
```

Example:

```
soap.tls://service.mac.mdm-portal.de/BASt-MDM-OTS2-  
DeliveryService/sub/2035000
```

The connections are usually maintained - as long as possible - permanently (and not rebuilt, e.g. every minute). The authentication takes place only when establishing a connection (see below).

The following Figure 14 shows an example of a sequence with a connection establishment, order by the client, data delivery by the MDM (here are just two deliveries indicated) and a connection terminated by the client (a disconnection in the reverse direction by the MDM would also be possible). The data client (client) is located on the left side.

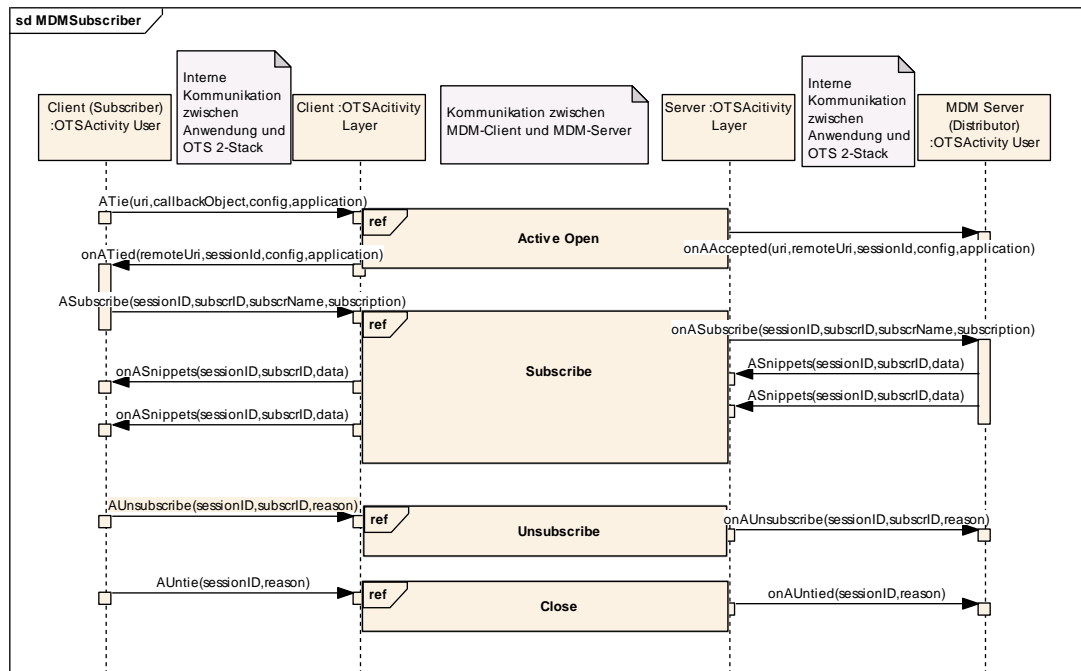


Figure 14: Sequence diagram OTS 2 communication between data clients and MDM

4.4.5.1 Connection establishment

By using its machine certificate, the data client system has to establish a TLS connection in the direction of MDM with the OTS 2 protocol connection "SOAP/HTTP with TLS encryption" (OTS 2 method *ATie*, "soap.tls:" is in the URL schema field).

For the implementation of the certificate transmission, please refer to chapter 8.1 (Use of Client Certificates for HTTPS Communication with Servers That Require a Certificate-based Authentication).

The data client must set the following special features in the configuration information:

- "a_subscriber=1"" (indicates that it is a data client)
- "a_c_datex_any_mdm=1" (encodes the specific conditions for the use of OTS 2 protocol in MDM)
- "t_targetURI" with the complete target URI, e.g. "t_targetURI=soap.tls://service.mac.mdm-portal.de/BASSt-MDM-OTS2-DeliveryService/sub/2035000" (is internally required by the MDM server OTS 2)

To establish a connection, see also [OTS2] chapter 7.6.4.1 and 7.6.4.2.

4.4.5.2 Order

Pursuant to OTS 2 protocol, the data client must place his order (OTS 2 method call *ASubscribe*) after a successful connection establishment (OTS 2 method call *onATied*).

The order has the type *acSubscriptionAnyType*. The required data is already determined by the selection of the service endpoint in MDM and it is therefore not specified in the order.

It is recommended to enter the MDM publication name into the field "subscrName" of the OTS 2 order and the MDM publication ID into the field "topic". All other optional fields in the OTS 2 order shall not apply.

For the order, please see also [OTS2] chapter 7.6.7.1 and 7.6.7.2.

4.4.5.3 Data delivery

After having placed the order, the data client regularly receives the MDM data (OTS 2 method call *onASnippet*). MDM shall always provide new data packets in the context of a push method as soon as the data arrives to MDM from the data supplier.

The data packets are of the type *acDataAnyType*. The data is supplied as BASE64-encoded and GZIP-compressed DATEX II binary package (see chapter 4.4.3).

For the data delivery, please see also [OTS2] chapter 7.6.7.3 and 7.6.7.4.

4.4.5.4 Connection termination

An existing connection can be terminated by either side (OTS 2 methods *AUnSubscribe* to terminate the order or delivery data and, then, *AUntie* to terminate a connection).

Connections will be terminated by the MDM only if a publication or a subscription is set out in an inactive mode from the administration (metadata directory), or if the MDM server is shut down (e.g. for maintenance purposes).

In the case of a connection termination (OTS 2 method call *onAUntied*), the client receives, in the first case, the reason "MDM Service Disabled" in the field "reason" and, in the second case, the reason "MDM Server Shutdown".

If the data client wants to terminate the connection, it will have to provide an appropriate justification in the "reason" field, e.g. "MDM Client Shutdown" or "MDM Client Restart" to create detailed log messages.

To terminate a connection, see also [OTS2] chapter 7.6.5 and 7.6.8.

5 Certificate-based M2M Communication

The security component of the MDM platform requires a certificate-based data exchange between the data supplier system and the platform, on the one side, and between the platform and the data client system, on the other.

This chapter begins with an overview of the functions of the security component and, then, describes the steps to be taken by the data suppliers and the data clients to request certificates and set them up for M2M communication.

The certificate is created following a request and then sent to the data supplier/data client by e-mail. The password that is required for signature is sent by fax.

The data supplier system/data client system must finally integrate the certificate into their IT infrastructure, so that the data exchange with the MDM platform can be authenticated.

5.1 Tasks of the Security Component

The security component is responsible for the realization of the safety aspects of the MDM platform. This includes, in particular, the authentication of data supplier systems and data client systems, which want to communicate with the MDM platform.

Before the data packets arriving at the MDM platform can be accepted, their origin must be checked. This includes the authentication of the data supplier system that is associated with the data packet using a digital certificate. Each data supplier system must have a valid certificate to be used for login at the platform. The security component authenticates the certificate sent by the data supplier system within the MDM platform.

Before a data packet can be sent to a data client system, the identity of this data client system needs to be checked. Each data client system must authenticate itself to the MDM platform using a digital certificate. The security component authenticates the certificate sent by the data client system within the MDM platform.

The confidentiality of communications between the data supplier system and the MDM platform, on the one hand, and the MDM platform and the data client system, on the other hand, must be ensured by an exclusive use of an SSL/TLS transport encryption.

The security component requires standards-compliant [X.509v3] certificates for authentication; see also [PKI]. The certificates must be technically involved in the HTTPS connection to the data client and data supplier systems via a client-side, certificate-based connection establishment. The presented certificates are checked for validity and whether they are blocked or not.

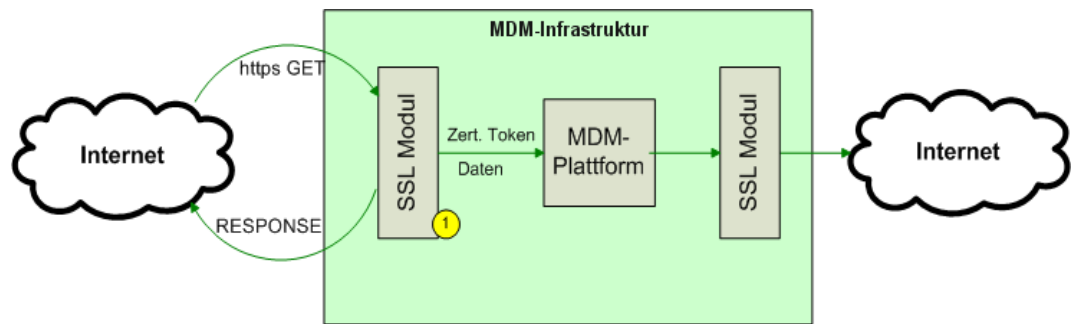


Figure 15: Overview of the security architecture

The SSL module 1 in Figure 15 sends a certificate request to the sender for predefined URLs, checks the validity of the obtained certificate and then verifies whether it is blocked or not. Afterwards, it forwards the certificate to the security component of the MDM platform.

5.2 Note on Server Name Indication

The SSL implementation JSSE in Java 6, used in the MDM platform, supports no Server Name Indication (SNI), see:

http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6985179

This means that data suppliers for the client pull method and data clients for the publisher push method cannot use any virtual server for M2M communication. Each registered machine can represent only a unique IP address.

5.3 Applying for a Machine Certificate

The operator of the MDM platform mediates between the data supplier or data client systems and the certificate issuer. Therefore, data suppliers and data clients apply - when registering - for one or multiple machine certificates via the administration GUI of the MDM platform. The certificate is however sent to them by the certificate-issuing organization and not by the operator of the MDM platform.

To request a machine certificate, you must already be registered on the MDM platform with your organization.

How to apply for a machine certificate on the MDM platform is described in [BHB].

5.4 Installing a Machine Certificate and Issuer Certificate

Note: The installation for IIS is described in chapter 9.1.

In the Apache Web server, integrate the machine certificate as follows:

```
SSLCertificateFile /usr/local/apache2/conf/ssl.crt/server.crt
```

Enter the associated private key as follows:

```
SSLCertificateKeyFile /usr/local/apache2/conf/ssl.crt/server.key
```

In addition, you must install the issuer certificate on the web server:

```
SSLCACertificateFile /usr/local/apache2/conf/ssl.crt/ca-bundle-  
client.crt
```

The certificate is encrypted by using the key with the password that has been sent to you via fax. Use the password to decrypt.

For more information on these directives, please see the mod_ssl documentation:

http://httpd.apache.org/docs/current/mod/mod_ssl.html#sslcertificatefile

http://httpd.apache.org/docs/current/mod/mod_ssl.html#sslcacertificatefile

Note: If you get the machine certificate and the issuer certificate within a common p12 file, you must extract both certificates from this file and then install them. The relevant instructions are provided in chapter 9.1

5.5 Authentication of the MDM Platform as Web Client

If the MDM platform acts as a web client in the M2M communication, it will then authenticate with its server certificate, provided that the web server has enabled this option on the data supplier or data client side. Data supplier and data client systems should enable this option and verify the certificate to determine that the requests were actually disposed of by the MDM platform.

The CA certificates required for verification can be downloaded from <http://hilfe.mdm->

portal.de/fileadmin/user_upload/Dokumente/Hilfeseite/MDM_CA-Cert.zip and must be stored in the data supplier or data client systems.

Note: Do not use the MDM server certificate for verification. It is changed on a regular basis.

5.6 Authentication of Data Supplier/Data Client Web Clients

If the data supplier or data client systems act as a web client in the M2M communication, the web client must then authenticate to the MDM platform by using its machine certificate. The platform will accept requests only from systems that are registered in the metadata directory. Based on the certificate, the machine can be assigned to the organization. Furthermore, it can be checked whether the organization is the owner of the publication or subscription for which data exchange is to take place.

It should be noted that the server certificate of the MDM has been issued to the MDM broker by the MDM's own CAs. Therefore, the server certificate should be checked on the client side against the CA certificates, which can be downloaded under: http://hilfe.mdm-portal.de/fileadmin/user_upload/Dokumente/Hilfeseite/MDM_CA-Cert.zip

In the Annex A, you will find a Java code example that demonstrates the use of client certificates both for the communication with a SOAP web service as well as for a web server via HTTPS.

6 Exceptions and Error Messages

6.1 Exception - Unchanged Data

If a DATEX II client pull request uses the header field "If-Modified-Since", and if there are no more recent data packets than those already gathered, an HTTP status code 304 = "Not-Modified" will be generated. The same shall apply if no data is not yet available.

6.2 Error Messages with SOAP Requests

Error messages with SOAP requests are reported as SOAP faults.

Here, the error message in the "faultstring" of the following SOAP response is sent:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>S:Server</faultcode>
      <faultstring>...</faultstring>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

6.3 Error Messages with HTTPS Requests

If no SOAP request is available, the error message will be sent with HTTP 503 and content type "text".

6.4 Error Handling in the Context of OTS 2 Protocol

Different error situations, which may occur for an OTS 2 client, are hereinafter described. For more information on the OTS 2 standard errors (e.g. parameters included) can be consulted in [OTS2].

6.4.1 Session setup

If the required certificates are missing or invalid, an error of the type 1301 (*TConnect* failed) will be initiated on the client side. This is done either by the method call *onError* or *onRemoteError*, depending on whether the error is already detected on the client or on the server side. In the field "reason" of the error, "Certificate error" is given as reason.

If the certificates are inappropriate (valid, but e.g. not suitable for the intended publication/subscription), an error 1301 (*TConnect* failed) will

also be triggered. The field "reason" will then include "Certificate inappropriate".

If you try to use an incorrect protocol connection (not soap.tls), an error of the type 1001 (invalid URI) will be triggered by *onError*.

If the feature "t_targetURI" is missing in the configuration information, an error of the type 5102 (rejected) will be triggered by *onRemoteError*. The field "reason" will include "Feature t_targetURI required".

If the feature "a_c_datex_any_mdm" is missing in the configuration information, an error of the type 3301 (*ATie* failed) will be triggered by *onRemoteError*. The field "reason" will include "Feature a_c_datex_any_mdm required".

If the server does not accept any connections or is not available, an error of the type 1301 (*TConnect* failed) will be triggered. The field "reason" will include "Unavailable URI".

6.4.2 Order

If the order is of the wrong type (not *acSubscriptionAnyType*), an error of the type 8709 (Subscription: invalid parameter) will be triggered. In the field "par", you find the invalid type.

6.4.3 Data delivery

If a data delivery to the MDM is of the wrong type (not *acDataAnyType* or not processable content), an MDM-specific error of the type 10001 will be triggered.

6.4.4 General

If there is no communication for a longer period of time and the connection is in a timeout or if the connection is unexpectedly interrupted for other reasons, an error of the type 1003 (Transport connection lost) will be triggered.

If a data transfer fails, an error 1501 (*TSendData* failed) will be triggered.

In both cases, the connection must be ended and, if necessary, a re-establishment of the connection must be attempted.

An internal error of the MDM broker component is displayed with the MDM-specific error of the type 10002.

7 Examples

7.1 HTTPS Interface

7.1.1 Data Supplier Client Pull HTTPS (Container)

The publication ID must be provided as a parameter in the URL.

Request:

```
GET https://<DG-Server>/<Context>?publicationID=2053008
content-type: text/plain
accept-encoding: identity,gzip
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<container xmlns="http://ws.bast.de/container/TrafficDataService"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2002/07/utility"
xmlns:ns3="http://www.w3.org/2000/09/xmldsig#">
  <header>
    <Identifier>
      <publicationId>2053008</publicationId>
    </Identifier>
  </header>
  <body>
    <binary id="test-id-bin" type="hexBinary">
      &lt;![CDATA[]]&gt;
    </binary>
    <xml schema="test-schema" id="test-id-xml">
      <n4:musterDatenRoot>
        <n4:trafficData origin="home" />
      </n4:musterDatenRoot>
    </xml>
  </body>
</container>
```

7.1.2 Data Supplier Publisher Push HTTPS (Container)

The publication ID is included in the XML data.

