European Network of
Transmission System Operators
for Electricity

MADES
Communication Standard

2011-11-07

DOCUMENT APPROVED ON 2012-01-18
VERSION 1.0

Table of Contents

1 INTRODUCTION 8
1.1 WHAT IS MADES ABOUT? 8
1.2 MADES GOVERNANCE 9
1.3 WHAT IS MADES INTENDED FOR? 9
1.4 WHAT MADES IS NOT? 11
1.5 USED ACRONYMS 13

2 HIGH LEVEL CONCEPTS 14
2.1 GENERAL OVERVIEW 14
2.2 MESSAGE DELIVERY AND TRANSPARENCY 15
2.3 SECURITY AND RELIABILITY 16
2.4 MAIN COMPONENTS 17
2.5 DISTRIBUTED ARCHITECTURE 18
2.6 COMPONENTS' EXPOSED INTERFACES 19
2.7 SECURITY FEATURES 20
2.7.1 Overview 20
2.7.2 Transport-layer Security 20
2.7.3 Message-level Security 21
2.7.4 Non Repudiation 23

3 COMPONENTS' FUNCTIONS 25
3.1 ROUTING MESSAGES 25
3.2 COMPONENT AND MESSAGE UNIQUE IDENTIFICATION (ID) 26
3.3 BUSINESS-TYPE OF A BUSINESS-MESSAGE 26
3.4 DELIVERY-STATUS OF A BUSINESS-MESSAGE 27
3.5 COMMUNICATION BETWEEN COMPONENTS 29
3.5.1 Establishing a Secured Communication Channel between two Components 29
3.5.2 Token Authentication of the Client Component 30
3.5.3 Requests' Authorization 30
3.5.4 Request/Reply Validation 31
3.6 STORING MESSAGES IN COMPONENTS 31
3.7 LIFECYCLE OF A MESSAGE STATE WITHIN A COMPONENT 32
3.8 TRANSFERRING A MESSAGE BETWEEN TWO COMPONENTS (HANDSHAKE) 33
3.9 ACCEPTING A MESSAGE 35
3.10 EVENT MANAGEMENT 36
3.10.1 Acknowledgements 36
3.10.2 Notifying Events 37
3.10.3 Lifecycle of an Acknowledgement 40
3.10.4 Processing a transferred Acknowledgement 40
3.11 MESSAGE EXPIRATION 41
3.12 CHECKING THE CONNECTIVITY BETWEEN TWO ENDPOINTS (TRACING-MESSAGES) 42
3.13 ORDERING THE MESSAGES (PRIORITY) 43
3.14 ENDPOINT FUNCTIONS 44
3.14.1 Compression 45
3.14.2 Signing 45
3.14.3 Encryption 46
3.15 GATEWAY FUNCTIONS 49
3.16 NODE FUNCTIONS 50
3.16.1 Synchronizing Directory with other Nodes 51
3.16.2 Updating the Synchronization Nodes' List 52
3.17 CERTIFICATES AND DIRECTORY MANAGEMENT 53

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.17.1 Definitions and Principles 53
3.17.2 Certificates: Format and Unique ID 54
3.17.3 Used Certificates and Issuers (CAs) 55
3.17.4 Directory Services 56
3.17.5 Caching Directory Data 57
3.17.6 Endpoint requesting Directory Services through a Gateway 57
3.17.7 Trusting the Certificates of others Components 57
3.17.8 Renewing the expired Certificates 58
3.17.9 Revoking a Certificate 58
3.18 About Gateways 60
3.18.1 Endpoint changing of Gateway 60
3.18.2 Endpoint without a Gateway 60

4 MANAGING THE VERSION OF THE MADES SPECIFICATION 61

4.1 Issues and Principles 61
4.1.1 Rolling out a New Version (Mversion and N-compliance) 61
4.1.2 Service Compatibility 62
4.1.3 Message Compatibility 62
4.1.4 Interface with BAs 63
4.2 Using the correct Version for Services and Messages 64
4.2.1 Node Synchronization and Authentication 64
4.2.2 Directory Services and Network Acceptance 65
4.2.3 Messaging Services 66
4.2.4 Which version to use to send a message? 67

5 INTERFACES AND SERVICES 69

5.1 Introduction 69
5.1.1 Error Codes 69
5.1.2 Types for Time 70
5.2 Endpoint Interface 70
5.2.1 Overview 70
5.2.2 Services 71
5.2.2.1 SendMessage Service 71
5.2.2.2 ReceiveMessage Service 72
5.2.2.3 CheckMessageStatus Service 73
5.2.2.4 ConnectivityTest Service 73
5.2.2.5 ConfirmReceiveMessage Service 74
5.2.3 File System Shared Folders (FSSF) 75
5.2.3.1 Introduction 75
5.2.3.2 Used Files and File Naming Convention 75
5.2.3.3 Concurrent Access to Files 77
5.2.3.4 Configuring FSSF 77
5.3 Gateway & Node Interfaces 78
5.3.1 Overview 78
5.3.2 Authentication Service 80
5.3.3 Messaging Services 81
5.3.3.1 “Transfer confirmation” versus “Acceptance” 81
5.3.3.2 UploadMessages Service 82
5.3.3.3 DownloadMessages Service 83
5.3.3.4 ConfirmDownload Service 84
5.3.4 Directory Services 85
5.3.4.1 SetComponentMversion Service 85
5.3.4.2 GetCertificate Service 86
5.3.4.3 GetComponent Service 87
5.3.5 Node Synchronization Interface 89
5.3.5.1 GetNodeMversion Service 89
5.3.5.2 GetAllDirectoryData Service 89
5.4 Format of the Node-list file 90
5.5 Typed Elements used by the Interfaces 91
5.5.1 Using MTOM 91
5.5.2 AuthenticationToken 91
5.5.3 Certificate 91

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.5.4 CertificateType (string enumeration) 92
5.5.5 ComponentCertificate 92
5.5.6 ComponentDescription 92
5.5.7 ComponentInformation 92
5.5.8 ComponentType (string enumeration) 93
5.5.9 Endpoint 93
5.5.10 InternalMessage 93
5.5.11 InternalMessageType (string enumeration) 95
5.5.12 MessageMetadata 95
5.5.13 MessageState (string enumeration) 96
5.5.14 MessageStatus 96
5.5.15 MessageTraceItem 97
5.5.16 MessageTraceState (string enumeration) 97
5.5.17 NotConfirmedMessageResponse 98
5.5.18 NotUploadedMessageResponse 98
5.5.19 ReceivedMessage 98
5.5.20 RoutingInformation 99
5.5.21 SentMessage 99
5.6 DESCRIPTION OF THE SERVICES USING WSDL 100
5.6.1 Endpoint Interface 101
5.6.2 Gateway & Node Interface 108
5.6.2.1 Authentication Service 108
5.6.2.2 Messaging Services 110
5.6.2.3 Directory Services 115
5.6.2.4 Node Synchronization Interface 120
5.6.3 XML Signature Example 124

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

List of figures

Figure 1: MADES overall view 9
Figure 2: MADES scope 10
Figure 3: MADES key features 14
Figure 4: MADES message delivery overview 15
Figure 5: MADES Security and Reliability 16
Figure 6: MADES components 17
Figure 7: MADES network distributed architecture 18
Figure 8: MADES Interfaces and Services 19
Figure 9: MADES security overview 20
Figure 10: MADES secure communication initiation 21
Figure 11: Message signature 21
Figure 12: Message encryption and decryption 22
Figure 13: Non repudiation 23
Figure 14: Delivery route of a business-message 25
Figure 15: Reported events during the delivery of a business-message 27
Figure 16: Lifecycle of the local state of a business-message within a component 32
Figure 17: Transfer handshake when uploading of a message 34
Figure 18: Transfer handshake when downloading of a message 34
Figure 19: Acknowledgements along the route of the business-message 37
Figure 20: Encryption process 47
Figure 21: A Node synchronizes with two other Nodes 51
Figure 22: Certificates and Certificate Authorities (CAs) for a MADES network 55
Figure 23: Managing the specification version – Node Synchronization and Authentication 64
Figure 24: Managing the specification version – Directory services 65
Figure 25: Managing the specification version – Messaging services 66
Figure 26: Managing the specification version – Which version to use to send a message? 67
Figure 27: Gateway & Node Interfaces – Overview 79
Figure 28: Gateway & Node Interfaces – Authentication Service 80
Figure 29: Gateway & Node Interfaces – Messaging Services 81
Figure 30: Gateway & Node Interfaces – Messaging Services – UploadMessages Service 82
Figure 31: Gateway & Node Interfaces – Messaging Services – DownloadMessages Service 83
Figure 32: Gateway & Node Interfaces – Messaging Services – Download authorization 83
Figure 33: Gateway & Node Interfaces – Messaging Services – ConfirmDownload Service 84
Figure 34: Gateway & Node Interfaces – Directory Services 85
Figure 35: Gateway & Node Interfaces – Directory Services – GetCertificate Service 86
Figure 36: Gateway & Node Interfaces – Directory Services – GetComponent Service 87

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Copyright notice:

Maintenance notice:

THIS DOCUMENT IS MAINTAINED BY THE ENTSO-E WG EDI. COMMENTS OR REMARKS ARE TO BE PROVIDED AT EDI.Library@entsoe.eu

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Revision History

Version	Release	Date	Paragraph	Comments
1	0	2012-01-18		Approved by the Market Committee.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

1 INTRODUCTION

The MADES initiative specifies a standard for a communication platform which every Transmission System Operator (TSO) in Europe may use to reliably and securely exchange documents. Consequently a European market participant (trader, distribution utilities, etc.) could benefit from a single, common, harmonized and secure platform for message exchange with the different TSOs; thus reducing the cost of building different IT platforms to interface with all the parties involved. This also represents an important step in facilitating parties entering into markets other than their national ones.

The “MADES” acronym is short for: **MArket Data Exchange Standard**.

The MADES initiative has been adopted by the ENTSO-E Electronic Data Interchange (EDI) Working Group (WG) to facilitate communication between TSOs and European electricity market participants.

The document is published on the ENTSO-E website (https://www.entsoe.eu).

1.1 WHAT IS MADES ABOUT?

MADES is a specification for a decentralized common communication platform based on international IT protocol standards:

- From a Business Application (BA) perspective, MADES specifies software interfaces to exchange electronic documents with other BAs. Such interfaces mainly provide means to send and receive documents using a so-called “MADES network”. Every step of the delivery process is acknowledged, and the sender can request about the delivery status of a document. This is done through acknowledgement, which are messages returned back to the sender. This makes MADES network usable for exchanging documents in business processes requiring reliable delivery.

- MADES also specifies all services for the Business Application (BA); the complexities of recipient localisation, recipient connection status, message routing and security are hidden from the connecting BA. MADES services include directory, authentication, encryption, signing, message tracking, message logging and temporary message storage.

A MADES network acts as a post-office organization. The transported object is a “message” in which the sender document is securely repackaged in an envelope (i.e. a header) containing all the necessary information for tracking, transportation and delivery.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

1.2 MADES GOVERNANCE

ENTSO-E shall continue to maintain the MADES. The aim of the governance is to provide stability in the standard. There will be at most one version released per year, to reduce the burden of change on the user community.

MADES version numbering is in full integers. Revisions may only apply to the specification document which could be reissued to fix reported errors or ambiguities. The last issued revision is the only one applicable for a version.

MADES specifies rules to allow a smooth rollout process when upgrading the implemented version of an existing operational network — see § 4.

1.3 WHAT IS MADES INTENDED FOR?

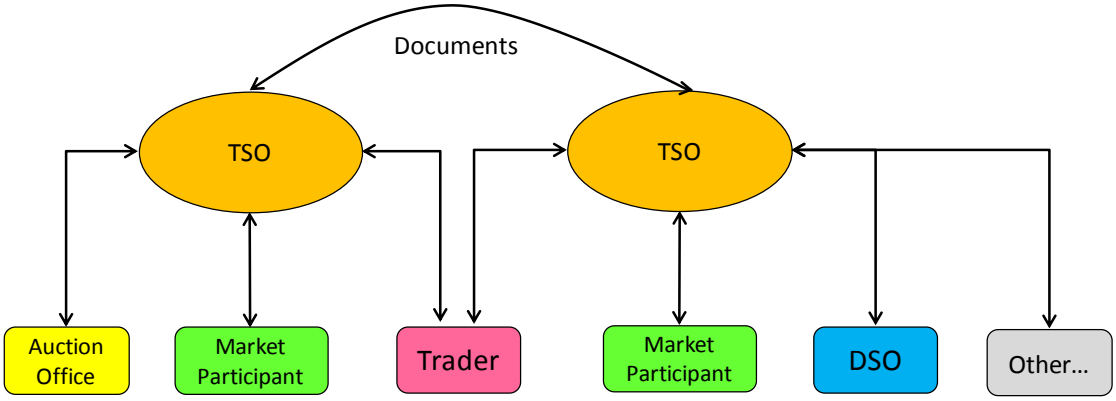


Figure 1: MADES overall view

MADES first intention is to provide TSOs with a standardized communication access point to securely exchange documents with others parties involved in the European electricity market. Such parties include TSOs, Distribution System Operators (DSO), Balance Responsible Parties (BRP), Capacity Traders (CT), Market Operators (MO), Producers, Transmission Capacity Allocators (TCA), etc., namely any party playing a role defined in "*The Harmonised Electricity Market Role Model*"[1].

Please be aware that MADES is a general communication standard. It is not limited in usage to market data or the electricity industry and can be used more widely for any non-real time data exchange application.

The MADES vision is that each party implements MADES access points (referred as Endpoints) connected to his Information System (IS), where he may securely send and receive documents to and from other parties.

MADES is not concerned with specific business functionality, nor creating a new IT standard, nor building communication infrastructure. For the market to operate correctly, parties must exchange electronic documents conforming to predefined business logic and in a cost effective way, namely by using existing IT standards and protocols over existing communication infrastructures.

[1] The Role Model document is available at ENTSO-E EDI Library.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

The purpose of MADES is to create a data exchange standard comprised of standard protocols and utilizing IT best practices to create a mechanism for exchanging data over any TCP/IP communication network, in order to facilitate business to business information exchanges.

New market rules induce new business processes and activities, and generally require new information exchanges between parties. Experience shows that, for the exchanges to operate according to the business goals, the chosen technical solution results from an agreement of involved parties gathering various constraints, including implementation time scale, vendors' offer, already existing communication links, integration capabilities of existing Information Systems, confidentiality of exchanged information, legal risks, etc.

Where business processes require information to be exchanged between multiple systems or multiple parties, solutions developed bi-laterally may become extremely complex, with each interface taking time, money and resources to be developed and be maintained. It is also a noticeable consequence that some parties acting in several countries, such as traders, may have to install different communication tools in order to interface with different trading solutions. The future vision is a single interface between all parties in all areas of the electricity market of Europe.

MADES is a step forward to a standardized communication solution, especially in the following areas:

- Future ENTSO-E European projects or TSO domain projects should reduce required resources and time to operate the new market rules (less design, less tests).
- A MADES access point may be implemented using any software compliant with this publicly accessible specification. Vendors are encouraged to either interface or integrate a MADES compliant client in their Business Applications.
- European market should be facilitated as multi-countries actors should have a single access to market, or can deploy a unique solution in different sites.

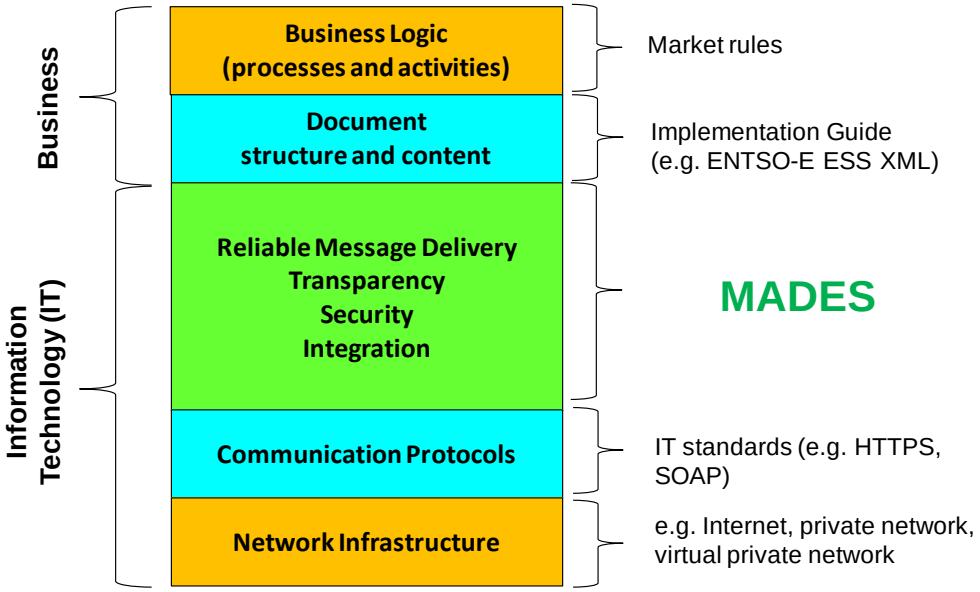


Figure 2: MADES scope

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

MADES is agnostic to the business logic using it, which means it can support any business process whatever the document types being transmitted might be (e.g. XML, binary) and whatever the sequence for the exchanges.

MADES is independent of the physical underlying communication Infrastructure, which can be any IP (Internet Protocol) network, such as Internet, a physical private infrastructure, or a multi access-point virtual private network (VPN).

MADES relies on and only on non-proprietary IT standards for communication protocols, data integrity, signing and confidentiality (encryption), peer access point authentication, peer party authentication, parties' directory (e.g. HTTPS, SOAP, X.509).

1.4 WHAT MADES IS NOT?

MADES is not a synchronous messaging system:

MADES specifies a framework for asynchronous communication; therefore the architecture contains hubs (namely Nodes) that offer a service to temporary store messages.

A synchronous messaging system would require that the sender access point establishes a direct IP connection to the recipient access point for exchanging messages, thus meaning that both have to be online at the same moment.

The way MADES achieves a transfer is to send each message to a message queue at the Node where the recipient then retrieves it. As an asynchronous process, there is no direct connection between sender and recipient, i.e. no handshake exists between peer access points.