```
<?xml version='1.0' encoding='UTF-8'?>
<ns3:containerRootElementEl xmlns="http://www.w3.org/2000/09/xmldsig#"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2002/07/utility"
xmlns:ns3="http://ws.bast.de/container/TrafficDataService">
  <ns3:header>
    <ns3:Identifier>
      <ns3:publicationId>12345</ns3:publicationId>
    </ns3:Identifier>
  </ns3:header>
  <ns3:body>
    <ns3:binary id="test-id-bin" type="hexBinary">
```

```

dGVzdC10ZXh0&#xD;.
</ns3:binary>
<ns3:xml schema="test-schema" id="test-id-xml">
  <n4:musterDatenRoot>
    <n4:trafficData origin="home"/>
  </n4:musterDatenRoot>
</ns3:xml>
</ns3:body>
</ns3:containerRootElementEl>

```

7.1.3 Data client Client Pull HTTPS (DATEX II)

The request must contain no more data. The subscription ID must be provided in the path of the URL and also as a parameter.

```

GET https://service.mac.mdm-portal.de/BASSt-MDM-
Interface/srv/2000000/clientPullService?subscriptionID=2000000

```

7.1.4 Data client Client Pull HTTPS (Container)

The request must contain no more data. The subscription ID must be provided as a parameter in the URL.

Request:

```

GET https://service.mac.mdm-portal.de/BASSt-MDM-
Interface/srv/container/v1.0?subscriptionID=2000000

```

7.2 SOAP Interface

7.2.1 Data Supplier Publisher Push SOAP (DATEX II)

The publication ID must be provided in the path of the URL.

```

https://service.mac.mdm-portal.de/BASSt-MDM-
Interface/srv/2000002/supplierPushService

```

```

<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <d2LogicalModel xmlns="http://datex2.eu/schema/2/2_0"
modelBaseVersion="2">
      <exchange>

<subscriptionReference>subscriptionReference</subscriptionReference>
        <supplierIdentification>
          <country>de</country>
          <nationalIdentifier>TestClient</nationalIdentifier>
          <internationalIdentifierExtension/>
        </supplierIdentification>
      </exchange>
      <payloadPublication xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:type="SituationPublication" lang="DE">

```



```

    <feedDescription>
      <values>
        <value lang="DE">test-test</value>
      </values>
    </feedDescription>
    <feedType>feedType</feedType>
    <publicationTime>2011-03-
02T10:36:34.336+01:00</publicationTime>
    <publicationCreator>
      <country>de</country>
      <nationalIdentifier>TestClient</nationalIdentifier>
      <internationalIdentifierExtension/>
    </publicationCreator>
    <situation version="0.1" id="GUID-Mattst-1299058594339">
      <overallSeverity>none</overallSeverity>
      <headerInformation>
        <areaOfInterest>regional</areaOfInterest>
      </headerInformation>
      <situationRecord xsi:type="AnimalPresenceObstruction">
        <generalPublicComment>
          <comment>
            <values>
              <value lang="DE"></value>
            </values>
          </comment>
        </generalPublicComment>
      </situationRecord>
    </situation>
  </payloadPublication>
</d2LogicalModel>
</S:Body>
</S:Envelope>

```

7.2.2 Data Supplier Client Push SOAP (Container)

The publication ID is included in the XML data.

```

<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns3:containerRootElementEl
xmlns="http://www.w3.org/2000/09/xmldsig#"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2002/07/utility"
xmlns:ns3="http://ws.bast.de/container/TrafficDataService">
      <ns3:header>
        <ns3:Identifier>
          <ns3:publicationId>12345</ns3:publicationId>
        </ns3:Identifier>
      </ns3:header>
      <ns3:body>
        <ns3:binary id="test-id-bin"
type="hexBinary">dGVzdC10ZXh0&#xD;.</ns3:binary>
        <ns3:xmlschema="test-schema" id="test-id-xml"/>
      </ns3:body>
    </ns3:containerRootElementEl>
  </S:Body>
</S:Envelope>

```

```

    </ns3:containerRootElement>
  </S:Body>
</S:Envelope>

```

7.2.3 Data client Client Pull SOAP (DATEX II)

The request must contain no more data. The subscription ID must be provided in the path of the URL.

```

https://service.mac.mdm-portal.de/BASSt-MDM-
Interface/srv/2000000/clientPullService

```

```

<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body />
</S:Envelope>

```

7.2.4 Data client Client Pull SOAP (Container)

The subscription ID is included in the XML data.

```

<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns3:pullContainerDataClientRequestEl
      xmlns="http://www.w3.org/2000/09/xmldsig#"
      xmlns:ns2="http://schemas.xmlsoap.org/ws/2002/07/utility"
      xmlns:ns3="http://ws.bast.de/container/TrafficDataService">
      <ns3:subscriptionId>2000000</ns3:subscriptionId>
    </ns3:pullContainerDataClientRequestEl>
  </S:Body>
</S:Envelope>

```

7.2.5 Data client Publisher Push SOAP (DATEX II)

Expected response from the data client system:

```

<D2LogicalModel:d2LogicalModel modelBaseVersion="2"
  xsi:schemaLocation="http://datex2.eu/schema/2/2_0/
  DATEXIIISchema_2_2_0.xsd"
  xmlns:D2LogicalModel="http://datex2.eu/schema/2/2_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <D2LogicalModel:exchange>
    <D2LogicalModel:response>acknowledge</D2LogicalModel:response>
  </D2LogicalModel:exchange>
  ...
</D2LogicalModel:d2LogicalModel>

```

7.3 OTS 2 Interface

7.3.1 Protocol example SOAP

The procedure for using the OTS 2 protocol is exemplified in the communication protocol of a data client.

For a data supplier, the process is essentially identical, except that the order (*aSubscribe*) and the data delivery (*aSnippets*) are transmitted in the reverse direction (the order is received over *tGetR* and the data delivery is sent over *tSend* instead of vice versa, as shown in the example).

7.3.1.1 Connection establishment

The subscription ID must be provided in the path of the URL and, additionally, the URL as a configuration parameter *t_targetURI*.

Request (*tConnect*):

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <tConnect xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <clientId>ff97c2b4c0a8013800d902546789ad4c</clientId>
      <username/>
      <password/>
      <localTransportId>1</localTransportId>
      <timeout>100000</timeout>
      <neededConfig version="m_configListClient_C3X">
        <cfgs>
          <cfg>
            <name>t_targetURI= soap.tls://service.mac.mdm-portal.de/BAST-
MDM-OTS2-DeliveryService/sub/2035000</name>
            <min>0</min>
            <max>0</max>
          </cfg>
        </cfgs>
      </neededConfig>
    </tConnect>
  </S:Body>
</S:Envelope>
```

Response (*tConnectR*):

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <tConnectR xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <transportId>
```

```

    <clientPart>1</clientPart>
    <serverPart>27</serverPart>
  </transportId>
  <config version="m_configListServer_S3A"/>
</tConnectR>
</env:Body>
</env:Envelope>

```

Request (*tGet*, *tGetR* Response s.u.):

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <tGet xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <transportId>
        <clientPart>1</clientPart>
        <serverPart>27</serverPart>
      </transportId>
    </tGet>
  </S:Body>
</S:Envelope>

```

Request (*sOpen* via *tSend*, *tSendR* Response is empty):

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <tSend xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <transportId>
        <clientPart>1</clientPart>
        <serverPart>27</serverPart>
      </transportId>
      <data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="sOpenType">
        <neededConfig version="m_configListClient_C2X"/>
      </data>
    </tSend>
  </S:Body>
</S:Envelope>

```

Response (*sOpenResponse* via *tGetR*, *tGet* Request see above):

```

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <tGetR xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <ds>
        <tSend>

```