This design has two main benefits:

- The recipient may be offline for a given duration[2] without losing any information. So, parties not involved in frequent or business critical processes can turn off their access points — actually have it installed on a non-permanently network connected computer.
- The security level for the architecture is higher than other methods since access points are not required to accept incoming connections, i.e. all connections are initiated by the client and so no exception to firewall rules is required.

MADES is not intended for real time messaging:

The magnitude for the end-to-end duration of message transportation is not defined and may significantly vary depending on the implementation and the underlying infrastructure due to the following considerations:

- The transfer process is asynchronous and not event-driven.
- The exchanged documents can range in size from kilobytes to several megabytes[3].

[2] The period for which messages will be retained may be configured – see § 3.9.

[3] The specific limits to maximum message size which be transferred will depend on the architecture and the configuration of each network.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

- The security management requires processing resource for signing, verifying signature, encryption, decryption.

As a consequence, MADES is not intended and should not be recommended nor considered for processes or projects dealing with real time messaging[4].

MADES is not about designing or delivering software:

MADES is concerned with the design of interfaces of access points and hubs (Nodes); however the internal design of those components is of out of scope. Internal design considerations could include:

- Functional architecture for message management, storage and archiving, security management, directory management.
- Logical and technical architecture for performance.
- Redundancy for high availability solution.
- Software packaging and installation process.
- Administration tools design and security (e.g. Graphical User Interface).
- Component supervision agent.
- Statistics and Key Performance Indicators (KPI) collection.

MADES is not about setting up a network:

MADES is a specification. A MADES network is a group of parties, each operating communication components (Endpoints, Nodes) which comply with the specification. Setting up a MADES network requires more than software, and a governance team must:

- choose the underlying network infrastructure,
- define the network access rules,
- define the access points identification scheme,
- define the network joining process (e.g. using a test network first),
- define which parties can or must host the Nodes,
- define the network supervision organisation,
- formalize the Node administrator role and tasks,
- define and supervise the planning in case of a version upgrade,
- specify the archiving/logging duration requirements,
- define the certificate policy which states the roles and duties of the different actors of the public key infrastructure, the certificate validity duration, the trusted certificate authorities (CAs), the process to revoke and renew the certificates,
- define the backup and archiving strategies,
- etc.

[4] Examples of real-time messaging to which this is not suited include signalling, voice or video communications.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

148 MADES is not a complete business solution:

149 Please refer back to Figure 2 to see what a complete solution would include.

150 Some parties may look for a plug-and-play tool (easy to install, use, administrate, upgrade),
151 where documents can be sent and received using drag-and-drop by selecting the recipient in
152 list, where logs and archives can be easily scanned, scrolled and browsed, where errors
153 send configurable alarms, etc.

154 MADES does not specify any Graphical User Interface; it focuses on interfaces to ensure
155 that access points interoperate. However MADES in no way impedes the creation of a
156 human machine interface layer which may use MADES for message transport.

1.5 USED ACRONYMS

AES	Advanced Encryption Standard — A symmetric cryptographic algorithm.
BA	Business Application
DER	Distinguished Encoding Rules — A format for X.509 digital certificates
DMZ	DeMilitarized Zone
EDI	Electronic Data Interchange
FSSF	File System Shared Folder
HTTPS	HTTP (HyperText Transfer Protocol) Secured with the TLS protocol to provide encrypted communication and secure identification of a web server.
IETF	Internet Engineering Task Force
ID	IDentity
IP	Internet Protocol
IS	Information System
IT	Information Technology
ITU-T	The standardization sector of the International Telecommunication Union (ITU)
PKI	Public Key Infrastructure
RFC	Request For Comments
RSA	Rivest Shamir Adleman – An asymmetric cryptographic algorithm.
SHA	Secure Hash Algorithm. SHA-1 and SHA-512 are cryptographic hash functions designed by the National Security Agency (United States Department of Defence).
SOAP	Simple Object Access Protocol
TLS	Transport Layer Security
TSO	Transmission System Operator
URL	Uniform Resource Locator
UTF-8	UCS (Universal Character Set) Transformation Format — 8-bit.
UUID	Universal Unique IDentifier
W3C	World Wide Web Consortium
WG	Working Group
WAN	Wide Area Network
XML	eXtended Markup Language
X.509	An ITU-T standard for a Public Key Infrastructure (PKI)

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

2 HIGH LEVEL CONCEPTS

2.1 GENERAL OVERVIEW

157 The purpose of the MADES standard is to specify a message delivery platform with following
158 key features:

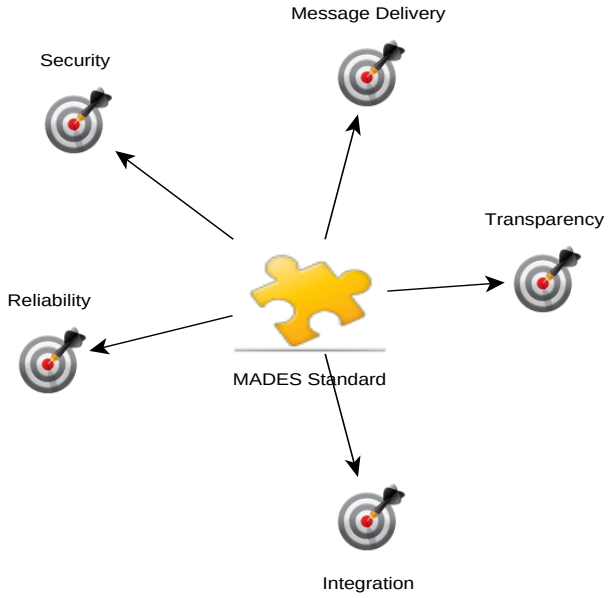


Figure 3: MADES key features

159 **Message delivery** – A party (sender) connected to the communication network can send a
160 message to another party (recipient), which is connected or can connect to the network.

161 **Transparency** – Any transported message can be tracked down to gather trustworthy
162 information about the state of delivery and traversal path.

163 **Security** – Only the recipient of the message is capable of reading the message-content.
164 The sender of any message can be unambiguously verified.

165 **Reliability** – A message cannot get lost.

166 **Integration** – The MADES functions for sending and receiving messages can be integrated
167 with wide variety of technologies.

168 Note: The first four key features (message delivery, security, transparency and reliability) are
169 capabilities of the communication system, while the other one (integration) is a design
170 characteristic of the components of the communication system.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

2.2 MESSAGE DELIVERY AND TRANSPARENCY

Message Delivery

The main feature of MADES is the message delivery function:

Figure 4: MADES message delivery overview

A message is transferred from a sender to a recipient. Both sender and recipient are Business Applications (BAs). A BA connects to a MADES Endpoint using a programming interface.

The sender and recipient view the MADES system only through the defined interface. The document transported between sender and recipient can be any text or binary data. Alongside with the document, a MADES message contains additional information, in a header (or envelope), including information to securely identify, transport and route the message such as a unique message ID, the identities of the sender and of the recipient, a business-type.

Transparency

The message path — from the sender's Endpoint to the recipient's Endpoint — goes through some components of the MADES network. When a message traverses a component, the later notifies the event and a new message (referred as an acknowledgement) is sent back to the sender's Endpoint. All the events notified during the message delivery can be retrieved by the sender's BA.

2.3 SECURITY AND RELIABILITY

188 MADES ensures reliable and secure message transfer and delivery:

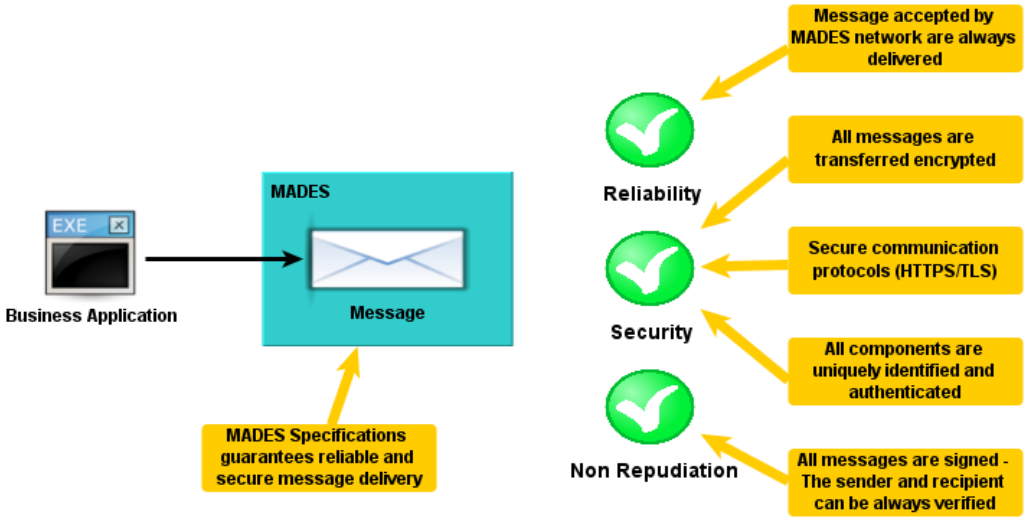


Figure 5: MADES Security and Reliability

189 A MADES communication system guarantees that any message accepted by the system will
190 not be lost. The sender can at any time check the delivery status of the messages
191 (delivering, delivered or failed).

192 The standard describes a logging mechanism to be implemented in all message handling
193 components to provide information about the message transfers; MADES describes non-
194 repudiation features, allowing the verification of a message and its header which includes the
195 sender, the recipient, the sending time, the delivery time, etc.

196 MADES defines the way to sign and encrypt the transported messages.

197 For the communication layer, the MADES components use the secure communication
198 protocols HTTPS. Information is transported encrypted. Moreover, both sides of
199 communication are authenticated using industry-standard PKI certificates.

200 For more information about the security features, see § 2.7.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

2.4 MAIN COMPONENTS

201 MADES describes three logical communication components and their interfaces.

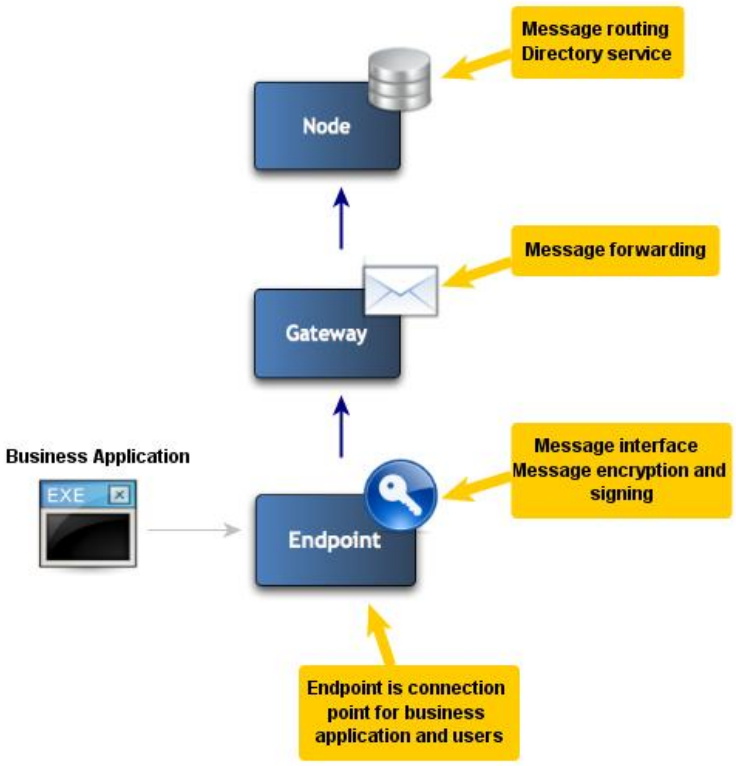


Figure 6: MADES components

202 From the users' or Business Application (BA) point of view, the crucial component is the
203 **Endpoint**, which provides the interface for the BAs to send and receive the messages.
204 Actually no graphical user interface is part of MADES; such an interface to provide a manual
205 way for sending and receiving messages is an application which can be integrated with the
206 Endpoint.

207 The **Gateway** component, as the name suggests, serves as a gateway for messages – it
208 either takes the message from an Endpoint and forwards it to a Node, or retrieves the
209 message from a Node and makes it available to an Endpoint. When deployed separately
210 from an Endpoint, a Gateway can be used to decouple and secure enterprise networks from
211 public networks (e.g. Internet).

212 The **Node** component serves as a central part of a MADES network. Each Node contains a
213 directory with information on all the registered network components, whether Endpoints,
214 Gateways or Nodes.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

2.5 DISTRIBUTED ARCHITECTURE

A MADES network may consist of multiple interconnected Nodes, each taking care of a part of the network:

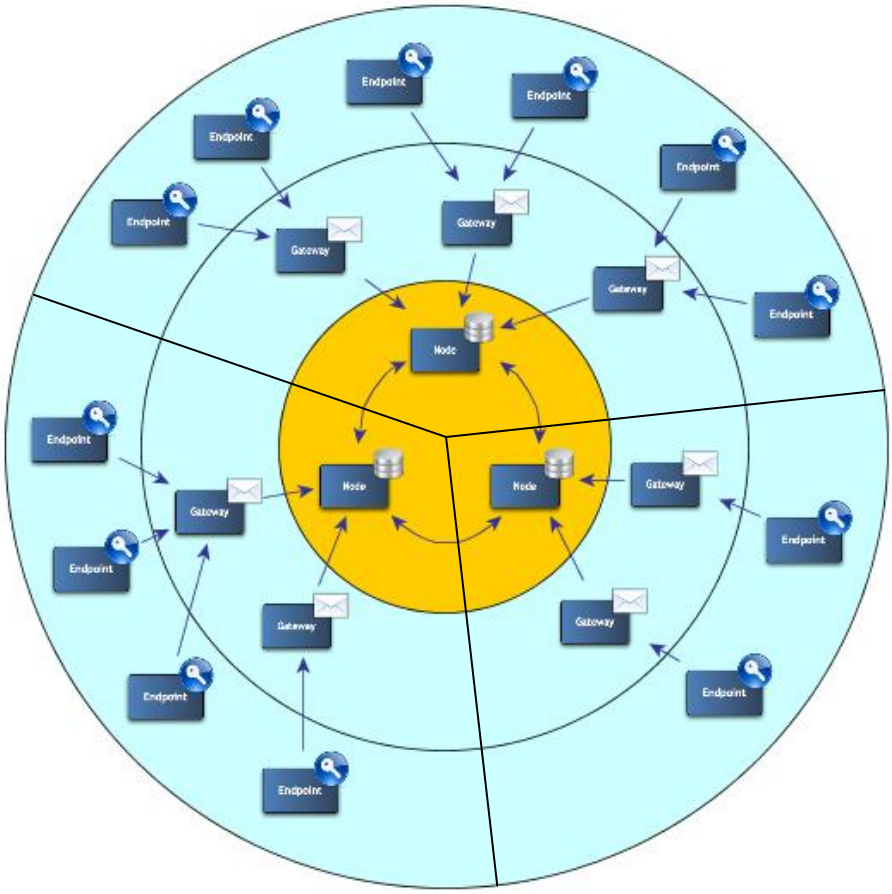


Figure 7: MADES network distributed architecture

A MADES network may contain a large number of collaborating components with the Nodes in the centre. A MADES network has a distributed architecture; it does not have any single central component. All Nodes have equal responsibilities; each manages a part of the network.

Each Endpoint or Gateway must register with a home Node. The components registered with a Node are referred as the registered components. Several Endpoints registered with the same home Node can share a Gateway also registered with the same home Node. Endpoints currently connected to a Gateway are the connected Endpoints.

Directory information about all registered Endpoints is regularly shared between the Nodes, using the Node synchronization interface; so that Endpoints registered with different home Nodes can exchange messages.

A Gateway can connect to any Node to send messages, but it can only receive messages from the home Node.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

2.6 COMPONENTS' EXPOSED INTERFACES

MADES standard specifies the interfaces between the components. All interfaces and services are presented in the following figure:

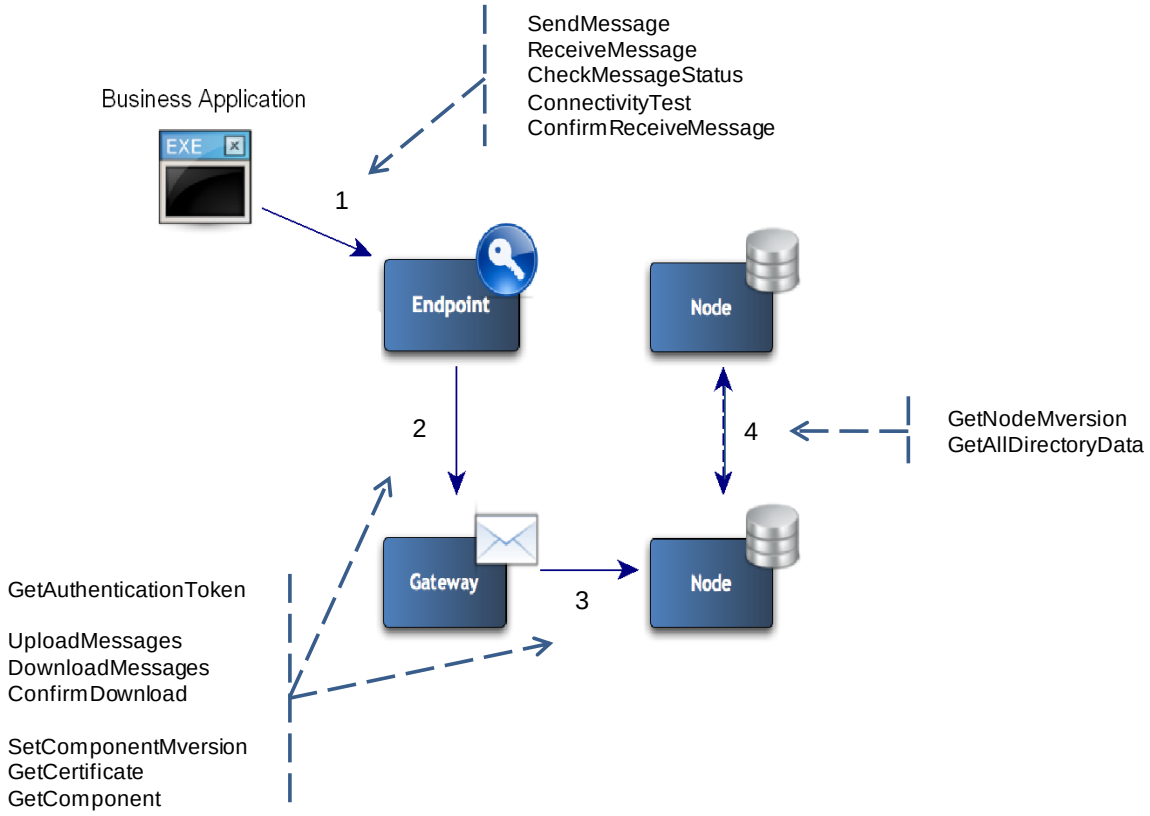


Figure 8: MADES Interfaces and Services

Each arrow on the figure show a component (at the tail of the arrow) using the interface exposed by the component at the tip of the arrow.