```

        <transportId>
          <clientPart>1</clientPart>
          <serverPart>27</serverPart>
        </transportId>
        <data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="sOpenResponseType">
          <sessionId>266419a0-1d02-11e1-a7c2-000c294483b2</sessionId>
          <config version="m_configListServer_S2A"/>
        </data>
      </tSend>
    </ds>
  </tGetR>
</env:Body>
</env:Envelope>

```

Request (*tGet*, *tGetR* Response see below):

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <tGet xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <transportId>
        <clientPart>1</clientPart>
        <serverPart>27</serverPart>
      </transportId>
    </tGet>
  </S:Body>
</S:Envelope>

```

Request (*aTie* via *sMsg* via *tSend*, *tSendR* Response is empty):

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <tSend xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <transportId>
        <clientPart>1</clientPart>
        <serverPart>27</serverPart>
      </transportId>
      <data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="sMsgType">
        <msg xsi:type="aTieType">
          <application xsi:type="acApplicationType">
            <appVersion>OTS2TestClient_V_1.0.0</appVersion>
          </application>
          <neededConfig version="configListCounterPart">
            <cfigs>
              <cfig>
                <name>s_layer</name>
                <min>1</min>

```

```

        <max>1</max>
      </cfg>
    <cfg>
      <name>a_distributor_sub</name>
      <min>1</min>
      <max>1</max>
    </cfg>
    <cfg>
      <name>a_subscriber</name>
      <min>0</min>
      <max>0</max>
    </cfg>
    <cfg>
      <name>a_c_datex_any_mdm</name>
      <min>0</min>
      <max>0</max>
    </cfg>
    <cfg>
      <name>t_layer</name>
      <min>1</min>
      <max>1</max>
    </cfg>
    <cfg>
      <name>a_layer</name>
      <min>1</min>
      <max>1</max>
    </cfg>
  </cfigs>
</neededConfig>
</msg>
</data>
</tSend>
</S:Body>
</S:Envelope>

```

Response (*aTieResponse* via *sMsg* via *tGetR*, *tGet* Request see above):

```

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <tGetR xmlns="http://opentrafficsystems.org/OTS2"
      xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
      konsortium.de/OCIT-I_OITD">
      <ds>
        <tSend>
          <transportId>
            <clientPart>1</clientPart>
            <serverPart>27</serverPart>
          </transportId>
          <data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="sMsgType">
            <msg xsi:type="aTieResponseType">
              <application xsi:type="acApplicationType">

```

```

<appVersion>GUI_OTS2_ActivityLayer_SRVTest_V_1.0.0</appVersion>
  </application>
  <config version="configListCounterPart">
    <cfgs>
      <cfg>
        <name>s_layer</name>
        <min>1</min>
        <max>1</max>
      </cfg>
      <cfg>
        <name>a_distributor_sub</name>
        <min>1</min>
        <max>1</max>
      </cfg>
      <cfg>
        <name>a_subscriber</name>
        <min>0</min>
        <max>0</max>
      </cfg>
      <cfg>
        <name>a_c_datex_any_mdm</name>
        <min>0</min>
        <max>0</max>
      </cfg>
      <cfg>
        <name>t_layer</name>
        <min>1</min>
        <max>1</max>
      </cfg>
      <cfg>
        <name>a_layer</name>
        <min>1</min>
        <max>1</max>
      </cfg>
    </cfgs>
  </config>
</msg>
</data>
</tSend>
</ds>
</tGetR>
</env:Body>
</env:Envelope>

```

7.3.1.2 Data order

In the field "topic", it is recommended to enter the MDM publication ID.

Request (*aSubscribe* via *sMsg* via *tSend*, *tSendR* Response is empty):

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>

```

```

<tSend xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
  <transportId>
    <clientPart>1</clientPart>
    <serverPart>27</serverPart>
  </transportId>
  <data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="sMsgType">
    <msg xsi:type="aSubscribeType">
      <subscrId>1322843092</subscrId>
      <subscrName>OTS2TestClient</subscrName>
      <subscription xsi:type="acSubscriptionAnyType">
        <topic>2035000</topic>
      </subscription>
    </msg>
  </data>
</tSend>
</S:Body>
</S:Envelope>

```

7.3.1.3 Data delivery

Request (*tGet*, *tGetR* Response see below):

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <tGet xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <transportId>
        <clientPart>1</clientPart>
        <serverPart>27</serverPart>
      </transportId>
    </tGet>
  </S:Body>
</S:Envelope>

```

Response (*aSnippets* via *sMsg* via *tGetR*, *tGet* Request see above):

```

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <tGetR xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <ds>
        <tSend>
          <transportId>
            <clientPart>1</clientPart>
            <serverPart>27</serverPart>
          </transportId>
          <data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="sMsgType">

```



```

        <msg xsi:type="aSnippetsType">
          <subscrId>1322843092</subscrId>
          <data xsi:type="acDataAnyType">
            <dataAny xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:type="xs:string">&lt;binary
type=&quot;base64BinaryDatex2Gzip&quot;
id=&quot;0&quot;&gt;H4sIAAAAAAADTdCZYkOa5D0S3Z5Dbsf2PJ+XTZ/U//qswIdx
skigQB8H6f4/f7Xfdz7dd3/I59v87ffpzPfZy/69uf+zvO...</dataAny>
          </data>
        </msg>
      </data>
    </tSend>
  </ds>
</tGetR>
</env:Body>
</env:Envelope>

```

7.3.1.4 Cancellation

Request (*aUnsubscribe* via *sMsg* via *tSend*, *tSendR* Response is empty):

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <tSend xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <transportId>
        <clientPart>1</clientPart>
        <serverPart>27</serverPart>
      </transportId>
      <data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="sMsgType">
        <msg xsi:type="aUnsubscribeType">
          <subscrId>1322843092</subscrId>
          <reason>OTS2TestClient closes</reason>
        </msg>
      </data>
    </tSend>
  </S:Body>
</S:Envelope>

```

7.3.1.5 Connection termination

Request (*sClose* via *tSend*, *tSendR* Response is empty):

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <tSend xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <transportId>
        <clientPart>1</clientPart>
        <serverPart>27</serverPart>
      </transportId>

```

```

    <data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="sCloseType">
      <sessionId>266419a0-1d02-11e1-a7c2-000c294483b2</sessionId>
      <reason>End TestClient</reason>
    </data>
  </tSend>
</S:Body>
</S:Envelope>

```

Response (*sCloseResponse* via *tSend* via *tGetR*, *tGet* Request not shown):

```

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <tGetR xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <ds>
        <tSend>
          <transportId>
            <clientPart>1</clientPart>
            <serverPart>27</serverPart>
          </transportId>
          <data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="sCloseResponseType">
            <sessionId>266419a0-1d02-11e1-a7c2-000c294483b2</sessionId>
          </data>
        </tSend>
      </ds>
    </tGetR>
  </env:Body>
</env:Envelope>

```

Request (*tDisconnect*):

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <tDisconnect xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <transportId>
        <clientPart>1</clientPart>
        <serverPart>27</serverPart>
      </transportId>
      <reason>close</reason>
    </tDisconnect>
  </S:Body>
</S:Envelope>

```

Response (*tDisconnectR*):

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <tDisconnectR xmlns="http://opentrafficsystems.org/OTS2"
xmlns:ns2="http://datex2.eu/schema/2_0RC2/2_0" xmlns:ns3="http://otec-
konsortium.de/OCIT-I_OITD">
      <rTransportId>
        <clientPart>1</clientPart>
        <serverPart>27</serverPart>
      </rTransportId>
    </tDisconnectR>
  </env:Body>
</env:Envelope>
```

8 Annex A

8.1 Use of Client Certificates for HTTPS Communication with Servers That Require a Certificate-based Authentication

```
package com.ottensoftware.bast.http.ssl;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.security.KeyStore;
import java.security.SecureRandom;
import java.util.List;
import java.util.Map;
import java.util.zip.GZIPInputStream;
import java.util.zip.Inflater;
import java.util.zip.InflaterInputStream;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.KeyManager;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;