1. Endpoint Interface → used by a Business Application (BA) – see § 5.2.
   An Endpoint shall implement this interface for a BA to connect to a MADES network.

2. Gateway Interface → used by the Endpoints – see § 5.3.
   A Gateway shall implement this interface for an Endpoint to connect to a MADES network.

3. Node Interface → used by the Gateways – see § 5.3.
   A Node shall implement this interface to allow a Gateway to transfer the messages and to query the Node directory.

4. Node Synchronization Interface → used by the Nodes – see § 5.3.5.
   A Node shall implement this interface to synchronize directory data with the other Nodes of the MADES network.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

2.7 SECURITY FEATURES

2.7.1 OVERVIEW

Main goals of the MADES security definition are summarized by the following points:

- The security solution is transparent to the Business Applications (BAs) – no specific implementation shall be required in the application to communicate securely.
- Any message shall be readable only by the recipient.
- The sender of any message shall be unambiguously identified.
- Non-repudiation of the messages – it shall be possible to unambiguously prove that the sender sent the message and that the recipient received it.
- All communication routes shall be encrypted. The security solution complies with the X.509 public key infrastructure.

The security issues are covered on two levels: transport-layer security and message-level security.

On the **transport-layer**, MADES requires the communication between two components to always be encrypted. Two components that exchange information must first unambiguously identify each other.

On the **message-level**, MADES requires that all messages shall be signed and encrypted, so the sender of the message can be unambiguously identified and the message is only readable by the intended recipient.

2.7.2 TRANSPORT-LAYER SECURITY

The transport–layer security of the communication between components relies on the transport protocol layer. When communicating, components shall use a secure protocol HTTPS providing the encryption of the communication route. Mutual authentication of communicating components shall be handled using X.509 certificates; both the client and the server shall authenticate themselves by their respective authentication X.509 certificates.

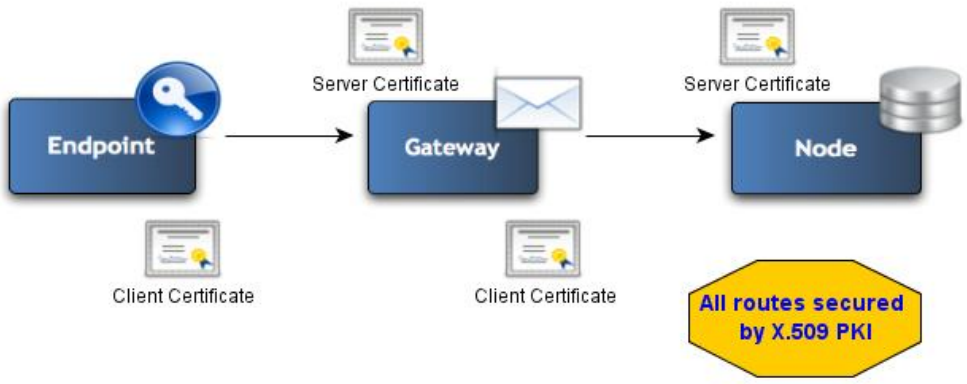


Figure 9: MADES security overview

The communication (i.e. the IP connection) between components shall always be initiated by the client. This provides higher security on client-side by not having to allow incoming connections through firewalls.

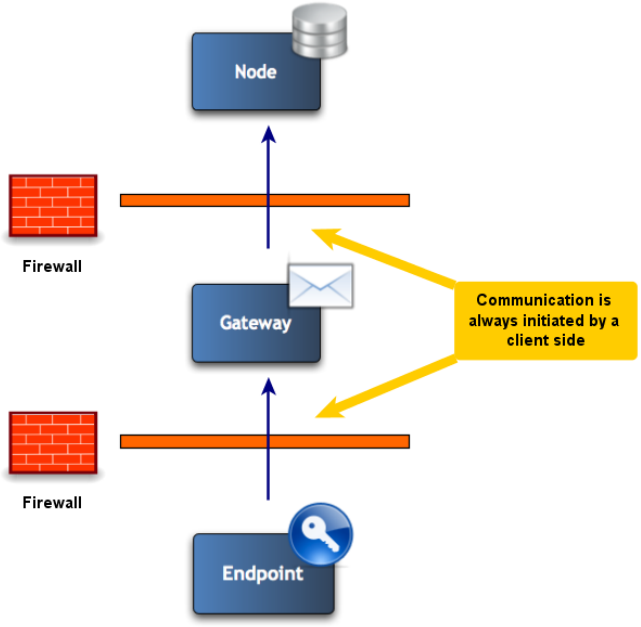


Figure 10: MADES secure communication initiation

2.7.3 MESSAGE-LEVEL SECURITY

The unambiguous identification of the sender of any message sent via the MADES network is enabled by usage of **digital signatures**:

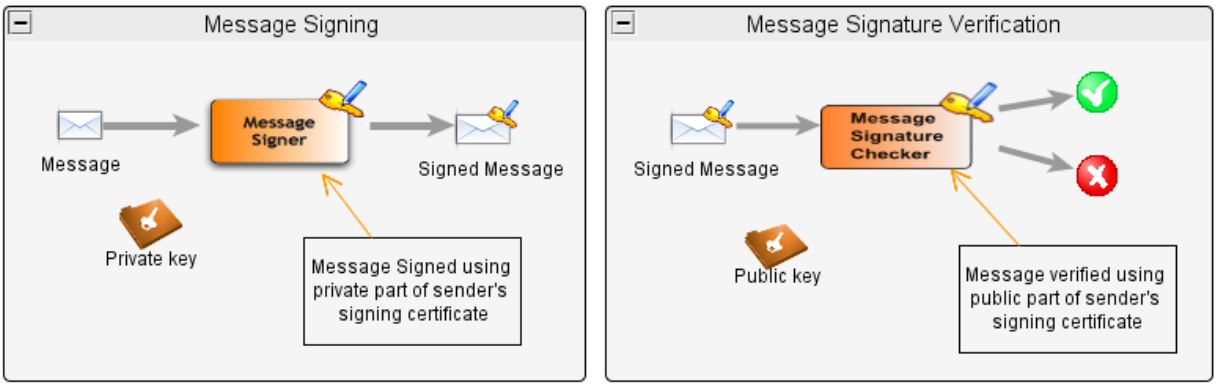


Figure 11: Message signature

On the sender's Endpoint, the message is signed using the sender's private key of a signing certificate. On the recipient's Endpoint, the message signature is verified using the sender's public key of the certificate.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

275 **Any message sent via the MADES shall be encrypted,** so that only the intended recipient
276 can read the message-content:

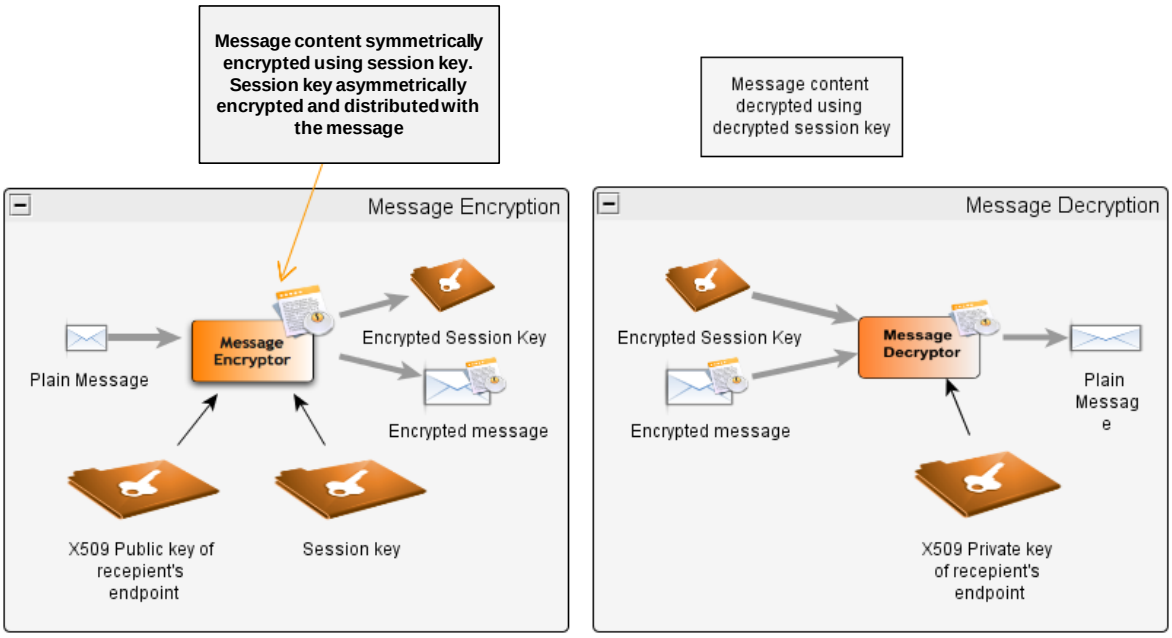


Figure 12: Message encryption and decryption

277 The content of the message (i.e. the document) is encoded with a randomly generated
278 session key, which is then itself encoded with the public key of the encryption certificate of
279 the recipient. The encoded key is transported together with the message.

280 The receiver decodes the key with the private key of his encryption certificate, and then uses
281 the key to decode the document.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

2.7.4 NON REPUDIATION

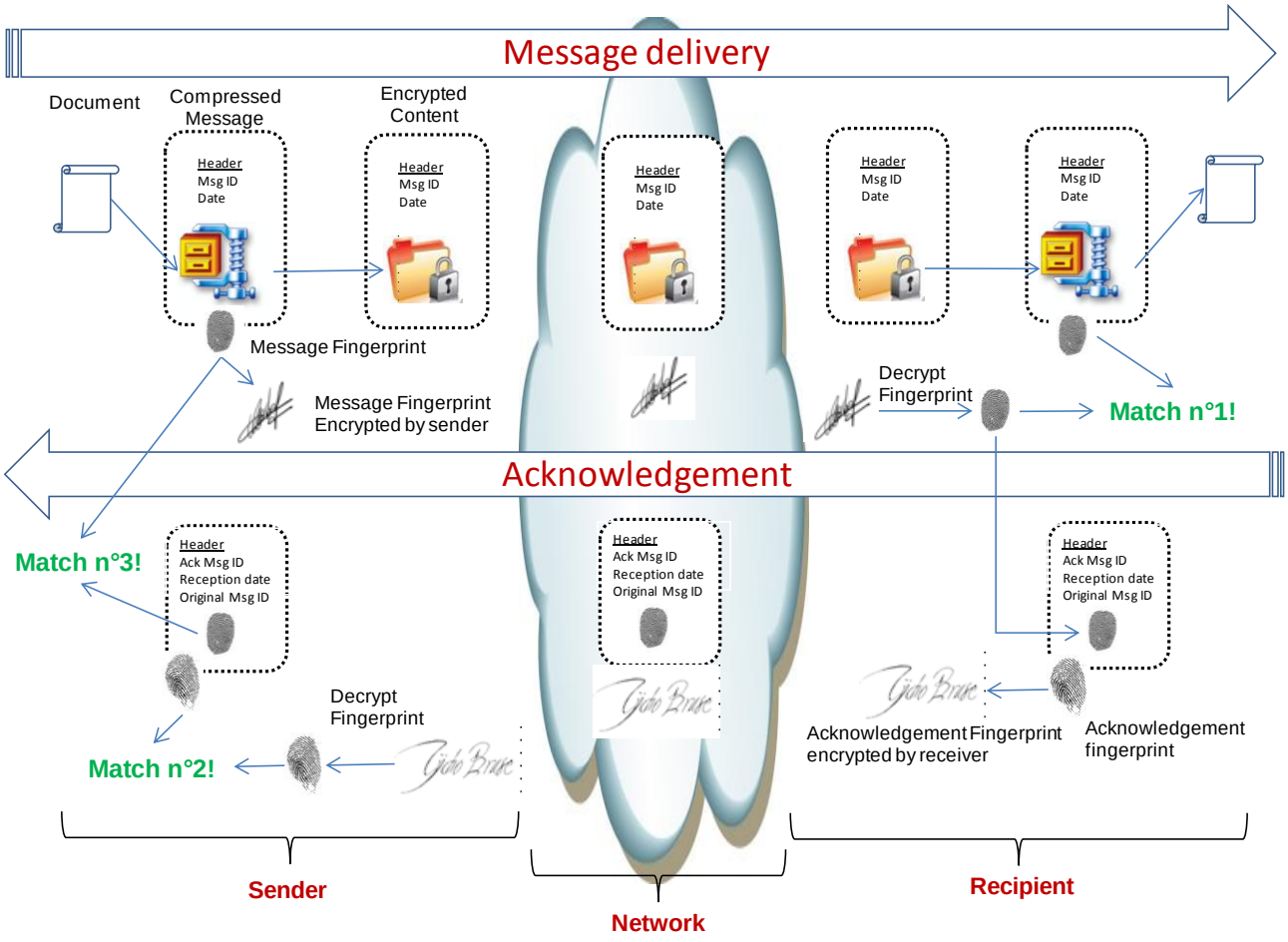


Figure 13: Non repudiation

Message delivery

The message fingerprint identifies uniquely the document together with some header information, such as the message unique ID (*MsgID*) and the sending date and time. The document may have been previously compressed.

Both the fingerprint and the document are encoded and transported to the recipient. The fingerprint is encoded in a way that uniquely identifies the sender (signature), and the document in a way that only the recipient can read it (encryption).

The recipient decodes both the fingerprint and the document. He verifies (match n°1) that the fingerprint, which he can regenerate from the message, matches the transported signed fingerprint (signature verification). Then he stores the decoded message and the encoded fingerprint. Both elements together with the signing certificate prove that the message was sent by the sender.

Acknowledgement

The recipient sends back a new message, the acknowledgement, using a similar process. The new message contains the unique ID of the original message *(Original Msg ID)* and the attached document is the fingerprint of the original message[5].

The acknowledgement is signed but not encrypted and transported to the sender of the original document.

When he receives the acknowledgement, the sender verifies (match n°2) that the acknowledgement was sent by the recipient (signature verification). He also verifies that the acknowledgement document is the original message fingerprint (match n°3).

The set, composed of the original message, the signed acknowledgement and the signing certificate, proves that the recipient received the original message.

[5] There are several acknowledgements during the message delivery; the one referred here is sent by the Endpoint of the recipient party after it correctly receives the message (n°6 in Figure 19).

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3 COMPONENTS' FUNCTIONS

3.1 ROUTING MESSAGES

A message shall be transported from the sender's BA (Business Application) to the recipient's BA using the following route.

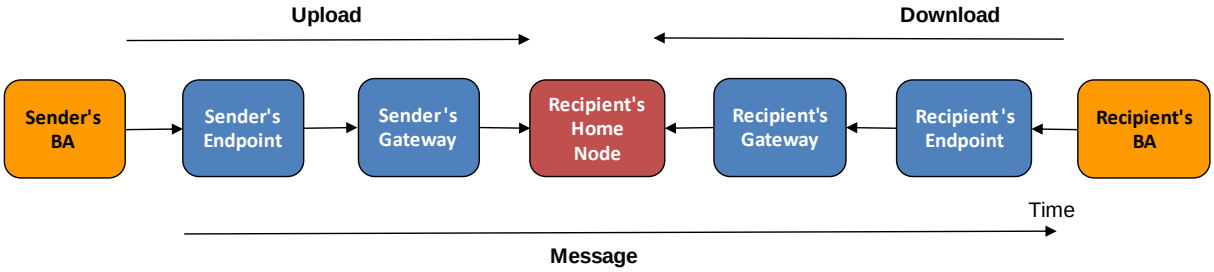


Figure 14: Delivery route of a business-message

The message is composed by the sender's Endpoint with information and document provided by the sender's BA. Then the message is transferred from component to component (from left to right in the figure) until the recipient's BA.

The message-content (also referred as the message-payload) is the document provided by the sender's BA. The composed message contains additional information (i.e. a header) used for security, routing and delivery tracking.

The arrows in "Figure 14" represent the IP connections between the components and each arrow goes from a client to a server. Thus, a message is uploaded (*or sent, or going out*) on the way from sender's BA to the recipient's Node. It is downloaded (*or received, or coming in*) on the way from the Node to the recipient's BA. "Transfer" is the generic word used to mean either upload or download.

The Node, which the message goes through, shall be the home Node of the recipient's Endpoint. Note that a reverse going message between the two BAs will not use the reverse route if sender and recipient have different home Nodes.

There are two types of messages:

- Business-message – is a message composed by the sender's Endpoint from a send request initiated by a sender's BA. The goal of MADES is to transport such business-messages to the requested recipient's Endpoint.
- Acknowledgement – is an ancillary message used for tracking the end-to-end delivery process of a business-message. The BAs do not know about those internal acknowledgements, but a BA can request the Endpoint about the delivery status of a previously sent business-message.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.2 COMPONENT AND MESSAGE UNIQUE IDENTIFICATION (ID)

Each component shall have a unique ID in a MADES network. The identification scheme is a network governance issue.

- The “component ID” (also referred as “component code”) is used to identify the component when exchanging with other components.
- A BA shall use an Endpoint ID to identify the recipient when sending a document. Conversely the sender Endpoint ID is provided to BA together with a received document.
- The IDs of the sender and the recipient are included in the header of each message.

When a component composes a message, it shall identify it with a UUID (Universal Unique Identifier) — as defined in IETF RFC 4122 (http://www.ietf.org/rfc/rfc4122.txt).

Note: When delivering a document, a recipient’s Endpoint supplies the BA with a guaranteed (i.e. authenticated) sender’s identity: the component ID of the sender’s Endpoint. However the sender’s identity is also often included within the document itself, and it is up to the BA that analyses the document to check that both identities match.

3.3 BUSINESS-TYPE OF A BUSINESS-MESSAGE

A MADES network may support multiple and concurrent business processes.

A party “P” having an Endpoint connected to the network can operate several BAs, implementing functions to support internal activities and exchanges with others parties in accordance to the roles he plays in the business processes.

The BAs request the Endpoint to send documents to other parties. The Endpoint supports concurrent requests from the BAs.

Other parties, while fulfilling their own roles in these business processes, may also send some documents to party “P”. For dispatching correctly the received documents between BAs, each BA may indicate a business-type when requesting for downloading a possibly newly received document.

So the business-type is mandatory text information provided by a sender’s BA, included in the header of the business-message, transported with the message to the recipient’s Endpoint, and used by a recipient’s BA to retrieve the only documents it shall process.

Each party is free to organise how he architectures activities, functions and BAs in his own Information System, in a way transparent to the other parties. However the business-types must be agreed between all parties as part of the overall information exchange design[6].

[6] • Business-types can be compared to port numbers. Competing applications in a machine use different port numbers so that received information can be correctly routed. Used port numbers have to be agreed between parties (e.g. 21:FTP, 22:SSH, 25:SMTP) but they are not part of IP protocol which accepts any number.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.4 DELIVERY-STATUS OF A BUSINESS-MESSAGE

The delivery process of each business-message is fully tracked. Tracking means that the components taking part in the routing process notifies the sender's Endpoint with events about the message. The reported events are:

- Delivery event — notifies that the business-message has been either:
  - ✓ transferred to a component; i.e. the component confirms it received the business-message ("Transfer confirmation" is defined in § 3.8).
  - ✓ accepted by a component; the component confirms that it received the business-message and that the message successfully passed validation. It also means that the message is ready to be transferred to the next component on the route to the recipient's Endpoint ("Acceptance" is defined in § 3.9)
- Failure event: — notifies that a business-message cannot be delivered because:
  - ✓ the message was rejected, for it fails the validation;
  - ✓ or an unrecoverable error occurred when processing the message.

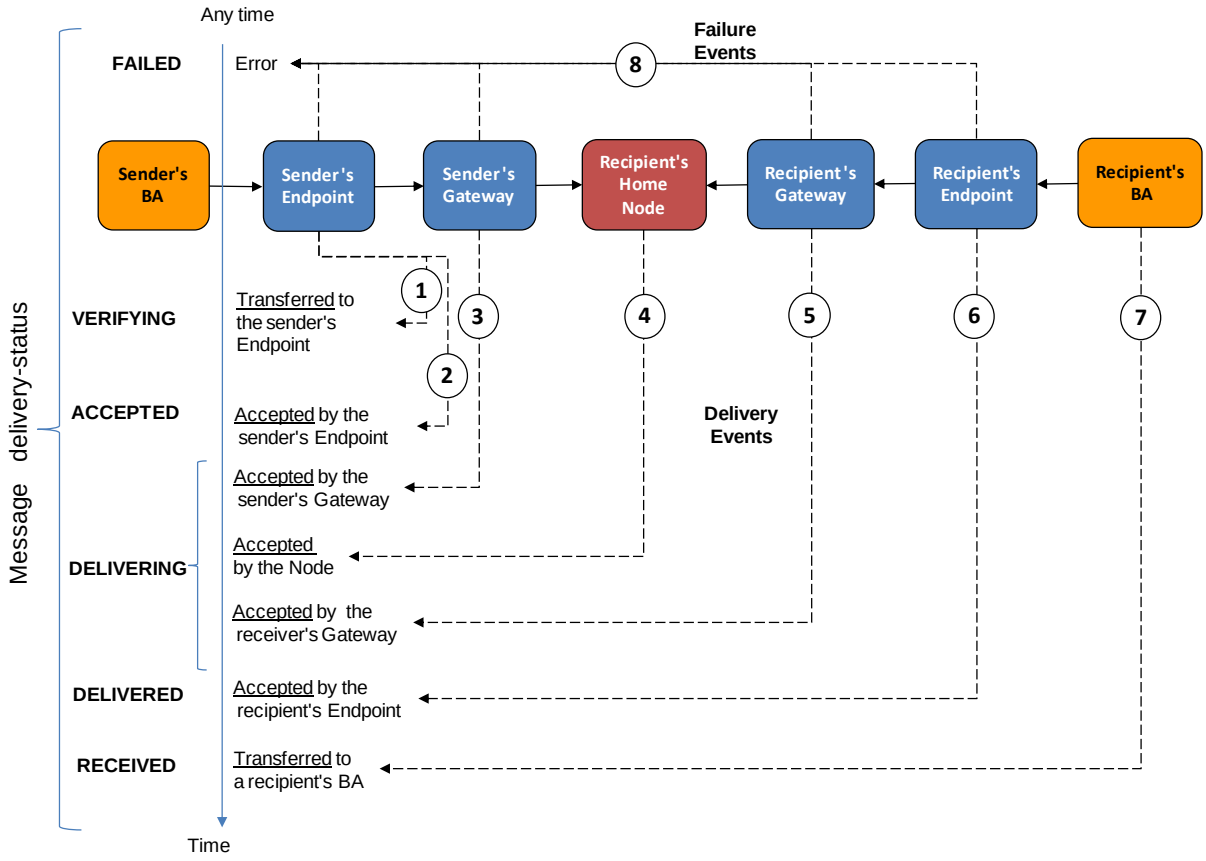


Figure 15: Reported events during the delivery of a business-message

The figure shows the possible events and the components that issue them.

The delivery-status of a business-message expresses the knowledge of the sender's Endpoint about the message delivery. The status can be requested at any moment by a sender's BA providing the message ID returned by the sender's Endpoint when the message was sent. The possible values are:

Message Delivery-status	Notified event
VERIFYING	The business-message has been transferred to the sender's Endpoint. Some additional checks are in progress before the Endpoint may accept it. E.g.: the Endpoint is waiting for signature by an external signing device (see § 3.17.3), or is waiting for the encryption certificate requested to the Node directory.
ACCEPTED	The business-message has been accepted by the sender's Endpoint. Conditions are met to transport the message in the network.
DELIVERING	The business-message has been accepted by an intermediate component (Gateways, Node) between the sender's Endpoint and the recipient's Endpoint. The list of components where the message has already been transported is available — see § 5.2.2.3, output attribute: "*Message trace path*".
DELIVERED	The business-message has been accepted by the recipient's Endpoint.
RECEIVED	The business-message has been transferred to a recipient's BA.
FAILED	The message delivery has failed. So, the business-message will not be transported any further.

Note: the acknowledgement, which notifies that a message has been accepted by the recipient's Endpoint or transferred to a recipient's BA, does not and shall not mean more that the document (i.e. the content of the message) has been technically and securely delivered to the Endpoint of the Information System (IS) of the recipient party. So far, the content of the document has not been analyzed. The probable and further analysis may result in a "functional acknowledgment", the document being then accepted or rejected according to the business rules. Such a functional acknowledgement can even be a new document that the MADES network will be entrusted to deliver as a new business-message to the sender of the original document.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.5 COMMUNICATION BETWEEN COMPONENTS

To communicate, a client component establishes a secured communication channel with a server component, and then issues requests through the channel.

The server component validates the request and replies. The client component receives back a request status and validates the reply.

3.5.1 ESTABLISHING A SECURED COMMUNICATION CHANNEL BETWEEN TWO COMPONENTS

A request from a client component to a server component shall only be processed after the client has established a secured (i.e. encrypted) communication channel with the server;

The communication channel shall be secured using the HTTPS (HTTP over TLS) protocol. So each peer, either client or server, verifies that the other is a valid and trusted network component — see § 3.17.7.

A client component shall be able to connect to a server component through a network proxy.

An Endpoint administrator shall be able to configure a primary and a secondary URL to connect to a Gateway.

A Gateway administrator shall be able to configure a primary and a secondary URL to connect to the home Node.

A Gateway may connect to any Node for uploading business-messages and acknowledgements addressed to a recipient's Endpoint registered with the connected Node. The Gateway shall request the "routing information" by its home Node directory — see § 5.3.4.3.

The Node URLs in directory should rather contain FQDNs (Fully Qualified Domain Names) than IP addresses to ease integration with the network architecture constraints of the parties.

About primary and secondary URLs: the Nodes are key components and thus require high availability. Availability techniques may vary, and redundancy or switch-over mechanism may not be seamless to other components. So a Node administrator may provide two URLs to access his Node. Consequently, the components that connect to a Node shall implement a mechanism to dynamically select the one URL which gains effective access.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.5.2 TOKEN AUTHENTICATION OF THE CLIENT COMPONENT

Apart for the Node-Node synchronization, the server shall always first identify the client, i.e. know its component ID to authorise the requests.

To do so, the client shall request the server for an authentication-token providing its own component ID — see § 5.3.2.

The server provides back a token which:

- is a randomly generated string (e.g. a UUID).
- has a limited validity returned to the client. The later has to request for a new token when expired or before the expiration time.

For every subsequent request (e.g. message transfer, directory query), the client shall always provide the server:

- the authentication-token;
- the signed authentication-token — “signed” means that the hash of the token is encoded using the RSA algorithm — see § 3.17.1 ;
- the ID of the authentication certificate used for signing the authentication-token.

For every received request, the server shall process the following checks:

- the authentication-token is a known and not expired token;
- the certificate used to sign is a valid and non-revoked certificate owned by the client — see § 3.17.9;
- the signature of the authentication-token is correct.

Note: Such a token based client authentication is neither part of nor linked to the TLS authentication, and thus not constrained by specifics of software products used for the implementation of the component (e.g. web servers, applications servers).

3.5.3 REQUESTS’ AUTHORIZATION

A server Node shall reject a request for downloading messages or a request on directory if the client component is not one of its registered components.

A server Gateway or a server Node shall reject a transfer request (download or upload) when it is already and concurrently processing the same request for the same client — see § 3.8.

To correctly process some requests issued by Endpoints and relayed by Gateways, a Node needs to identify the first (i.e. to know the IDs), because a wrong ID could disrupt the delivery behaviour[7]. For such requests[8], the Endpoint shall provide its own component ID signed with the authentication certificate. This is very similar to a password authentication of the Endpoint by the Node over strongly secured channels (SSL + Token). In case the authentication fails, the Node shall reject the Endpoint request.

[7] E.g. Downloading messages (see § 5.3.3.3) using an incorrect ID may result in hijacking messages.

[8] Concerned services are: *DownloadMessages*, *SetComponentMversion*.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.5.4 REQUEST/REPLY VALIDATION

The server shall validate data of any request and the client shall check the status and validate data of any reply.

Validation prevents for foreseeable errors to occur and shall include:

1. Check that all mandatory request/reply elements are set.
2. Checks that all set elements do not contain any illegal characters, have the expected format and size, and have values in expected list or range.
3. Check that the combination values of elements forms a valid set.

In case the request (or the reply) is a message transfer, the validation by the target component shall include any additional check to ensure that the transferred messages can be durably stored — see § 3.6 (e.g. the size of the message-content does not exceed the maximal allowed size).

3.6 STORING MESSAGES IN COMPONENTS

A component shall contain an internal message-box where it durably stores messages. Durable (or persistent) means that the message shall be recovered after a software crash or a hardware failure, when the component restarts (reboot or switch to a backup component in a redundant architecture).

The stored information about a message (either business-message or acknowledgement) shall be: the content, the header.

The Endpoints shall store the compressed (if requested – see § 3.14.1) but non-encrypted content of the message. The stored header shall include the message signature.

Within the message-box, a message shall be associated with additional information only used locally by the component:

- Transfer timestamp — set by the component when the message is created/stored in the message box, and used for priority management — see § 3.13.
- State — see § 3.7 for possible values and lifecycle.
- Priority — see § 3.13.
- Receive timestamp (only used for a business-message), the time when the message was accepted by the recipient's Endpoint — set when processing the acknowledgements of the message — see § 3.10.4

A component administrator shall be able to configure a purge strategy for each business-type. A purge strategy indicates how the component manages a business-message that has reached the final state (see § 3.7). Possible strategies should include:

- Delete only the message-content (document).
- Delete the whole message and acknowledgements.
- Never delete the message.

A component administrator shall be able to (long term) archive messages and then delete the correctly archived messages from the message-box.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.7 LIFECYCLE OF A MESSAGE STATE WITHIN A COMPONENT

479 A business-message has a local state in every component that processes it, and all these
480 states do not have the same values at the same time. The state is not transported data; it is
481 not part of the message header. The lifecycle of the state of a business-message within a
482 component is shown in the figure:

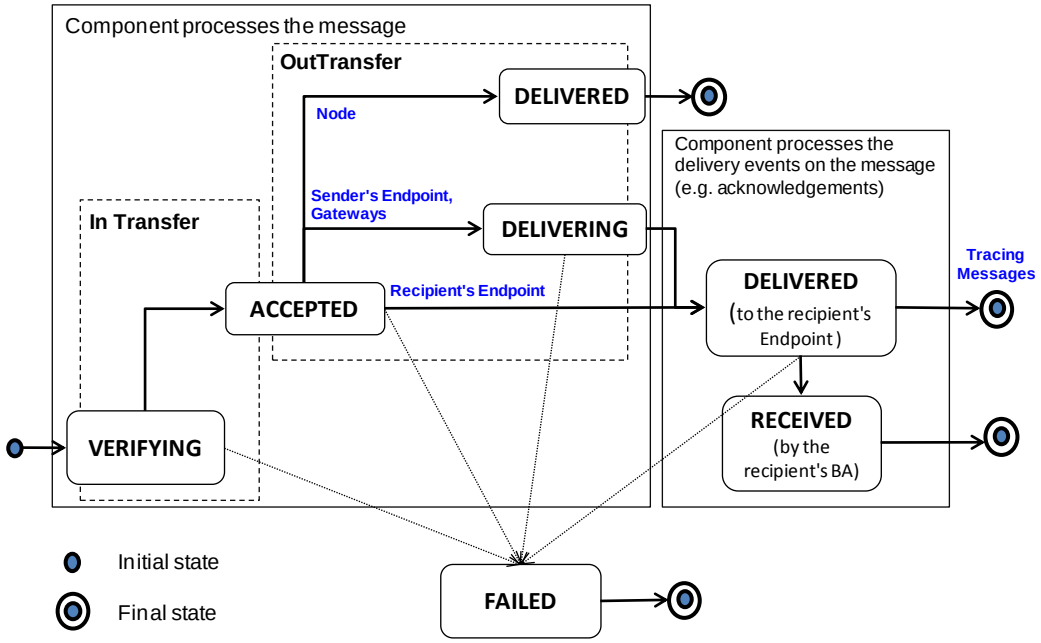


Figure 16: Lifecycle of the local state of a business-message within a component

483 Possible states of a business-message are the followings:

Business-message State	Description
VERIFYING	The successful transfer of the business-message in the component has been confirmed to the component which sent it and, before it may accept it, the message is currently passing some validation checks or pending (e.g. waiting for an external certificate to perform security operations such as signature/encryption). Note: a business-message may never be in this state if there is no validation by the component or if this validation is processed before the transfer confirmation (e.g. Node).
ACCEPTED	The business-message has been accepted by the component, and is pending for transfer to the next component on the message route.
DELIVERING	The business-message has been successfully transferred to the next component.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Business-message State	Description
DELIVERED RECEIVED FAILED	After the business-message has been transferred to the next component, the message state is set to the status of the acknowledgements coming back and which inform about the message delivery (see § 3.10). A business-message in the FAILED state is not delivered. A component shall set the business-message state to FAILED when it sends a failure-acknowledgement for the message.

Notes:

- After a message has been successfully transferred (downloaded) from a Node, the state within the Node shall move to DELIVERED which is the final state (and not to DELIVERING). The reason is the following: If a Node is not the home Node of the sender's Endpoint of a business-message, no acknowledgement will ever inform about the rest of the message delivery.
- When a message is accepted by the recipient's Endpoint, the state within the Endpoint shall be directly set to DELIVERED, because the message has reached the destination Endpoint and does not have a next component.
- The delivery-status of a business-message, as defined in § 3.4, is the local state of the message in the sender's Endpoint.

3.8 TRANSFERRING A MESSAGE BETWEEN TWO COMPONENTS (HANDSHAKE)

The transfer handshake is the mechanism which ensures that no message can be lost while passing from a component to another. A component (referred as "target component") that receives a message shall confirm to the sender component (referred as "source component") that the message has been transferred.

A component is responsible for the message delivery from the moment it sends the transfer confirmation to the previous component until the moment it receives the transfer confirmation from the next component.

The target component confirms a message transfer to tell the source component that it took responsibility for the message, and that it should not transfer it again. It means that either:

- the message has been stored in a durable way,
- the processing of the message has generated an error that has been logged (e.g. message inconsistency).

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

The handshake mechanism differs whether the transfer is an upload or a download, as shown in the following figures. When downloading, the target component initiates the request and an additional request is used to confirm the transfer to the source component.