/*
 * This class demonstrates the use of client certificates for the communication
 * via HTTPS with servers requesting a certificate-based authentication.
 *
 *
 * Requirements:
 * The private key and the client certificate are available as PKCS12 file.
 * The password of the key file is known.
 *
 * The certificate of the server(s), with which connections are to be
 * established, are available as Java keystore file(1)
 *
 * (1) You can determine the certificates of the MDM platform using a browser,
 * or extract them from the PKCS-12 file using openssl.
 *
 * You create your own truststore file using:
 * <JAVA_HOME>\bin\keytool -import -trustcacerts -alias "own alias for the
 * holder of the certificate"
 * -file "Path to the certificate to be imported" keystore "file name of the
 * truststore file"
```

```

*
* Example: C:\Programme\Java\jdk1.6.0_22\bin\keytool -import -trustcacerts -
alias BUCServiceManagementCertificateAuthority
* -file ./trustcerts/BUCServiceManagementCertificateAuthority.crt -keystore
mytruststore.jks
*
* Enter the keystore password:          <With the first certificate, set your own
password. Use this password for all subsequent imports>
* Enter the password again: <Password repetition>
* The following is the output of the certificate to be imported:
* Owner: EMAILADDRESS=dalbers@materna.de, CN=BUC Service Management SSL CA,
OU=BU
* C Service Management, O=MATERNA GmbH, ST=North Rhine-Westphalia, C=DE
* Originator: EMAILADDRESS=dalbers@materna.de, CN=BUC Service Management
Certificate Authority, OU=BUC Service Management, O=MATERNA GmbH, L=Dortmund,
ST=North Rhine-Westphalia, C=DE
* Serial number: 9
* Valid from: Thu Feb 03 12:00:52 CET 2011 until: Sat Feb 02 12:00:52 CET 2013
* Digital thumbprint of the certificate:
*           MD5:  0A:DA:6A:FB:5B:AD:F7:31:77:BD:BB:8E:4C:C2:58:08
*           SHA1:
E8:4A:92:C2:05:65:15:78:E9:B6:C1:D7:53:FC:60:48:F4:E3:25:AB
*           Signature algorithm name: SHA1withRSA
*           Version: 3
*           Enhancements:
*           #1: ObjectId: 2.5.29.19 Criticality=true
*           BasicConstraints:[
*           CA:true
*           PathLen:0
*           ]
*           #2: ObjectId: 2.5.29.15
Criticality=false
*           KeyUsage [
*           Key_CertSign
*           Crl_Sign
*           ]
*           #3: ObjectId: 2.5.29.14
Criticality=false
*           SubjectKeyIdentifier [
*           KeyIdentifier [0000: 32 60 8E F6
A7 CE 7B 92   CB 28 0F AB 05 05 96 9A  2`.....(.....0010: A2 95 CD 9A
....]]
*           #4: ObjectId: 2.5.29.31
Criticality=false
*           CRLDistributionPoints [
*           [DistributionPoint:      [URIName:
http://ca.mbucqa.de/sslca.crl]
*           ReasonFlags:    key compromise
*           CA compromise]
*           #5:
ObjectId: 2.16.840.1.113730.1.1 Criticality=false

```

```

*
    NetscapeCertType [
*
SSL CA
*
*
*
*
*
ObjectId: 2.5.29.35 Criticality=false
*
    AuthorityKeyIdentifier [
*
        KeyIdentifier [0000: D9 D4 57 C9 3E 64 CC EA    37 18 37 5D EB C1 6B 89
..W.>d..7.7]..k.0010: FB 4B 89 B0
*
*
*
*
*
*
* Do you trust this certificate? [No]: Yes
* Certificate was added to keystore.
*
*
*
* @author Frank Rossol
*
*/
public class TestClientCertHttpDateX2 {

    /**
     * returns an SSL connection by using client certificates to the specified
URL
     * @param urlString URL to which the connection is to be established
     * @return HttpURLConnection Https connection
     */
    private HttpURLConnection getSecureConnection(String urlString) {
        try {
            // Password of the P12 file
            String password="myKeyPass";
            // Password of the truststore file that contains the
certificates of the MDM platform
            String trustPassword="myTrustStorePass";
            // Type of keystore file
            String keyStoreType = "pkcs12";
            // Type of truststore file
            String trustStoreType = "JKS";

            // Load p12 file
            KeyStore keyStore = KeyStore.getInstance(keyStoreType);
            keyStore.load(new
FileInputStream("C:/MDMClient/java/SSL/privkey/myPrivateKey.p12"),
password.toCharArray());
            KeyManagerFactory kmf =KeyManagerFactory.getInstance("SunX509",
"SunJSSE");
            kmf.init(keyStore, password.toCharArray());

            // Load truststore
            KeyStore trustStore = KeyStore.getInstance(trustStoreType);

```

```

        trustStore.load(new
FileInputStream("C:/MDMClient/java/SSL/myTrustStore.jks"),
trustPasswort.toCharArray());
        TrustManagerFactory tmf = TrustManagerFactory.getInstance("PKIX");
        tmf.init(trustStore);
        TrustManager tms [] =tmf.getTrustManagers();

        SSLContext sslcontext = SSLContext.getInstance("TLS");
        KeyManager[] km = kmf.getKeyManagers();
        SecureRandom random = new SecureRandom();

        // Initialize SSL context with keystore and truststore
instances
        sslcontext.init(km, tms, random);
        SSLSocketFactory sslfactory =sslcontext.getSocketFactory();
        HttpURLConnection.setDefaultSSLSocketFactory(sslfactory);

        URL url = new URL(urlString);
        HttpURLConnection con  =(HttpURLConnection)
url.openConnection();
        con.setSSLSocketFactory(sslfactory);

        HostnameVerifier hostnameVerifier = new HostnameVerifier() {
            // Here, install own name check of the server
            @Override
            public boolean verify(String hostname, SSLSession
session) {

                return true;

            }

        };
        con.setHostnameVerifier(hostnameVerifier );
        con.setDefaultUseCaches(false);
        return con;
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
    return null;
}

/**
 * This function calls up DATEX2 data from platform
 */
private void testClientPullHttpsDatexII() {
    String subscriptionId = "123456";
    // Server name under which the MDM platform is accessible.
    String mdmHost="service.mac.mdm-portal.de";
    String url = "https://" + mdmHost +"/BAST-MDM-
Interface/srv/"+subscriptionId+"/clientPullService?subscriptionId="+
subscriptionId;
    HttpURLConnection con  = null;
    try {

```

```

        // Connect to server
        con = getSecureConnection(url);
        con.setRequestMethod("GET");
        con.setRequestProperty("Accept-Encoding", "gzip");
        con.setUseCaches (false);
        con.setDoInput(true);
        con.setDoOutput(true);
        con.connect();

        // Connection status
        int responseCode = con.getResponseCode() ;
        // Data retrieved successfully (HTTP-Server Code)
        if (responseCode == 200) {
            System.out.println("Get successful");
        }
        else if (responseCode == 400) {
            System.out.println("No subscription parameter or no data has been
given");
        }
        else if (responseCode == 404) {
            System.out.println("Subscription parameter could not be assigned or
the subscription is no longer valid");
        }
        else {
            System.out.println("Error: " + responseCode);
            // Content of error
            try {
                InputStream is = con.getErrorStream();
                BufferedReader rd = new BufferedReader(new
InputStreamReader(is));
                String line;
                StringBuffer response = new StringBuffer();
                while((line = rd.readLine()) != null) {
                    response.append(line);
                    response.append('\r');
                }
                rd.close();
                System.out.println(response.toString());
            }
            catch (Exception ex) {
                ex.printStackTrace();
            }
        }

        InputStream is = null;
        // How does the transmission of content occur?
        String encoding = con.getContentEncoding();

        // Generate the required wrapper based on encoding
        if (encoding != null && encoding.equalsIgnoreCase("gzip")) {
            is = new GZIPInputStream(con.getInputStream());
        }
        else if (encoding != null && encoding.equalsIgnoreCase("deflate"))
    {

```