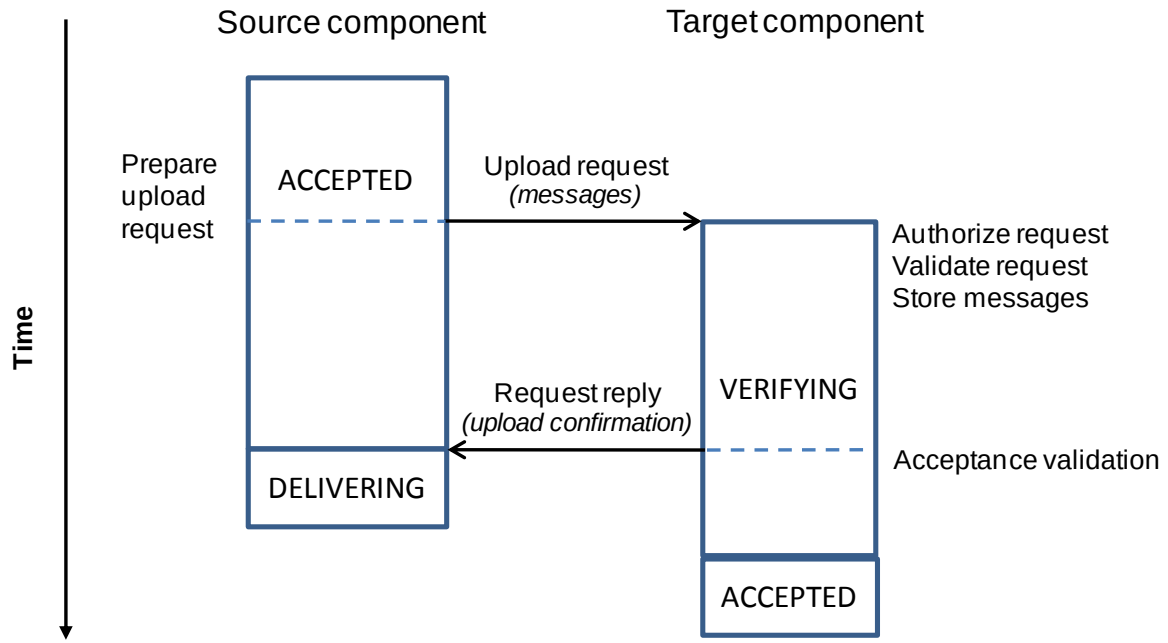


Figure 17: Transfer handshake when uploading of a message

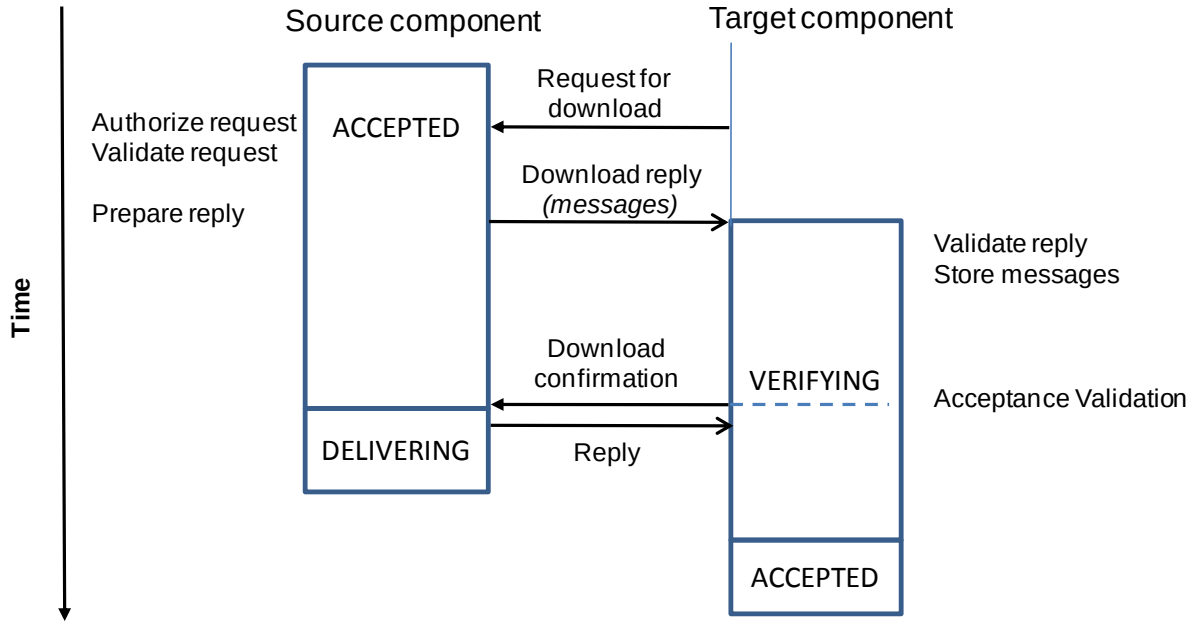


Figure 18: Transfer handshake when downloading of a message

Note: the use of the VERIFYING state is a component internal design issue. A target component can confirm the transfer and set the message in the VERIFYING state before asynchronously processing the acceptance checks. A target component can confirm the transfer after it processes synchronously the acceptance checks, so the message state is directly set to ACCEPTED (or FAILED) — see § 3.9.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

The handshake mechanism applies whether a transfer request contains one or several messages. Actually, the MADES interfaces for uploads and downloads can transfer bulk messages, mixing business-messages and acknowledgements — see § 5.3.3. When multiple messages are transferred simultaneously, the confirmation shall apply to all the transferred messages. Note that the BAs can only transfer (send or receive) business-messages one by one with their Endpoint.

A server component shall not authorize a transfer request from a client component while the same request from the same client is currently being processed. This is necessary to fulfil the correct delivery sequence of two messages with the same business-type — see § 3.13. The bulk transfer is intended to gain performance without the use of concurrent requests.

The source component shall change the message state to the next state (generally DELIVERING) after it receives the transfer confirmation.

When the connection between the components is established or recovered after a failure, the source component shall transfer all pending messages in the ACCEPTED state. Note that it may happen that some of those messages have already been transferred, that the target component already sent the confirmation, but that the source component did not receive it or failed while processing it. So the target component may receive an already existing message (recognized with the message ID). In this situation, it shall then just confirm the transfer and log this duplicate transfer event.

3.9 ACCEPTING A MESSAGE

A component shall accept a transferred message after it passed the validation checks described in the next table.

Component	Validation checks
Sender's Endpoint	The transferred message can only be a business-message: ✓ Existence of the recipient's Endpoint. ✓ Availability of the encryption certificate of the recipient's Endpoint, i.e. successfully retrieved from directory cache or home Node directory. ✓ Successful signature of the message. ✓ Note: the business-message shall be compressed (if requested) while received by the Endpoint, and it shall be encrypted when uploaded to the Gateway.
Sender's Gateway	*(no validation check)*
Node	The transferred message can be a business-message or an acknowledgement: ✓ The recipient's Endpoint has registered with the Node. ✓ The sender's Endpoint exists in the directory and owns the certificate used to sign the message. ✓ The certificates used for signing and encryption (if encrypted) exists and are not revoked (see § 3.17.9).

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Component	Validation checks
Recipient's Gateway	*(no validation check)*
Recipient's Endpoint	The transferred message can be a business-message or an acknowledgement: ✓ Successful decryption of the content, when encrypted. ✓ Successful verification of the signature, when signed. ✓ When the message is an acknowledgement notifying the event n°6 (see Figure 15), successful match between the acknowledgement content and the original message fingerprint (hash). ✓ Note: a compressed business-message shall be uncompressed when transferring to a recipient's BA.

When a business-message is accepted, the following operations shall be processed as a transaction[9]:

- The message is updated (e.g. decrypted content; change of the local state according to the lifecycle).
- The component notifies the related delivery event – see § 3.10.2.

When a business-message is rejected, the component shall notify a failure-event and update the message as a transaction.

When an acknowledgement is rejected, the component shall log the error and set the acknowledgement state to FAILED; this stops the delivery.

3.10 EVENT MANAGEMENT

3.10.1 ACKNOWLEDGEMENTS

A component can notify an event which occurs when delivering a business-message, by sending an acknowledgment to the sender's Endpoint of the message. The business-message on which the event occurs is referred as the original message, and its ID shall be included in the acknowledgement header.

An acknowledgement shall be routed and delivered using the same transfer (upload and download) mechanism as the business-messages, without being acknowledged itself.

An acknowledgement shall have the same business-type as the original message.

The content of an acknowledgement shall never be compressed or encrypted.

[9] In the whole specification a "transaction" means an operation that must succeed or fail as a complete unit and cannot remain in an intermediate state.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.10.2 NOTIFYING EVENTS

553 "Notifying an event" on a message means either:

554 • Sending an acknowledgement containing event information.

555 • If the event occurs in the sender's Endpoint, event information is locally stored, so it
556 can be reported to the sender's BA requesting for the message delivery status.

557 The event issuer shall update the message according the event (e.g. the message state).

558 Previous operations shall be realized as a transaction, and the issuer shall log the event.

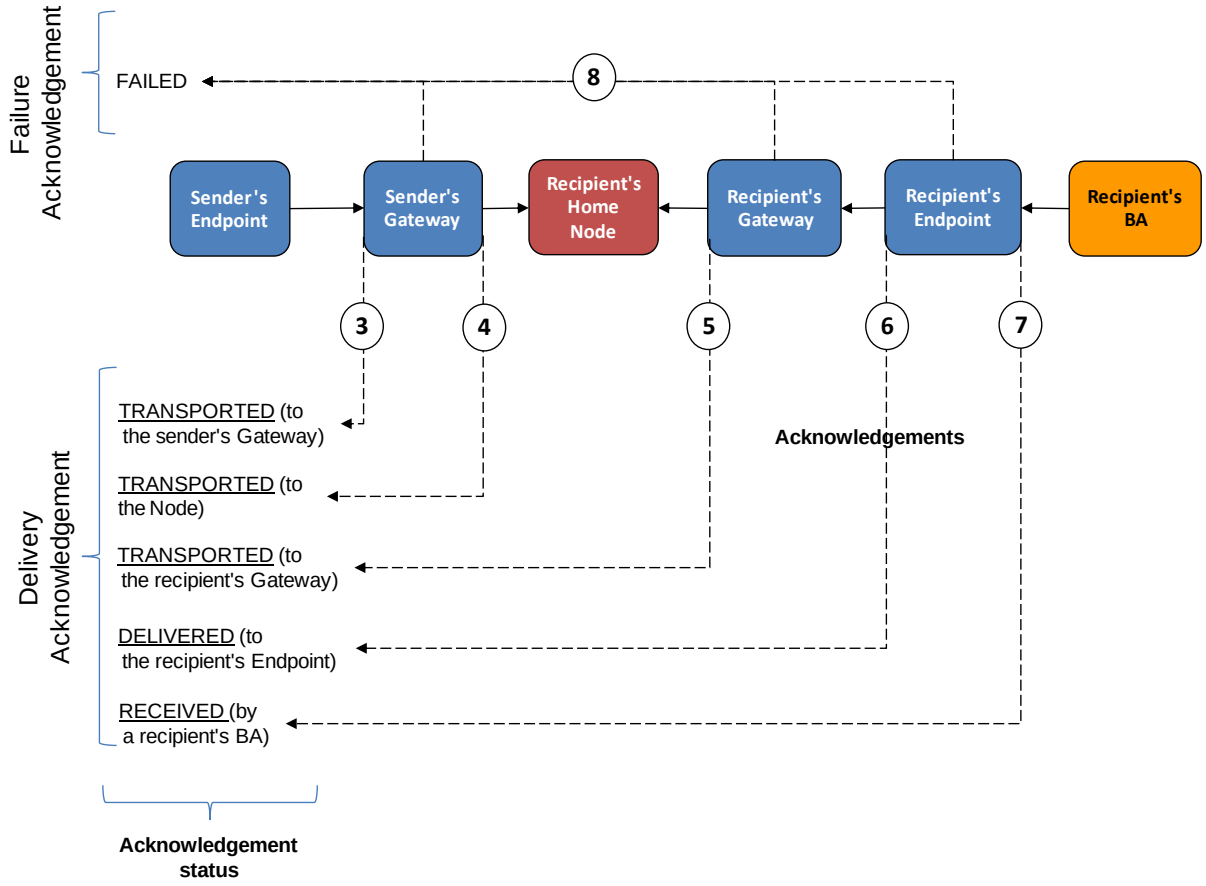


Figure 19: Acknowledgements along the route of the business-message

559 The figure shows the issuers and the events notified with acknowledgements. The events are
560 numbered as in Figure 15.

Event	Event characteristics
1	Status: VERIFYING Issuer: Sender's Endpoint Acknowledger: None (The event is internal to the Sender's Endpoint and does not generate an acknowledgement)

Event	Event characteristics
2	Status: ACCEPTED Issuer: Sender's Endpoint Acknowledger: None (The event is internal to the Sender's Endpoint and does not generate an acknowledgement)
3	Status: TRANSPORTED Issuer: Sender's Gateway Acknowledger: Sender's Gateway ✓ Content: irrelevant but as least one character. ✓ Internal type: DELIVERY_ACKNOWLEDGEMENT ✓ Signed: No ✓ Original message state: ACCEPTED
4	Status: TRANSPORTED Issuer: Recipient's Node Acknowledger: Sender's Gateway ✓ Content: it shall contain the "Node component ID" to report to the sender the Node the message was sent to (unless it would not know). ✓ Internal type: DELIVERY_ACKNOWLEDGEMENT ✓ Signed: No ✓ Original message state: DELIVERING Comment: A Node never sends an acknowledgement because it could not be delivered if it is not the home Node of the sender's Endpoint of the original message. The reason is that the Gateway of that the sender's Endpoint won't connect to the recipient's Node for downloading messages, including acknowledgements. Thus the Node delegates to the sender's Gateway the issuance of the acknowledgement by notifying in the upload response whether it accepts or rejects the business-message.
5	Status: TRANSPORTED Issuer: Recipient's Gateway Acknowledger: Recipient's Gateway ✓ Content: irrelevant but as least one character. ✓ Internal type: DELIVERY_ACKNOWLEDGEMENT ✓ Signed: No ✓ Original message state: ACCEPTED
6	Status: DELIVERED Issuer: Recipient's Endpoint Acknowledger: Recipient's Endpoint ✓ Content: the non-encoded message fingerprint (hash) of the original message — see § 3.14.2. ✓ Internal type: DELIVERY_ACKNOWLEDGEMENT ✓ Signed: Yes ✓ Original message state: DELIVERED Comment: State and status are set to DELIVERED because the acknowledger is the recipient's Endpoint of the original message.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Event	Event characteristics
7	Status: RECEIVED Issuer: a recipient's BA Acknowledger: Recipient's Endpoint ✓ Content: irrelevant but as least one character. ✓ Internal type: RECEIVE_ACKNOWLEDGEMENT ✓ Signed: Yes ✓ Original message state: RECEIVED Comment: A recipient's BA (not a MADES component) delegates to the recipient's Endpoint the issuance of the acknowledgement notifying that the successful transfer of the business-message.
8	Status: FAILED Issuer: any component except the Node. Acknowledger: any issuer except the sender's Endpoint ✓ Content: an English readable description of the encountered error or the reason why the message was not accepted. ✓ Internal type: FAILURE_ACKNOWLEDGEMENT ✓ Signed: Only if the issuer is the recipient's Endpoint. ✓ Original message state: FAILED Comment: The event is referred as the "failure-event". In case a failure-event occurred in the sender's Endpoint it processes it internally and does not send an acknowledgement.

561 Legend:

Characteristic	Description
Status	The value to set to the "*state"* element of the "*trace*" item (see § 5.5.15) reporting the event to the sender's BA through the *CheckMessageStatus* service - see § 5.2.2.3.
Issuer	The component that notifies (issues) the event.
Acknowledger	The component that sends the acknowledgement, possibly none or possibly different from the issuer.
Content	The content of the acknowledgement message.
Signed	Whether the acknowledgement message is signed or not.
Internal type	The value to assign to the *internalType* element of the acknowledgement – see § 5.5.10
Original message state	The value to set to the local state of the original message in the issuer component when issuing the event.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.10.3 LIFECYCLE OF AN ACKNOWLEDGEMENT

562 The possible values for the local state of an acknowledgement within a component are a
563 subset of the states of a business-message.

564 Unless signed using an external device (see § 3.17.3), an acknowledgement is created in the
565 ACCEPTED state, otherwise in the VERIFYING state.

Acknowledgement State	Description
VERIFYING	The acknowledgement has been created by the component or successfully transferred to it, and a signature operation is currently processed or pending (e.g. waiting for the external device to be signed, or waiting for the external certificate to verify the signature).
ACCEPTED	The acknowledgement is pending for transfer to the next component towards the destination Endpoint.
DELIVERED	The acknowledgement has been successfully transferred to the next component or has reached the destination, i.e. the sender's Endpoint of the original message.
FAILED	The component encountered an unrecoverable error when processing the acknowledgement. The acknowledgement will never be transferred to another component.

3.10.4 PROCESSING A TRANSFERRED ACKNOWLEDGEMENT

566 A component shall always accept a transferred acknowledgement.

567 When processing a transferred acknowledgement:

568 • The component shall log the event notified by the acknowledgement.

569 • In case an unrecoverable or an acceptance error (see § 3.9) occurs, the component
570 shall transitionally set the acknowledgement and the original message to the FAILED
571 state, and log the error. This stops the acknowledgement delivery.

572 • Otherwise the component shall transitionally:
573 ✓ update the state of the original message according to the event in conformance
574 to Figure 15;
575 ✓ when the event is n°6, set the "receive timestamp" of the original message to the
576 time the acknowledgement was created (*generated* item – see § 5.5.10).

577 Notes:
578 • The original message may not exist in a Node for it was delivered through another
579 Node. The acknowledgement shall then be correctly processed and route.
580 • The original message may be in the ACCEPTED state. This may happen when the
581 message was transferred and the component did not receive the confirmation. When
582 the connection is back, it may receive acknowledgements before the message is
583 transferred again.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.11 MESSAGE EXPIRATION

The message expiration is a mechanism to notify the sender's BA that a business-message has not been delivered in the due time to the recipient's Endpoint. When the time limit is exceeded, the sender's Endpoint changes the state of the message to FAILED.

The expiration time of a business-message is the time limit when the sender's Endpoint will declare that the message delivery has failed, because it has not received the acknowledgement notifying that the message was accepted by the recipient's Endpoint (event n°6).

Setting the expiration time of a message:
An Endpoint administrator shall be able to configure maximum durations for the delivery of the business-messages as:

- duration values associated to the business-types;
- a non zero and positive default duration value.