```

        is = new InflaterInputStream(con.getInputStream(), new
Inflater(true));
    }
    else {
        is = con.getInputStream();
    }

    // Read results
    BufferedReader rd = new BufferedReader(new InputStreamReader(is));
    String line;
    StringBuffer response = new StringBuffer();
    while((line = rd.readLine()) != null) {
        response.append(line);
        response.append('\r');
    }
    rd.close();
    System.out.println(response.toString());
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
    finally {
        if(con != null) {
            con.disconnect();
        }
    }
}

/**
 * @param args
 */
public static void main(String[] args) {
    TestClientCertHttpDatex2 tc = new TestClientCertHttpDatex2();
    tc.testClientPullHttpsDatexII();
}
}

```

8.2 Use of Client Certificates for the Communication with SOAP Web Services That Require a Certificate-based Authentication

```
package com.ottensoftware.bast.soap.ssl;

import java.net.URL;

import javax.xml.namespace.QName;

import eu.datex2.schema._2._2_0.D2LogicalModel;
import eu.datex2.wsdl.clientpull._2_0.client.ClientPullService;
import eu.datex2.wsdl.clientpull._2_0.client.ClientPullService_Service;
/*
 * This class demonstrates the use of client certificates for the communication
 * with SOAP web services that require a certificate-based authentication.
 *
 *
 * Requirements:
 * The private key and the client certificate are available as PKCS12 file.
 * The password of the key file is known.
 *
 * The certificate of the server(s), with which connections are to be
 * established, are available as Java keystore file(1)
 *
 * (1) You can determine the certificates of the MDM platform using a browser,
 * or extract them from the PKCS-12 file using openssl.
 *
 * You create your own truststore file using:
 * <JAVA_HOME>\bin\keytool -import -trustcacerts -alias "own alias for the
 * holder of the certificate"
 * -file "Path to the certificate to be imported" keystore "file name of the
 * truststore file"
 *
 * Example: C:\Programme\Java\jdk1.6.0_22\bin\keytool -import -trustcacerts -
 * alias BUCServiceManagementCertificateAuthority
 * -file ./trustcerts/BUCServiceManagementCertificateAuthority.crt -keystore
 * mytruststore.jks
 *
 * Enter the keystore password again: <With the first certificate, set your own
 * password. Use this password for all subsequent imports>
 * Enter the password again: <Password repetition>
 * The following is the output of the certificate to be imported:
 * Owner: EMAILADDRESS=dalbers@materna.de, CN=BUC Service Management SSL CA,
 * OU=BU
 * C Service Management, O=MATERNA GmbH, ST=North Rhine-Westphalia, C=DE
 * Originator: EMAILADDRESS=dalbers@materna.de, CN=BUC Service Management
 * Certificate Authority, OU=BUC Service Management, O=MATERNA GmbH, L=Dortmund,
 * ST=North Rhine-Westphalia, C=DE
 * Serial number: 9

```

```

* Valid from: Thu Feb 03 12:00:52 CET 2011 until: Sat Feb 02 12:00:52 CET 2013
* Digital thumbprint of the certificate:
*       MD5: 0A:DA:6A:FB:5B:AD:F7:31:77:BD:BB:8E:4C:C2:58:08
*       SHA1:
E8:4A:92:C2:05:65:15:78:E9:B6:C1:D7:53:FC:60:48:F4:E3:25:AB
*       Signature algorithm name: SHA1withRSA
*       Version: 3
*       Enhancements:
*       #1: ObjectId: 2.5.29.19 Criticality=true
*       BasicConstraints:[
*       CA:true
*       PathLen:0
*       ]
*       #2: ObjectId: 2.5.29.15
Criticality=false
*       KeyUsage [
*       Key_CertSign
*       Crl_Sign
*       ]
*       #3: ObjectId: 2.5.29.14
Criticality=false
*       SubjectKeyIdentifier [
*       KeyIdentifier [0000: 32 60 8E F6
A7 CE 7B 92 CB 28 0F AB 05 05 96 9A 2`.....(.....0010: A2 95 CD 9A
....]]
*       #4: ObjectId: 2.5.29.31
Criticality=false
*       CRLDistributionPoints [
*       [DistributionPoint: [URIName:
http://ca.mbucqa.de/sslca.crl]
*       ReasonFlags: key compromise
*       CA compromise]
*       #5:
ObjectId: 2.16.840.1.113730.1.1 Criticality=false
*       NetscapeCertType [
*       SSL CA
*       ]
*       #6:
ObjectId: 2.5.29.35 Criticality=false
*       AuthorityKeyIdentifier [
*       KeyIdentifier [0000: D9 D4 57 C9 3E 64 CC EA 37 18 37 5D EB C1 6B 89
..W.>d..7.7]..k.0010: FB 4B 89 B0 .K..]
*       ]
*       Do you trust this certificate? [No]: Yes
* Certificate was added to keystore.
*

```

```

*
* @author Frank Rossol
*
*/
public class TestClientCertDatex2 {

    /**
     * @return
     */
    public D2LogicalModel testClientPullService() {
        URL wsdlUrl = null;
        QName serviceName = null;
        String subscriptionId="123456";
        // Server name under which the MDM platform is accessible
        String mdmHost="service.mac.mdm-portal.de";
        ClientPullService cpsp = null;
        try {
            wsdlUrl = new URL("https://" + mdmHost + "/BAST-MDM-
Interface/srv/"+ subscriptionId +"/clientPullService?wsdl");
            serviceName = new QName("http://datex2.eu/wsdl/clientPull/2_0",
"clientPullService");

            ClientPullService_Service cps = new
ClientPullService_Service(wsdlUrl,serviceName);
            cpsp = cps.getClientPullServicePort();
            D2LogicalModel d2LogicalModel = cpsp.getDatex2Data();
            return d2LogicalModel;

        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
        return null;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        // Path to the keystore file that contains the machine code
        System.setProperty("javax.net.ssl.keyStore",
"C:/MDMClient/java/SSL/privkey/myPrivateKey.p12");
        // Password of the keystore file
        System.setProperty("javax.net.ssl.keyStorePassword", "myKeyPass");
        // Type of keystore file
        System.setProperty("javax.net.ssl.keyStoreType", "pkcs12");

        // Path to the truststore file that contains the MDM certificates

        System.setProperty("javax.net.ssl.trustStore", "C:/MDMClient/java/SSL/myTrustS
tore.jks");
    }
}

```

```

        // Password of truststore file

        System.setProperty("javax.net.ssl.trustStorePassword","myTrustStorePass");

        // Releases all SOAP communication for debugging purposes.

        System.setProperty("com.sun.xml.ws.transport.http.client.HttpTransportPipe.dump","true");

        try {
            TestClientCertDatex2 tc = new TestClientCertDatex2();
            tc.testClientPullService();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

```

9 Annex B

9.1 Processing the p12 File for Apache Server Configuration

The Apache server configuration cannot handle any files of the type p12. For processing, manual steps that are described in the following chapters are required:

Export first the keys and certificates. Run the following command in the command prompt:

```
openssl.exe pkcs12 -in <p12-Datei> -out <sammeldatei.pem>
```

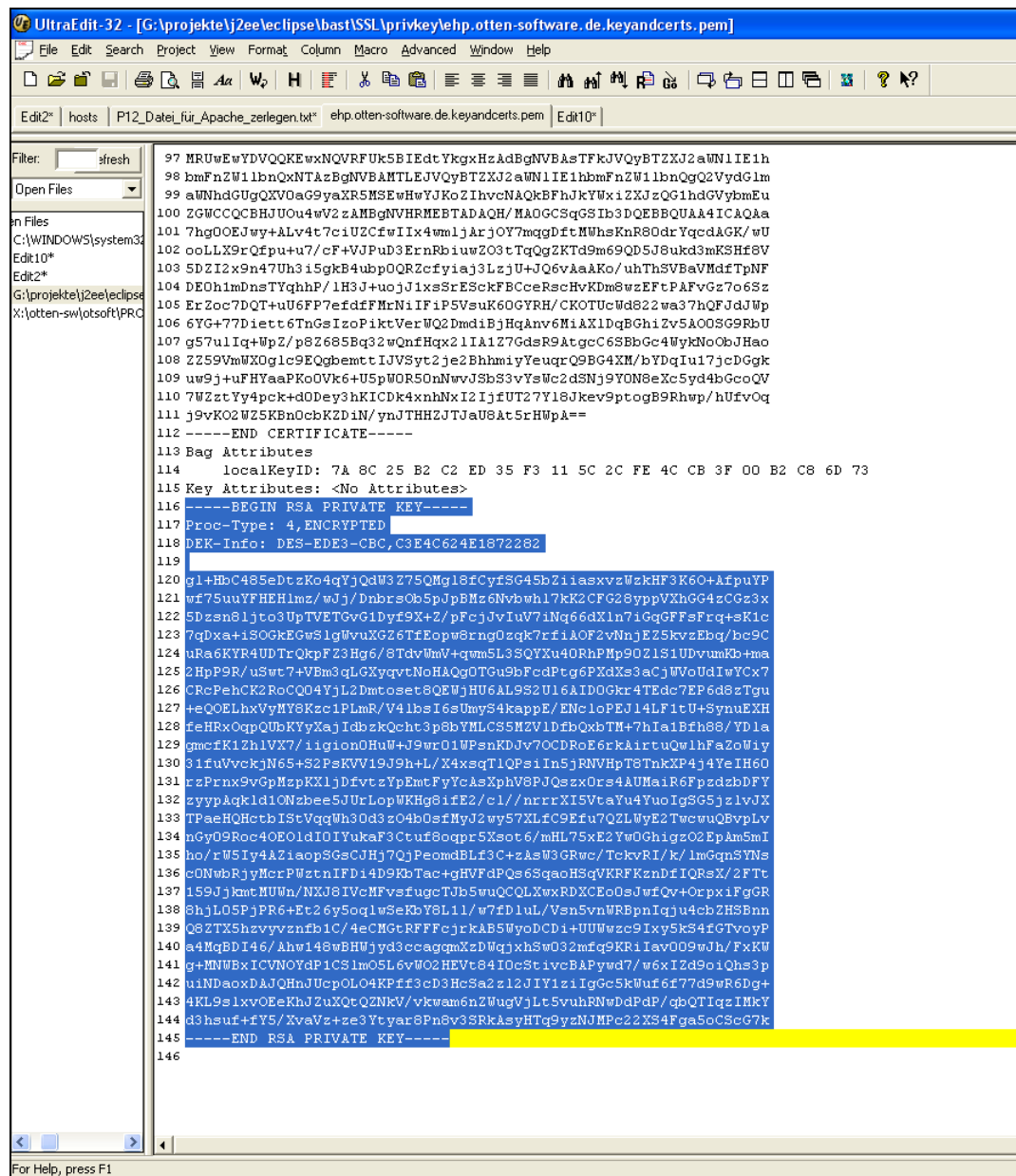
Example:

```
openssl.exe pkcs12 -in ehp.otten-software.de.p12 -out ehp.otten-software.de.keyandcerts.pem
```

Enter the certificate passwords in the openssl environment:

```
>Enter Import Password:      <Password from fax>
>MAC verified OK
>Enter PEM pass phrase:      <Self-selected passphrase for the key>
>Verifying - Enter PEM passphrase: <Repetition of the self-selected
passphrase for the key>
```

Open the file <sammeldatei.pem> with a text editor:



```
97 MRUwEwYDVQKEwxNQVRfUk5BIEdtYkgxHZAAdBgNVBAsTFkJKVQyBTZXXJ2aWN1IE1h
98 bmFnZW11bnQxNTAzBgNVBAMTEJVVyBTZXXJ2aWN1IE1hbmFnZW11bnQyQ2VydGlm
99 aWNhdGUgQXV0aG9yaXR5MSSEWwYjKoZlIhvcNAQkBFhJkYXUxI2XUzQg1hdGVybmEu
100 ZGwCCQCBHJUU0u4wV2zAMBgNVHRMBETADAQH/MAOGCSqGSIb3DQEBBQUAA4ICAQAa
101 7hgOOEJjy+ALv4t7ciUZCfwIIX4wmljArjOY7mqgDftMWhsKnR80drYqcdAGK/wU
102 oolLX9rQfpu+u7/cF+VJPUd3ErnRbiuwZ03tTqQgZKTd9m69QD5J8ukd3mKSHf8V
103 5DZ12x9n47Uh3i5gkB4ubpOQRZcfyiaj3LzjU+JQ6vAaAKo/uhThSVBaVmdfTpNf
104 DEOh1mDnsTYqhhP/1H3J+uoJ1xsSrESckFBCCeRscHvKdm8wzEftPAFvGz7o6S2
105 ErZoc7DQT+uU6FP7efdfMrNiIFiP5VsuK60GYRH/CKOTUcWd822wa37hQfJdJWp
106 6YG+77Diet6TnGsIzoPiktVerWQ2DmdibJHqAnv6MiAX1DqBghiZv5A00SG9RbU
107 g57u1Iq+Upz/p8Z685Bq32wQnfHqx21IA127GdsR9AtgcC6SBbGc4WYkNoObJHao
108 Z2S9VnWx0g1c9EQgbemttIJVSyt2je2BhhmiyYeuqrQ9BG4XM/bYdQu17jcdGgk
109 uw9j+uFHYaaPKoOVk6+U5pW0R50nNwvJsbS3vYsWc2dSNj9YON8eXc5ydh4bGmoQV
110 7WZctYy4pck+d0DeY3hKICDk4xnhNxI2IjUT27Y18Jkev9ptogB9Rhpw/hUfvOq
111 j9vKo2WZ5KBN0ckZDiN/ynJTHH2JTJaU8At5rHWpA==
112 -----END CERTIFICATE-----
113 Bag Attributes
114   localKeyID: 7A 8C 25 B2 C2 ED 35 F3 11 5C 2C FE 4C CB 3F 00 B2 C8 6D 73
115 Key Attributes: <No Attributes>
116 -----BEGIN RSA PRIVATE KEY-----
117 Proc-Type: 4, ENCRYPTED
118 DEK-Info: DES-EDE3-CBC,C3E4C624E1872282
119
120 g1+HbC485eDtcKo4qYjQdW3Z75QMgl8fCyfSG45bZ1iasxvzWzkHF3K60+AfpuYP
121 wf75uuYFHEHlmz/wJj/DnbsOb5pJpBMz6Nvbwh17kK2CFG28yppVXhGG4zCGz3x
122 5Dzsn81jtc3UpTVETGvG1Dyf9X+Z/pFcjJvIuV7iNq66dXln7iGqGFFsFrq+sK1c
123 7qDxa+i5OGkEGwSlgWvuXGZ6TfEopw8rng0zqk7rfiAOF2vNnJzE5kzvEbq/bc9C
124 uRa6KYR4UDTrQkpFZ3Hg6/8TdvWmV+qwm5L3SQYXu40RhFMp90Z1S1UDvunaKb+ma
125 2HpP9R/uSw7+VBm3qLGXyqvtNoHAQgOTGu9bFcdPtg6PXds3aCjWVoUdiWYC7u
126 CRcPehCK2RoCQ04YjL2Dmtoset8QEWjHU6AL9S2U16AID0Gkr4TEdc7EP6d8zTgu
127 +eQOELhxVYMY8Kzc1PLmR/V4lbsI6sUmyS4kappe/ENCloPEJ14LF1tU+SynuEXH
128 feHRxOqpQUbKYXajIdbzkQcht3p8bYMLCS5MZV1dfbQxbTM+7hIa1Bfh88/YDla
129 gmcFk12h1VX7/iigionDHuW+J9wr01WpSnKDJv70CDRoE6rkairtuQw1hFa2oWiy
130 3ifuVvckjN65+S2PeKVV19J9h+L/X4xsgT1QF=iIn5jRNVHPT8TnkXP4j4YeIH60
131 rzPrnx9vGpMzpKX1jDfvztzYpEmtFyYcAsXphV8PJQsxx0rs4AUMaiR6FpdzdbDFY
132 zyypqgkld1cNzbee5JUrLopWKHg8ifE2/cl//nrrrXI5VtaYu4YuoIg8G5jzlvJX
133 TPaeHQHctbIstVqWqH3Od3z04b0sfMyJ2wy57XLf9CEfu7QZLWye2TweWuQbvpLv
134 nGy09Roc4OEOldIOIYukaF3Ctuf8oqpr5Xsot6/mHL75xE2YwOGhigzO2EpAm5mI
135 ho/rW5Iy4ZiaopSGcJHj7QjPeomdBLf3C+zAsW3GRwc/TckvRI/k/1mGqnSYNs
136 cONwbRjyMcrPwztnIFDi4D9KbTac+gHVfDPQs6SgaoHSqVKRfKznDfIQRsX/2FTt
137 159JjkmTmUWn/NXJ8IVcMFvsfugeTJb5wuQCLXwxRDXCEoDsJwfQv+OrpxiFgGR
138 8hjL05PJPR6+Et26y5oqlwSeKhY8L1l/w7fdluL/Vsn5vnWRBpnIqj4ubc2H3Bnn
139 Q82TXShzvyvznfb1C/4eCMGtRFFFcjrkAB5WyoDCD1+UUWwzc9Ixy5kS4fGTvoyp
140 a4MgBDI46/Ahw148wBHWjyd3ccagqmXzDwqjxhSw032mfq9KRiIav009wJh/FxKW
141 g+MNUWbXICVNOYdP1CS1mO5L6vW02HEVt84I0cStivcBAPyvd7/w6xI2d9oiQhs3p
142 uINDaoxDAJQHnJUcPOLO4KFPf3cD3HcSa2z12JIY1ziIgGc5kWuf6f77d9wR6Dg+
143 4KL9slxvOEKhJ2uXQtQZNkV/vkwam6n2WugVjLt5vuhRNwDdPdP/qbQTIqzIMKY
144 d3hsuf+fY5/XvaVz+ze3Ytyar8Pn8v3SRkAsyHTq9yzNJMPc22XS4Fga5oCScG7k
145 -----END RSA PRIVATE KEY-----
146
```

Figure 16: File <sammeldatei.pem>

Copy the part of

```
--- BEGIN RSA PRIVATE KEY ---
```

until

```
---END RSA PRIVATE KEY ---
```

to a new file named <server.key>

Remove the passphrase to prevent that it is requested each time the server is restarted:

```
openssl rsa -in <server.key> -out <server.key.nopass >
```

Example:

```
openssl rsa -in server.key -out ehp.otten-software.de.key  
> Enter passphrase for server.key:      <Enter the previously self-  
selected passphrase>  
>writing RSA key
```

Enter the generated .key file in the Apache configuration under the following attribute:

```
SSLCertificateKeyFile
```

As a next step, split the certificates into two files. To do this, first open the file <sammeldatei.pem> with a text editor:

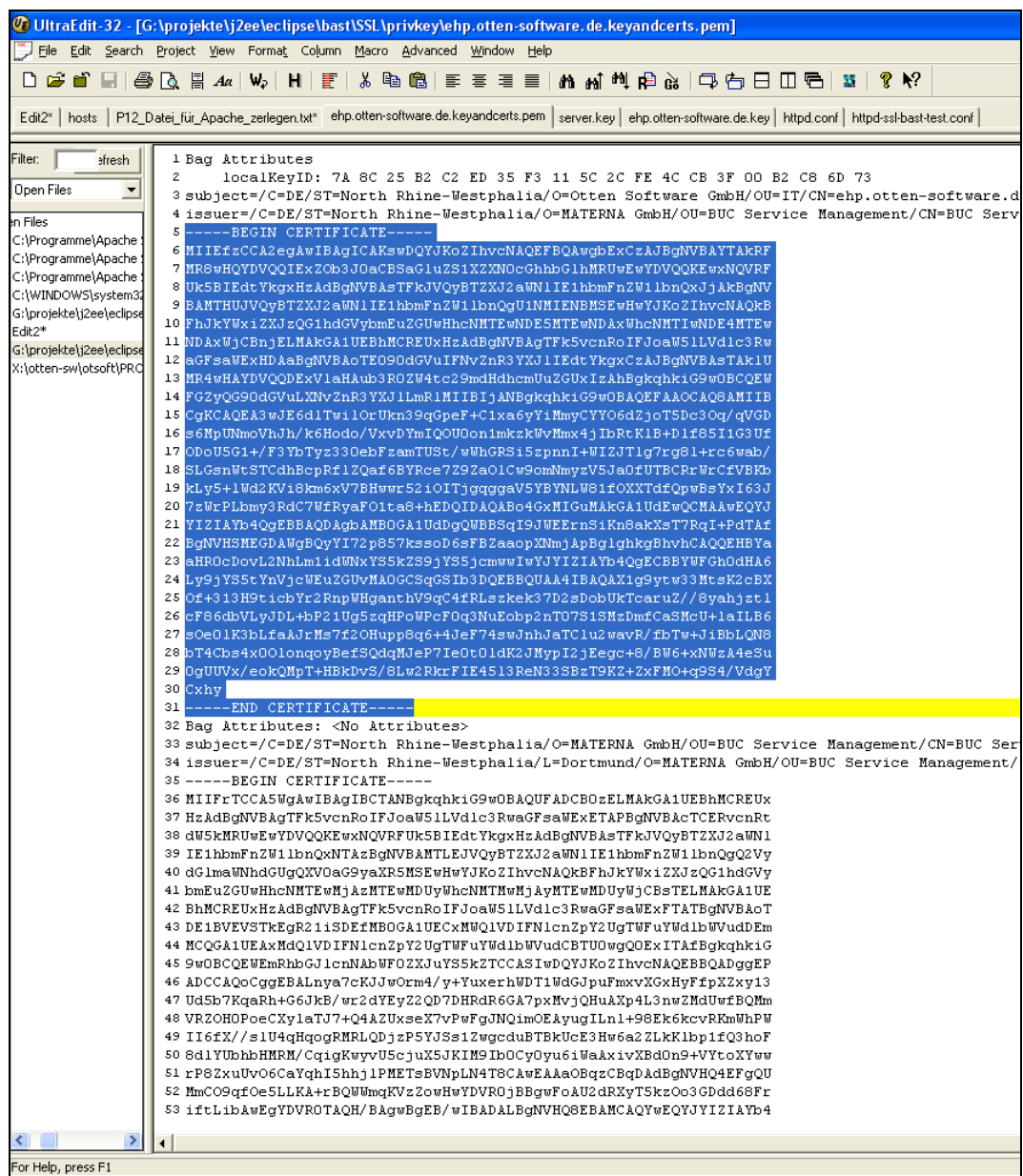


Figure 17: File <sammeldatei.pem>

Copy the server certificate into a new text file <server.crt>.

Enter this file in the Apache configuration under the following attribute:

SSLCertificateFile

Copy the remaining certificates into a new text file <ca-cert-chain.crt>.

Enter this file in the Apache configuration under the following attribute:

SSLCertificateChainFile

Enter the MDM client certificate incl. the certificate hierarchy under the following Apache attribute:

SSLCACertificateFile

Example of an Apache configuration:

```
SSLCertificateFile "C:\Programme\Apache Software  
Foundation\Apache2.2\conf\ssl\ssl.crt\ehp.otten-software.de.crt"  
SSLCertificateKeyFile "C:\Programme\Apache Software  
Foundation\Apache2.2\conf\ssl\ssl.key\ehp.otten-software.de.key"  
SSLCertificateChainFile "C:\Programme\Apache Software  
Foundation\Apache2.2\conf\ssl\ssl.crt\bast_cert_chain.crt"  
SSLCACertificateFile "C:\Programme\Apache Software  
Foundation\Apache2.2\conf\ssl\ssl.crt\bast_trust_chain.crt"
```