The expiration time is part of the header of a message — see § 5.5.10, *expirationTime*. The time count shall start when the sender's Endpoint confirms the transfer of the business-message (event n°1). The expiration time shall be set by the sender's Endpoint according to the business-type of the message. Otherwise the default duration value shall be used.

The expiration time of an acknowledgement is the expiration time of the original message.

Looking for the expired messages:
Each component shall cyclically look for the expired messages either business-messages or acknowledgements. A message expires when the expiration time is past and the local state is not amongst DELIVERED, RECEIVED or FAILED.

The sender's Endpoint shall notify the expiration of a business-message using an event-failure. Otherwise the component shall set the local message states to FAILED and log the expiration (date, message ID, sender, recipient, sending time, expiration time).

Note: the default value for maximum delivery duration is a general mechanism to set to FAILED the state of the messages whose delivery cannot be processed for whatever reason, ensuring then that they will not be forever delivering (i.e. "zombie" messages').

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.12 CHECKING THE CONNECTIVITY BETWEEN TWO ENDPOINTS (TRACING-MESSAGES)

A tracing-message is a business-message used to check end-to-end connectivity between two Endpoints using the message tracking process. The message header contains a special type for a tracing-message (TRACING_MESSAGE — see § 5.5.11).

A BA can request to process a connectivity test with any Endpoint. The sender's Endpoint shall then compose and send a tracing-message to deliver to the required destination Endpoint.

To check that the tracing-message reached the recipient's Endpoint, the sender's BA can check its delivery status, as for any business-message.

The business-type and the content of a tracing-message are irrelevant but shall have at least one character. As any business-message, a tracing-message is signed and the content is encrypted. So the tracing-message delivery success includes the checks of the certificates' set-up and processing.

The header of the acknowledgements whose original messages are tracing-messages also have a special type (TRACING_ACKNOWLEDGEMENT — see § 5.5.11).

Because no recipient's BA will ever request for the tracing-message, the final state of a tracing-message is DELIVERED in all components — see § 3.7.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.13 ORDERING THE MESSAGES (PRIORITY)

A component administrator shall be able to configure priority values according to the business-types, and to configure a default priority value for unknown business-types.

A business-message shall have the priority configured for the business-type if defined; otherwise the default priority.

The component shall process pending messages and elaborate the transferred list of messages using the following order:

1. A message with higher priority is processed first.
2. If two messages have the same priority, the one that was first transferred by the component is processed first — see “Transfer timestamp” in § 3.6.

Remarks:

- The message priority is local to a component. It may differ between components and is not transported information.
- Assume that two messages (M1 and M2) of the same business-type are sent in this order by BA1 to BA2. If BA2 receives both messages, M1 shall be received first. Whatever priority is configured for the business-type by each component, the delivery order shall remain unchanged.
- An acknowledgement has the same priority as the original message, because it has the same business-type.
- The priority of the tracing-messages may be configurable; otherwise they have the default priority.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.14 ENDPOINT FUNCTIONS

An Endpoint provides interfaces for BAs to send and receive messages in a secure way. An Endpoint shall provide the following functions:

- Communication:
  - Connect to a Gateway using HTTPS.
  - Validate the send-requests from the BAs.
  - Validate the receive-requests from the BAs and provide the received documents.
- Pre-processing the to-send business-messages:
  - Compose the business-messages (e.g. create the message ID, set the expiration time, and compress the content).
  - Check the existence of the recipients and get their encryption certificates by the home Node directory.
  - Generate the message signature, and encrypt the message-content.
- Post-processing the received business-messages:
  - Get the signing certificates by the home Node directory.
  - Decrypt the message-content, verify the message signature, and uncompress the message-content.
- Notifying the events on the message delivery:
  - Send and process the acknowledgements.
  - Verify the signature and the content of the acknowledgements.
- Processing the messages:
  - Upload and download the encrypted business-messages to and from a Gateway.
  - Store the messages in the local message-box.
  - Process the validation checks.
  - Look cyclically for the expired messages.
  - Update the messages' states according to delivery progress.
  - Manage the queues with messages pending for uploading or downloading.
  - Process the messages according to local priority rules.
- Processing the tracing-messages:
  - Validate the connectivity test requested by the BAs.
  - Compose the message.
  - Process the tracing-messages downloaded from the Gateway.
- Requesting the Gateway for directory information:
  - Retrieve other Endpoints signing and encryption certificates.
  - Durably store the used signing certificates of the other Endpoints.
- Replying to the messages status requests from BAs.
- Administration:
  - Synchronize the Endpoint time with a reliable source (recommendation is to use a standard OS mechanism such as NTP, the Network Time Protocol).
  - Install Endpoint and CAs certificates (initial and renewed).
  - Archive and purge the logs.
  - Archive and purge the messages.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.14.1 COMPRESSION

688 The sender's Endpoint shall compress the content of each business-message whose
689 business-type is configured to do so.

690 The Endpoint administrator shall be able to configure the business-types of the business-
691 messages that shall be compressed.

692 The tracing-messages and the acknowledgements shall not be compressed.

693 Compression shall be done using the ZIP algorithm.

694 Compression means that the message-content is encoded and that following metadata[10]
695 shall be added to the message header — see § 5.5.10:

Metadata Attribute Name	Metadata Type	Description
Compression	BOOLEAN	Value := true (i.e. the message was compressed)

696 The header of a non-compressed message may also contain the metadata with the attribute
697 value set to "false".

3.14.2 SIGNING

698 The signing principles are presented in § 3.17.1.

699 Only the Endpoints sign the messages. A sender's Endpoint shall sign every business-
700 message. The recipient's Endpoint shall sign the acknowledgement notifying event n°6.

701 An Endpoint shall verify the signature of every signed message that it receives. A message is
702 signed if it contains the signature metadata (see further).

703 Signature algorithm shall be RSA-SHA (IETF RFC 3110 - http://www.ietf.org/rfc/rfc3110.txt).
704 The signature format shall comply with the "XML Signature Syntax and Processing standard"
705 (http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/).

706 An Endpoint shall encode the message hash using the private key of the signing certificate.

707 The manifest used to generate the message hash is:

708 Compress (*content*) + *baMessageID* + *extension* + *generated* + *internalType*
709 + *messageID* + *relatedMessageID* + *receiverCode* + *senderCode* +
710 *senderDescription* + *SenderApplication* + *businessType.*

711 Where:
712 • the italic names refer to the attribute of the internal message structure as
713 described in § 5.5.10;
714 • "+" is the binary concatenation of message attributes in UTF-8 encoding;
715 • Note that the content of the message may be compressed or not, according to
716 § 3.14.1, but not encrypted.

[10] « Metadata » refers to the part of the message header named "metadata" — see § 5.5.10.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

717 Signing means that following metadata is added to the message header — see § 5.5.10:

Metadata Attribute Name	Metadata type	value
Algorithm	STRING	Value := SHA-512 (*The algorithm used to generate the message hash*).
Certificate ID	STRING	The ID of the certificate whose private key was used to generate the signature, i.e. to encode the message hash — see § 3.17.2.
Signature	STRING	The message signature compliant with the "XML Signature Syntax and Processing standard". The XML signature document is embedded here as a string. *(An example of an XML signature document is provided in § 5.6.3).*

718 When receiving a message an Endpoint shall check the message was signed, i.e. if the
719 header contains signature metadata, and then:

720 1. Recover the signing certificate from the cache or from the home Node.
721 2. Verify that the certificate was valid when the message was generated (Certificate
722 expiration date-time is a certificate attribute. The generated date-time of a message is
723 part of the message header);
724 3. Verify the XML signature (i.e. the "signature" attribute of the metadata) using the
725 public key of the signing certificate.
726 4. Regenerate the message hash of the received message using the "Algorithm".
727 5. Verify that the hash provided by operations 3 and 4 are equal.

3.14.3 ENCRYPTION

728 The encryption principles are presented in § 3.17.1.

729 The sender's Endpoint shall encrypt a business-message before it is uploaded. The
730 recipient's Endpoint shall decrypt a business-message after it is downloaded.

731 Only the message-content of the business-message shall be encrypted. The
732 acknowledgments shall never be encrypted.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

733 The encryption and decryption processes use a combination of asymmetric and symmetric
734 cryptography.

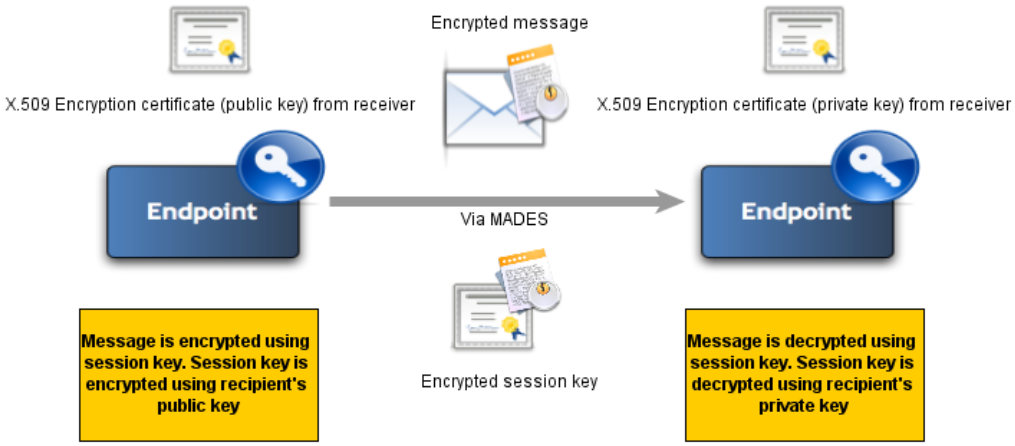


Figure 20: Encryption process

735 The sender's Endpoint shall use the encryption certificate of the recipient's Endpoint. It
736 retrieves it by the home Node.

737 To encrypt the message-content, the Endpoint shall first generate a random symmetric
738 encryption key (called the session key), which is used to encode the content of the message.
739 Then the symmetric key shall be encoded using the public key of the encryption certificate of
740 the recipient's Endpoint.

741 The symmetric algorithm used to encode the message shall be AES (Advanced Encryption
742 Standard) and the key size shall be 256 bits.

743 Encryption means that the message-content is encoded and following metadata shall be
744 added to the message header — see § 5.5.10):

Metadata Attribute Name	Metadata Type	Description
Cipher	STRING	Value := AES-256 (*The algorithm used to encrypt the message-content with the session key*).
Certificate ID	STRING	The ID of the certificate whose public key is used to encode the session key — see § 3.17.2.
Session key	BYTE_ARRAY	The value of the session key encoded using RSA algorithm and the public key of the encryption certificate.

When receiving a message the recipient's Endpoint shall check if the message is encrypted, i.e. if the header contains encryption metadata, and then:

1. Verify that the certificate ID used to encrypt the message is one of the owned encryption certificates;
2. Verify that the encryption certificate was valid when the message was generated (Certificate expiration date-time is a certificate attribute. The generated date-time of a message is part of the message header);
3. Decode the symmetric session key using the corresponding private key of the certificate;
4. Decode the message-content using the decoded session key and the algorithm used for encryption.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.15 GATEWAY FUNCTIONS

When implemented on a separate machine, a Gateway provides means to:

- decouple the WAN domain and the party IT internal domain, acting like a proxy that can be located in a DMZ,
- Implement a single network connection point for several Endpoints.

A Gateway shall provide the following functions:

- Communication:
  - Connect to the home Node using HTTPS.
  - Authorize the HTTPS connections from the Endpoints.
- Notifying events on the message delivery:
  - Send and process the acknowledgements.
- Processing messages:
  - Upload and download the messages to and from the connected Endpoints and the home Node.
  - Store the messages in the local message-box.
  - Look cyclically for the expired messages.
  - Update the messages' states according to their delivery progress.
  - Manage the queues of messages pending for uploading or downloading.
  - Process the messages according to local priority rules.
- Requesting to the home Node directory:
  - Retrieve the URLs of the recipient's Nodes of the messages.
  - Retrieve directory information and certificate for its own needs or as requested by the connected Endpoints.
- Administration:
  - Synchronize the Gateway time with a reliable source (recommendation is to use a standard OS mechanism such as NTP, the Network Time Protocol).
  - Install Gateway and CAs certificates (initial and renewed).
  - Archive and purge the logs.
  - Archive and purge the messages.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.16 NODE FUNCTIONS

784 A Node shall provide the following functions:

785 • Communication:
786 ○ Authorize the HTTPS connections from the Gateways or the other Nodes.
787 ○ Connect to the other Nodes using HTTPS.
788 • Processing the messages:
789 ○ Upload and download the messages to and from the Gateways.
790 ○ Store the messages in the local message-box.
791 ○ Process the validation checks.
792 ○ Look cyclically for the expired messages.
793 ○ Update the messages' states according to the delivery progress.
794 ○ Manage the queues of messages pending for downloading.
795 ○ Process the messages according to the local priority rules.
796 • Directory services:
797 ○ Provide the registered Gateways with the Nodes' URLs and the Endpoints'
798 description and certificates.
799 ○ Request the others Nodes for their reference directory data — see § 3.16.1.
800 ○ Reply to the others Nodes' synchronization requests — see § 3.16.1.
801 ○ Manage directory data and the data version (Dversion).
802 • Administration:
803 ○ Register the Gateways and Endpoints.
804 ○ Generate the certificates for the registered components
805 ○ Import signing and encryption certificates from externals CAs.
806 ○ Revoke the Endpoint and Gateway certificates — see § 3.17.9.
807 ○ Import the synchronization Nodes' List — see § 3.16.2.
808 ○ Synchronize the Node time with a reliable source (recommendation is to use a
809 standard OS mechanism such as NTP, the Network Time Protocol).
810 ○ Archive and purge the logs.
811 ○ Archive and purge the messages.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.16.1 SYNCHRONIZING DIRECTORY WITH OTHER NODES

A Node directory is the master reference for all data regarding a sub-network composed of the Node itself and the registered Gateways and Endpoints.

The synchronization between the Nodes is carried out cyclically or on the Node Administrator demand. Each Node requests the others Nodes for their sub-network data and stores it in its own directory.

The synchronization frequency is defined by the network governance.

Example: the following figure shows the Node A that connects to the Node B, and then to the Node C to obtain their directory data.

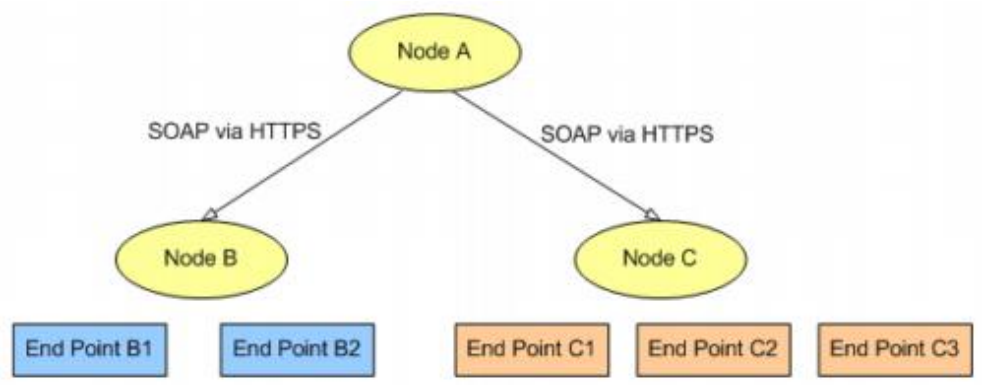


Figure 21: A Node synchronizes with two other Nodes

Each Node shall manage a directory version number, referred as the “Dversion” for the reference data, which increases every time they are updated.

Each Node shall store reference data of the other Nodes and the corresponding Dversion. The Dversion shall always be transferred together with the directory data. When requesting for data of another Node, the client shall provide the Dversion of the remote data that it already possesses, so the reply can just inform that data is already up-to-date. Thus the Nodes may synchronise frequently (e.g. 5 minutes).

Synchronized data shall include:

- information and certificates of all Endpoints and Gateways registered with the Node,
- information and certificates of the Node itself.

After received data has been validated (e.g. check that none of the received component ID already exist in another sub-network), the update in directory shall be a transaction.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.16.2 UPDATING THE SYNCHRONIZATION NODES' LIST

The network Administrator is responsible to build and send to all the Node Administrators the list and the access information of all the Nodes of the network, namely for each Node:

- The Node component ID.
- The Node access URLs (primary, secondary).

The list is a single file, referred as the Node-list file, whose format is described in § 5.4.

A Node Administrator shall import the file to update the Nodes to synchronize with.

- The Node shall process the file as a transaction, i.e. any error (e.g. incorrect format, non-unique component ID, missing certificate and internal error) shall cause the rollback of the whole update process, and the directory data shall remain unchanged.
- The importation process shall ignore information about the current Node which is included in the list.
- The synchronization process to update the current Node directory shall be stopped during the list importation.

After importation, the Node Administrator can restart the synchronization to update the directory with the reference data of the Nodes.

A Node shall memorize the last time it successfully retrieved the data of each Node of the Node-list file. Such information shall be accessible by the Administrator.

Note: A message exchanged between two Endpoints having different home Nodes can only be delivered correctly (including acknowledgements) after the two Nodes have synchronized with each other at least once.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.17 CERTIFICATES AND DIRECTORY MANAGEMENT

3.17.1 DEFINITIONS AND PRINCIPLES

The security of a MADES network is based on a Public Key Infrastructure (PKI). Such infrastructure binds certificates both to the network components and to the parties using the network. Indeed the components cross-check their identities before exchanging information, the sender parties want that the only intended recipients can read the documents, and each party want to authenticate the senders of the documents he received.

Certificates use asymmetric cryptography based on private and public keys. On the contrary of symmetric cryptography, encoding is done using one key and decoding using the other key (which is different, and hence the asymmetry). Where the public key can easily be deduced from the private key, the reverse operation is a very complex mathematical challenge. RSA algorithm is generally used for encoding and decoding.

Encryption:

- A document is encrypted[11] when it is encoded with a randomly generated symmetric key. The key is attached to the document in a secret way, being encoded itself with the recipient's public key.
- To decrypt the document, the recipient shall first decode the "encoded symmetric key" using its private key, and then decode the document with the symmetric key.

Signing:

- A signature is an encoded fingerprint of a list of resources. The list is referred as the signature manifest. The technical word for the fingerprint is a "hash", which is generated via a strong one-way transformation (e.g. SHA-1, SHA-512). The exact manifest of a MADES message is described in § 3.14.2.
- The algorithm used to generate the hash does not require any key, so anyone having the manifest can generate the hash. Building another meaningful manifest generating the same hash is also a complex mathematical challenge. The signature is the hash encoded with the sender's private key.
- Thus anyone having the manifest, the signature and the sender's public key can verify that the manifest is the one that was manipulated by the sender when he generated the signature. The sender cannot repudiate a manifest he signed.
- Signing refers to the full process, i.e. generating the hash and encoding it.
- Verifying a signature includes: regenerating the hash from the manifest, decoding the signature, and checking that both results are equal.

A Certificate Authority (CA) is an entity that issues certificates.

A certificate:

- contains a public key, a name;
- is signed with the private key of the certificate of the issuer CA;

[11] Beware that "Encoding" and "Encryption" are not synonym here. "Encoding" refers to an algorithmic operation, while "Encryption" is the process described here which ensures confidentiality. Both "Encryption" and "Signing" processes use "Encoding" operations.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

- has an expiration date, which is sooner than the expiration date of the certificate of the issuer CA.

When signing a certificate, the issuer CA certifies the ownership of the keys (private and public) by the party whose name is in the certificate. Other parties can verify the certificate signature using the certificate of the issuer CA. So, if parties trust a CA, they can then rely upon the signatures generated using the certificates that the CA has issued.

The certificate of a CA may itself have been issued and signed by another CA, the later delegating to the first the right to issue certificates. The certification chain of a certificate shows the delegation sequence of CAs: it is the list of the certificates of all CAs' from the issuer CA until an unsigned or self-signed certificate, referred as the root certificate. The root certificates of the main public trusting organizations are available and already included in all popular web browsers.

A valid certificate is a non-expired certificate. An expired certificate shall not be used for authentication, encryption or for signing a document. However, it can still be used to decrypt old documents or verify their signatures, and thus to prove whatever may be necessary.

3.17.2 CERTIFICATES: FORMAT AND UNIQUE ID

All components (Endpoints, Gateways, and Nodes) shall use certificates to be authenticated by their communication peers (transport-layer security), to sign and to encrypt the messages (message-level security) when necessary.

The format of the certificates shall comply with the X.509 ITU-T standard, and the certificates' keys shall have a length of 2048 bits.

The exact concatenation of the standardized attributes "issuer" and "serial number" of a certificate forms a unique ID, referred as the "certificate ID".

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.17.3 USED CERTIFICATES AND ISSUERS (CAS)

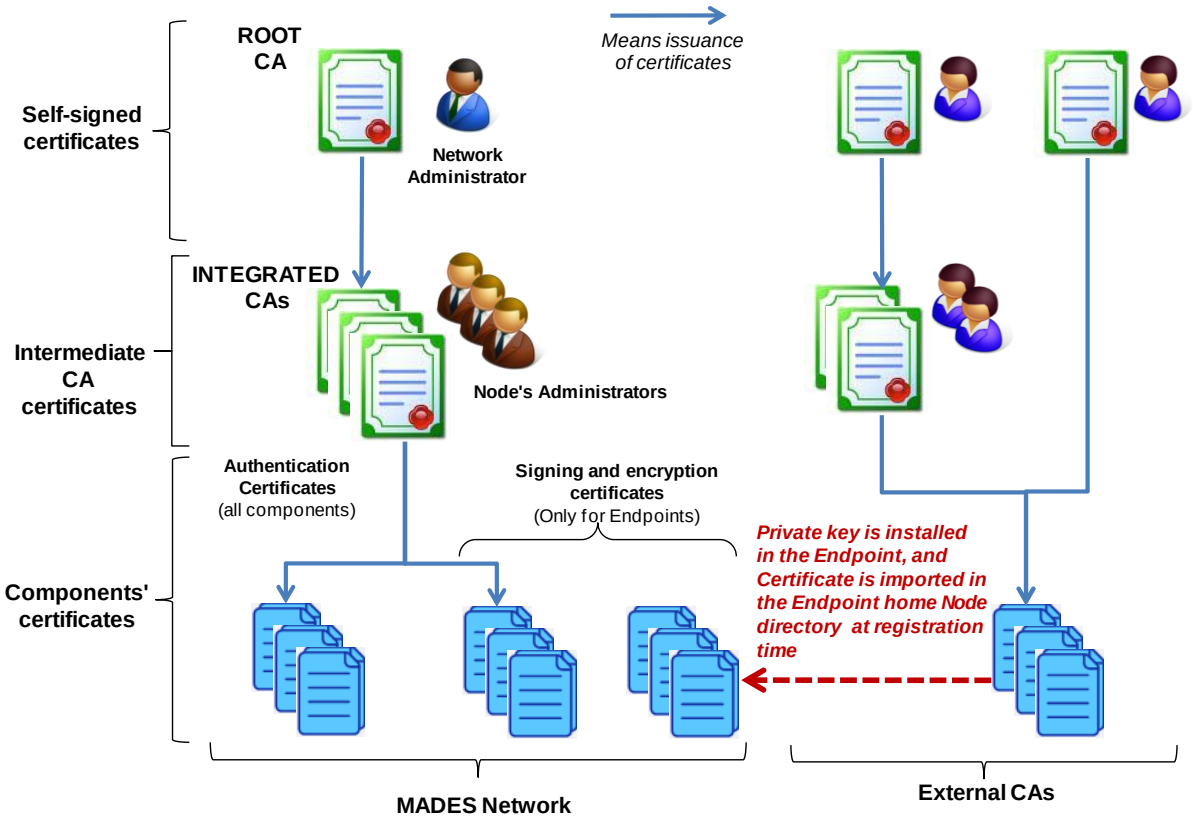


Figure 22: Certificates and Certificate Authorities (CAs) for a MADES network

Transport-layer security (Authorize data exchanges)
Each component (Endpoint, Gateway and Node) shall own an authentication certificate published in the home Node directory.

The authentication certificates are issued by the Network organization as follows:

- The organization owns a ROOT CA certificate.
- The organization delegates to each Node Administrator the right to issue the authentication certificates of the registered Endpoints and Gateways. Each Administrator owns an INTEGRATED CA certificate issued by the ROOT CA.

Each component shall store the authentication certificate and the corresponding private key. The authentication certificate is used whether the component acts as a client or a server.

Whatever the operation using the authentication certificate, it shall fail when the certificate has expired.

Message-level security (Protect message confidentiality and authenticate message issuer)
Each Endpoint shall own an encryption certificate published in the home Node directory for the others Endpoints to encrypt the business-messages they sent. The Endpoint uses the corresponding private key to decrypt the business-messages it receives.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Each Endpoint shall own a signing certificate published in the home Node directory for the others Endpoints to verify the signature of the messages they receive. The Endpoint uses the corresponding private key to sign the messages it sends.

The signing and encryption certificates can be issued by the home Node Administrator using the INTEGRATED CA, or by an EXTERNAL CA trusted by the network parties.

The signing certificate of an Endpoint can either be stored locally or inserted in a coding/decoding external device (e.g. smart cards).

The Endpoint shall never encrypt or sign a message using an expired certificate. The Endpoint shall not decrypt or verify a message signature using a certificate that was expired at the time the message was created (creation time is included in the message header).

Every Endpoint shall durably store the signing certificates of the other Endpoints in order to possess all necessary evidence.

3.17.4 DIRECTORY SERVICES

Content and updates

Each Node shall contain a directory where all the network components are described. Each entry for a component in the directory shall include:

- the component ID (non significant);
- the component display name (human readable);
- the component type (Endpoint, Gateway, Node);
- the technical contact information for operation or administration: name of the responsible person, e-mail and phones; the latter should be non personal (hotline, operation centre, functional/generic e-mail);
- the certificates owned by the component (one or several for each purpose including authentication, signing, and/or encryption – when applicable).

The Node administrator shall be able to update the directory entry of any of the registered components. This includes registering, updating and removing components, importing or renewing certificates for components. The description of the other Nodes and their registered components is imported using the Node synchronization — see § 3.16.1 and § 3.16.2.

Queries

The Gateways and Endpoints shall query their home Node directory to get information on a component and to retrieve certificates (e.g. to encrypt a message, to verify a message signature or to authenticate a component that signed a token or a component ID) — see § 5.3.4.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.17.5 CACHING DIRECTORY DATA

To reduce the request flow on the Node directory, the Endpoints and Gateways shall implement a caching mechanism for directory data.

A Node shall implement a TTL (Time-To-Live) mechanism, whose duration is configurable. It shall provide an expiration time for any dataset returned by a directory request: the time when the request is processed + the TTL value.

An Endpoint or a Gateway shall not use the data in cache when expired, and shall then request again for the data.

3.17.6 ENDPOINT REQUESTING DIRECTORY SERVICES THROUGH A GATEWAY

If the required directory data is not found or has expired in the Endpoint cache, the Endpoint shall request the data from the Gateway.

If the required directory data is not found or has expired in the Gateway cache, the Gateway shall request the data from the home Node directory.

This process shall be synchronous, the Endpoint waiting for the Gateway response. In case there is no connection established with the Node, the Gateway shall return a NO_CONNECTION error, so the Endpoint is not blocked, but has to retry later.

3.17.7 TRUSTING THE CERTIFICATES OF OTHERS COMPONENTS

Authentication

A component shall only communicate with a peer component if:

- the peer component belongs to the ROOT CA certification chain;
- the authentication certificate of the peer component is successfully verified.

During the TLS authentication phase, each peer shall convey to the other the following ordered certificate chain:

1. Its own authentication certificate.
2. The INTEGRATED CA certificate certified by the ROOT CA and which certifies the authentication certificate.

A component shall trust the ROOT CA and any authentication certificate provided by the home Node (e.g. used for token-authentication).

Signing and encryption

The Endpoint shall trust the signing and encryption certificates provided by the home Node.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.17.8 RENEWING THE EXPIRED CERTIFICATES

Renewing the authentication certificates

In case the authentication certificate of a component is renewed, the component will convey the certificate, possibly certified by a new INTEGRATED CA, to the peer component during the TLS authentication phase. When the INTEGRATED CA certificate is signed by the ROOT CA certificate, the communication is possible.

Additionally every component shall be configurable to communicate with components whose certificates may belong to two distinct certificate chains. So when the ROOT CA is renewed, components can communicate whether their certificate belongs to the old or to the new certification chain.

Renewing process (authentication, signing and encryption):

- The issuer CA shall renew the certificate enough time before the old one expires so that their validity periods overlap.
- The new certificate is published (imported) into the component home Node.
- The new certificate is then installed into the component (including the private key), before the old certificate expires.

To do so:

- A Node directory shall be able to store two certificates of any type for a component.
- Until the old encryption certificate expires, a Node shall not provide the new one when replying to a directory request, for it may not have been installed into the owner component — see rules for the *GetCertificate* service; § 5.3.4.2.
- An Endpoint shall be able to contain simultaneously two encryption certificates (private key).

Installing a new certificate into a component shall not last more than 5 minutes to ensure business continuity.

3.17.9 REVOKING A CERTIFICATE

A certificate shall only be revoked for security reasons when there is a reasonable doubt that it could be misused.

Revoking a certificate is a request to the certificate issuer. As a result, the issuer usually inserts the certificate serial number in a Certificate Revocation List (CRL), which can be publicly accessed. MADES does not implement such a CRL mechanism.

Within a MADES network, revoking a certificate is a request to the administrator of the Node where the component has registered. The Node administration tool shall provide the ability to revoke any certificate of a registered Endpoint or Gateway. The certificate shall then be tagged as revoked in the Node directory[12]. This tag is:

- propagated to others Nodes by the Node synchronization mechanism ;
- used to decide whether a requested certificate is delivered or not — see § 5.3.4.2.

[12] This shall increase the directory Dversion — see § 3.16.1.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

The Node administrator shall be able to revoke a certificate either issued by the INTEGRATED CA or by an EXTERNAL CA.

Such revocation of a certificate issued by an EXTERNAL CA has no link with the revocation process stated by the issuer in his certificate policy. The certificate owner shall also and always and independently ask for the certificate revocation by the certificate issuer. No MADES components ever access to any CRL of an external issuer.

The consequences of a certificate revocation, resulting from the message delivery mechanism described in the previous sections, are summarized in the next table.

Revoked certificate	Consequences
Endpoint Signing certificate	• From the revocation moment, all business-messages and all signed acknowledgements coming from the Endpoint and uploaded to the home Node will be rejected. • The business-messages and the signed acknowledgements coming from the Endpoint and uploaded to another Node will be rejected after the Node has synchronized with the Endpoint home Node. • The messages to and from the Endpoint pending in a Node (home or not) before the revocation tag is updated in the Node, will be delivered.
Endpoint Encryption certificate	• From the revocation moment, all business-messages for the Endpoint and uploaded to the home Node will be rejected. • The business-messages for the Endpoint pending in the home Node before the revocation moment, will be delivered to the Endpoint.
Endpoint Authentication certificate	• From the revocation moment, the home Node will reject downloading messages for the Endpoint. Those messages will be delivered (if not expired) after the Endpoint has renewed the certificate. • The Endpoint can still exchange with the Gateway until the revoked certificate cached data expires in the Gateway cache.
Gateway Authentication certificate	• The authentication-token to the home Node cannot be renewed → all requests are rejected by the Node. • The authentication-token to the others Nodes cannot be renewed after they synchronize with the Gateway home Node.

Note: The process to renew a revoked certificate is defined by the network governance.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

3.18 ABOUT GATEWAYS

3.18.1 ENDPOINT CHANGING OF GATEWAY

A Node shall not restrict the Gateways that an Endpoint can use. An Endpoint may change of Gateway in case of failure.

3.18.2 ENDPOINT WITHOUT A GATEWAY

The MADES specification has been designed so that a single component could merge the Endpoint and the Gateway functions and communicate with the BAs and the Nodes.

Such a component complies with the MADES specification and shall be registered as an Endpoint.

The component may not notify the events n°3 and n°5 during a business-message delivery (acceptance by Gateways — see Figure 15).

Other components shall behave correctly when such a component is included in the network. Following are examples of over-specified functions that shall be avoided:

- A Node checks that the connecting client is a Gateway.
- A component checks that event n°3 or event n°5 is missing during a message delivery.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

4 MANAGING THE VERSION OF THE MADES SPECIFICATION

4.1 ISSUES AND PRINCIPLES

When the MADES specification changes from version N-1 to version N, a MADES network should then upgrade to the new version.

A “big bang” rollout on all components would be both complex to coordinate and risky regarding business continuity, and thus inacceptable.

A smooth rollout means that an upgraded Endpoint can successfully exchange messages with a non-upgraded Endpoint (using version N-1), and that two upgraded Endpoints can successfully exchange messages using version N.

This section shows how such a rollout shall be done and the constraints that any version of the specification should satisfy.

4.1.1 ROLLING OUT A NEW VERSION (MVERSION AND N-COMPLIANCE)

The rollout of a new version shall start with Nodes. A Gateway shall only upgrade after the home Node did. An upgraded Endpoint shall only connect to an upgraded Gateway.

A Node upgraded to version N can successfully process the requests from the non upgraded Gateways if and only if it still exposes interfaces compliant with version N-1. So, as a general rule, a server upgraded to version N shall still expose the N-1 compliant interfaces. Also the upgraded Gateways and Nodes shall still request, being clients, the non-upgraded Nodes (e.g. for authentication, message transfer and synchronization).

A component is referred as N-compliant when it complies with version N of the MADES specification, and when it can successfully transfer messages with components which comply with version N-1.

From version 2, every component shall be N-compliant. This means that it complies with a version and can exchange using the previous version.

Every component shall access the installed version to which it complies, referred as the Mversion (MADES version).

Notations:

- A N-service or a N-interface is a service or an interface that complies with the version N of the MADES specification.
- A N-component is a N-compliant component (e.g. N-Node, N-Endpoint). Note that a N-component exposes (as server) or uses (as client) N-services and N-1-services.
- A N-message is a message composed according to the version N of the MADES specification.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

4.1.2 SERVICE COMPATIBILITY

A N-component server exposes both N-interface and N-1-interface. This does not mean that the N-interface is completely new (e.g. some services may not change).

It is up to the specification team to decide which and how the functions, the interfaces and the services evolve. The possible changes for a service are among the followings:

N°	Service changes
1	The service does not change.
2	The description of the service does not change but the way the elements are used in queries and responses does change. E.g.: • Some previously technically optional elements are now functionally required. • New values are now possible in the elements (e.g. new encryption algorithm using different metadata).
3	The description of the service changes in a compatible way. E.g.: • A new optional element is created. • A mandatory element becomes optional. • An unused optional element is removed.
4	The description of the service changes in a non compatible way → Actually, it's a new service with a new name.

In order to allow the specification team to use all these possibilities, most services include a "*serviceMversion*" element as part of the request. So the service behaviour can change without creating a new service, provided the client uses that element to tell the server which version of the specification it is working with. The server can then process the request and reply as expected.

4.1.3 MESSAGE COMPATIBILITY

The description of a message or the way a message is composed may evolve from version N-1 to version N. When this happens, a N-1-component will probably fail to process a N-message.

To ensure that a sender's Endpoint composes a message that a recipient's Endpoint can understand, the principles are the followings:

- The Node directory shall store (dynamically) the installed Mversion of the registered Endpoints, which is then transferred to other Nodes through the synchronization process.
- Each Endpoint shall notify its installed Mversion to the home Node when starting.
- The directory services shall provide the installed Mversion of an Endpoint.
- The description of a message contains a "*messageMversion"* element which tells the version of the specification to which the message complies.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

- The transfer (upload and download) N-services shall mix N-messages and N-1-messages, e.g. the collection of messages transferred in the *UploadMessage* request can contain both N-messages and N-1 messages.
- A sender's Endpoint shall compose a business-message that the recipient's Endpoint can understand —see detailed rules in § 4.2.4.
- A component shall compose an acknowledgment using the same Mversion as the original message.
- In case a component receives a message that it cannot process:
  - ✓ It shall reject the message, while confirming the transfer when a Node.
  - ✓ Otherwise it shall log the error and (if possible) it may store the message in the FAILED state, and shall issue a failure-event.

4.1.4 INTERFACE WITH BAS

The BAs are not concerned with the MADES specification version; so the used Mversion is not an element of the Endpoint interface.

A change in a service of the Endpoint interface should be backward compatible; otherwise the new specification should create a new service, and both (old and new) services should be described in the new specification. New BAs would then use the new service, and existing applications would migrate to the new service. Thus the migration timescale for the BAs can be kept independent of the network component upgrade.

An Endpoint Administrator shall be able to configure an association between a business-type and a minimum required Mversion. The default value is 1. It can be used when some new features available from this version are required for the business process (e.g. new encryption algorithm).

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

4.2 USING THE CORRECT VERSION FOR SERVICES AND MESSAGES

4.2.1 NODE SYNCHRONIZATION AND AUTHENTICATION

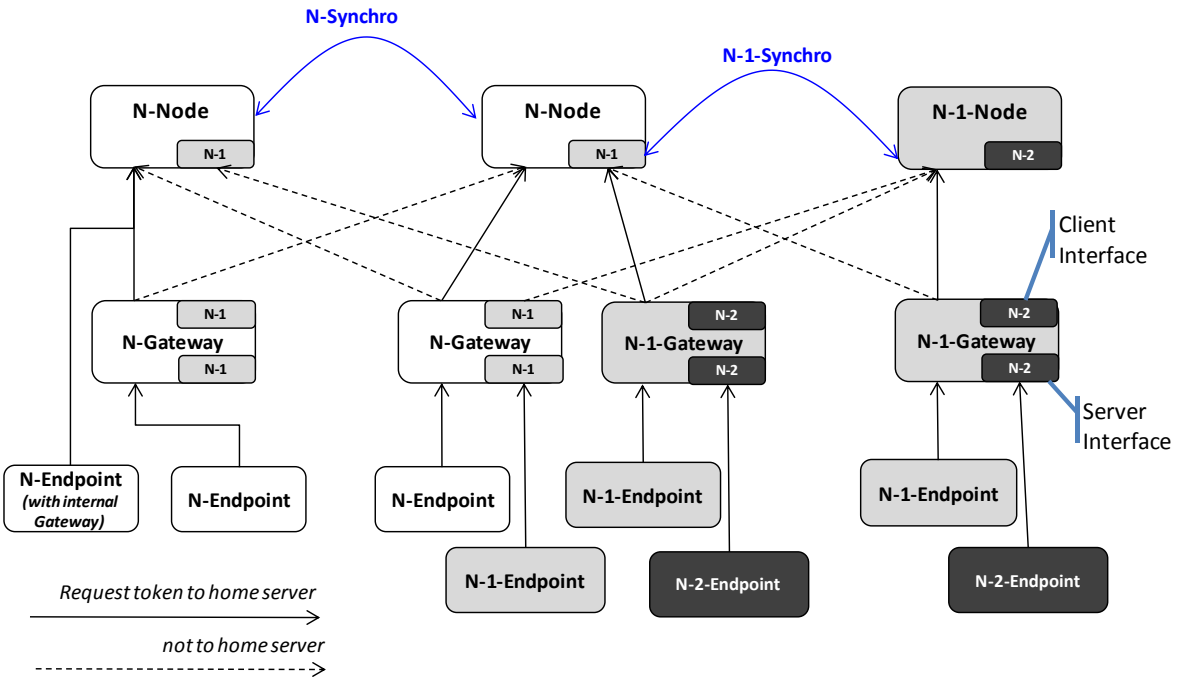


Figure 23: Managing the specification version – Node Synchronization and Authentication

The figure shows which version of the authentication and synchronisation service is used between components. A N-component server also exposes a N-1-interface and, acting as a client can request the N-1-interface of a N-1-component server.

Node synchronization:

- A Node shall request and store the Mversion of each Node of the Node-list file.
- The *GetNodeMversion* service (see § 5.3.5.1) shall be used by a Node to get the Mversion of a Node of the Node-list file, each time the Node (re)starts and each time the Node-list file is updated.
- A N_A-Node shall stop synchronizing with a N_B-Node when $|N_A-N_B|>1$.
- When requesting for a N_B-Node data using the *GetAllDirectoryData* service (see § 5.3.5.2), a N_A-Node shall use the N-service, where $N=Min(N_A,N_B)$.

Requesting for an authentication-token:

- A N-Endpoint shall request for a token to a Gateway using the N-service.
- A N-Gateway shall request for a token to the home Node using the N-service.
- A N_G-Gateway shall request for a token to another N_N-Node using the N-service where $N=Min(N_G,N_N)$[13]. The Mversion of the Node is available in the home directory with the Node routing information.

[13] In a network where the rollout rule is more binding (*A Gateway shall only upgrade after all Nodes did*), a N-Gateway can always and simply use the N-services with any Node (home or other).

4.2.2 DIRECTORY SERVICES AND NETWORK ACCEPTANCE

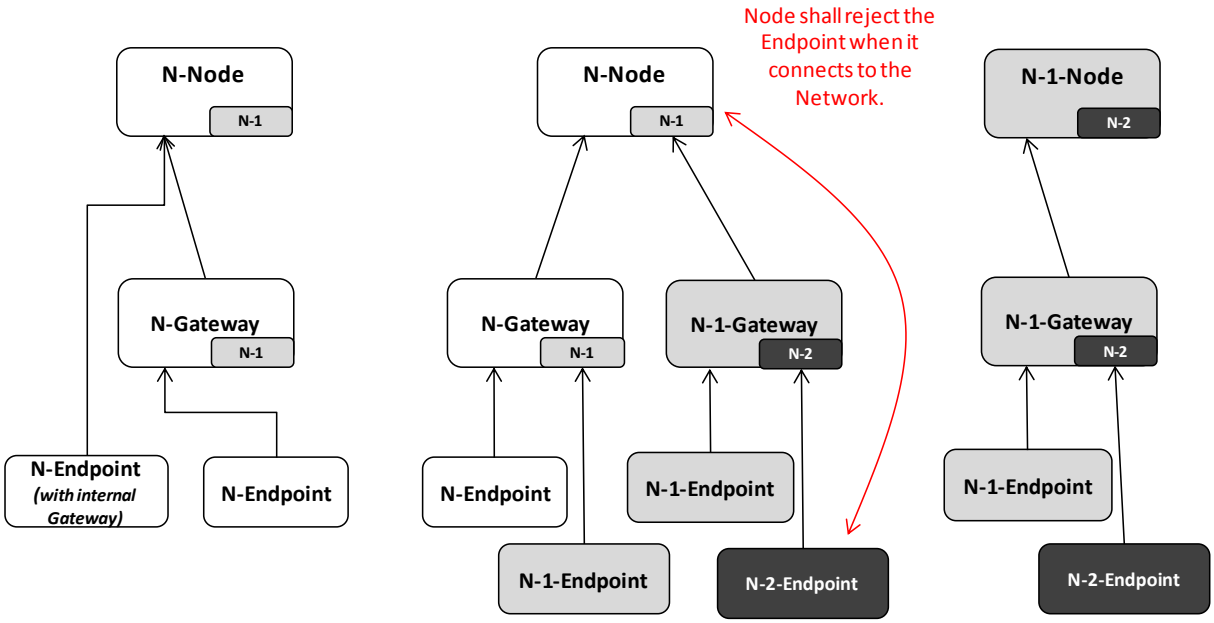


Figure 24: Managing the specification version – Directory services

A N-component shall always use the N-interface when requesting a directory service.

After a client component has obtained an authentication-token from its home server (Endpoint with Gateway, Gateway with home Node), it shall always request for acceptance in the network.

To do so, the component uses the *SetComponentMversion* service (see § 5.3.4.1) to tell the server about its installed Mversion. The reply informs the component whether it is accepted or rejected by the network.

A rejected component shall log the error and stop running.

Acceptance by the home Node

- A N_N-Node shall reject a N_C-component when either:
  - ✓ the component did not register with the Node;
  - ✓ the Node cannot authenticate the component (i.e. incorrect signed Endpoint ID);
  - ✓ (N_C>N_N) or (N_C<N_N-1).
- Otherwise the Node shall accept the component, store the component Mversion in the directory, increase the directory data version (Dversion) if N_C changes, and reply providing its own Mversion (N_N).
- A Node shall log that the Mversion of a component has changed; or that the component has been rejected.

Gateway relaying an Endpoint request:

- A Gateway shall durably store the last known Mversion of the home Node : N_{LN}
- A N_G-Gateway shall reject a N_E-Endpoint when either:
  - ✓ The Gateway cannot authenticate the Endpoint (i.e. incorrect signed ID);
  - ✓ (N_E>N_G) or (N_E<N_G-1) ;

- ✓ The home Node is not accessible and the Gateway did never connect to it (i.e. N_{LN} is unknown)
- ✓ The home Node is not accessible and ((N_E>N_{LN}) or (N_E<N_{LN}-1)).

- Otherwise the Gateway relays the request to the home Node and returns the Node reply to the Endpoint, or accepts the Endpoint providing N_{LN} as the Node version.

The issue is that the BA, the Endpoint, the Gateway and possibly other Nodes communicate during the Node downtime (whatever possible considering the directory data in cache).

Upgrading a component

- When a server component restarts, after the installed version has changed, no session information (e.g. token) from the previously connected clients shall remain. This ensures that all clients will newly request for network acceptance.
- In case a running Gateway detects that the home Node Mversion has changed, it shall force all currently connected Endpoints to newly request for network acceptance (e.g. by expiring the authentication-tokens).

Note: In case an N-1-Endpoint is stopped, other components will continue to send it N-1-messages. When it comes back to the network, being upgraded to version N, other components will still continue to send it N-1-messages until their directory cache (see § 3.17.5) is renewed. But the Endpoint will process correctly those N-1-messages. Only the pending N-2-messages will be rejected, but anyway the Endpoint cannot exchange anymore with those peers until they upgrade.

4.2.3 MESSAGING SERVICES

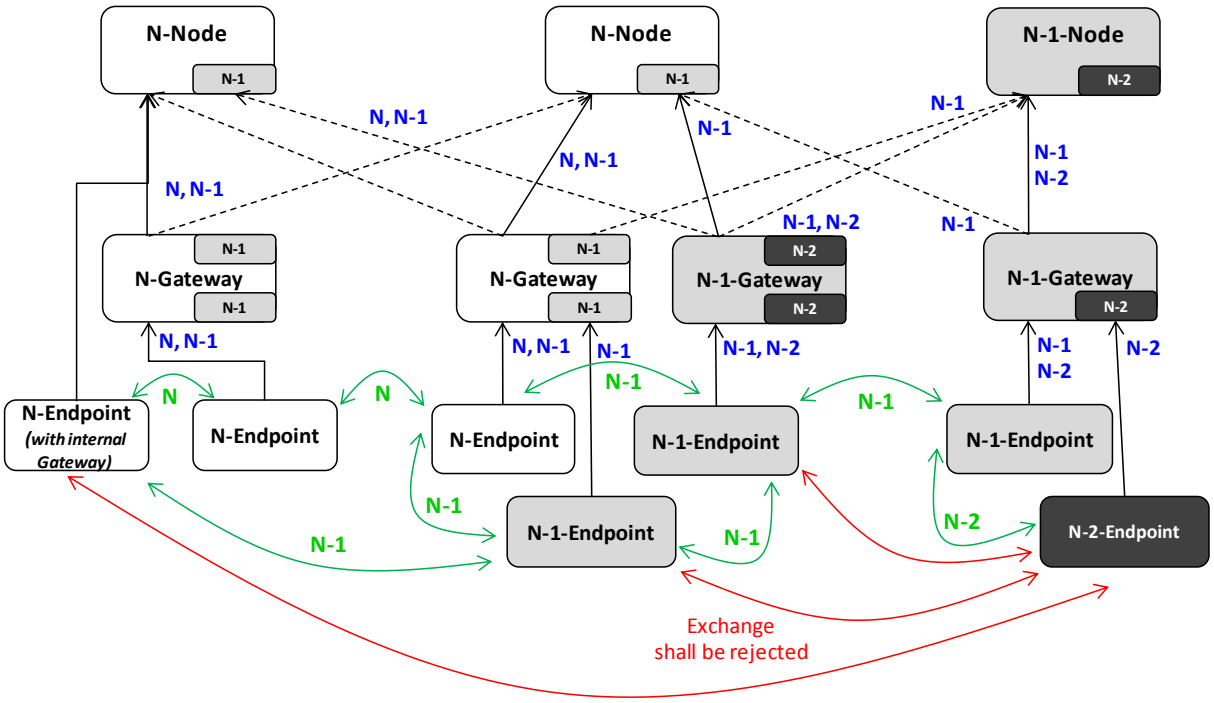


Figure 25: Managing the specification version – Messaging services

The figure shows:

- the messaging services that shall be used between components and the possible Mversion of the transferred messages (in blue);
- the Endpoints that can exchange messages and the required Mversion for the exchange (in green).

The figure presents a situation where two Endpoints cannot exchange although they only have 1 version difference (N-2, N-1). The reason is that the N-1-Endpoint has registered with a N-Node. And the N-Node will reject either the business-message or the acknowledgement.

4.2.4 WHICH VERSION TO USE TO SEND A MESSAGE?

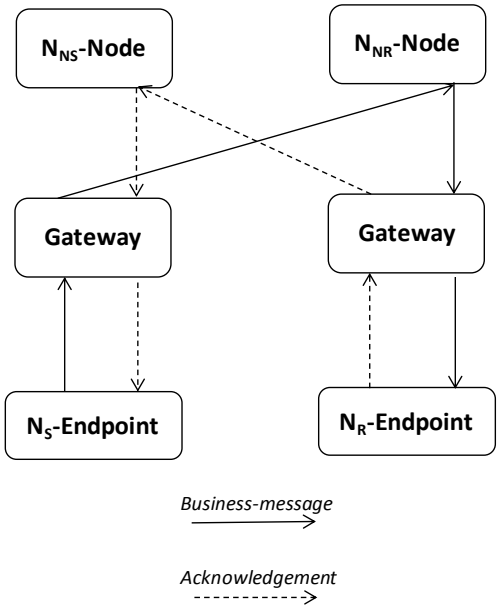


Figure 26: Managing the specification version – Which version to use to send a message?

Mversion	
N_S	The Mversion of the sender's Endpoint.
N_{NS}	The Mversion of the home Node of the sender's Endpoint.
N_R	The Mversion of the recipient's Endpoint.
N_{NR}	The Mversion of the home Node of the recipient's Endpoint.
N_B— see § 4.1.4	The minimum Mversion required for the business-type

1184 The used version should be $N = Min (N_S, N_R)$, however the message shall be rejected if one
1185 of the following conditions is verified:

Condition	Reason for rejection
N_R unknown	The MADES version of the recipient's Endpoint is unknown.
$\|N_R - N_S\| > 1$	The sender's Endpoint and the recipient's Endpoint are not MADES compatible.
$\|N_{NR} - N_S\| > 1$	The sender's Endpoint is not MADES compatible with the recipient's Node
$\|N_{NS} - N_R\| > 1$	The recipient's Endpoint is not MADES compatible with the sender's Node.
$N_B > N_S$	The sender's Endpoint is not MADES compatible with the minimal version required for the business-type
$N_B > N_R$	The recipient's Endpoint is not MADES compatible with the minimal version required for the business-type.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5 INTERFACES AND SERVICES

5.1 INTRODUCTION

This chapter all services for components to exchange each other or with the BAs. The description provides all elements in a request and the corresponding reply independently of an implementation language.

5.1.1 ERROR CODES

In case a service encounters an unrecoverable error, it returns information on the error. When not described the set of the returned elements is as follows:

Element name	Description	Element type
errorCode	A code representing the type of error.	string
errorID	Unique identification of the error.	string
errorMessage	An English readable text describing the error.	string
errorDetails	(optional) Additional English readable details about the error context.	string

String value for errorCode	Description
INVALID_PARAMETERS	The provided parameters (i.e. request elements) are incomplete, are not in the expected format or do not have the expected syntax.
AUTHENTICATION_ERROR	The peer component cannot be authenticated
VALIDATION_ERROR	The message is not valid (content size exceeded, unknown sender/recipient, signature is not valid etc.).
INTERNAL_ERROR	Internal application error. The error was not caused by the content of request but by the application itself (*Null Pointer Exception* in code, full database etc.)
NO_CONNECTION	Used in Endpoint to Gateway communication, when there is no connection to the Node.
CONCURRENT_ERROR	The server component is already processing a concurrent request from the same client.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.1.2 TYPES FOR TIME

All date and time shall be expressed in UTC (Coordinated Universal Time). The used time types are:

- "timestamp" technically means "xsd:long", and the value is the number of milliseconds since 'midnight 1.1.1970 UTC'.
- "dateTime" technically means "xsd:dateTime" and the value is according to the XSD specification (http://www.w3.org/TR/xmlschema-2/#dateTime).

5.2 ENDPOINT INTERFACE

5.2.1 OVERVIEW

The Endpoint interface provides the Business Applications (BAs) with the access to the MADES communication network.

MADES specifies this interface using Web services – The BA calls the web services exposed by the Endpoint.

There are 5 available services:

- *SendMessage* — used to upload a message into the Endpoint in order to send it to another Endpoint.
- *ReceiveMessage* — used to download a message from the Endpoint.
- *CheckMessageStatus* — used to check the current delivery status of a message.
- *ConnectivityTest* — used to check if another Endpoint can be reached.
- *ConfirmReceiveMessage* — used to notify the Endpoint that a received message has been technically accepted by a BA.

The BAs can access to the network using files. This interface is called FSSF (File System Shared Folders) and is described in § 5.2.3.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.2.2 SERVICES

5.2.2.1 SENDMESSAGE SERVICE

The *SendMessage* service is used by a BA to upload a message into the Endpoint in order to send it to another Endpoint.

Service request elements

Element name	Description	Element type	Required
message	Sending context, content and requested destination of the message.	SentMessage (see § 5.5.21)	True
conversationID	Unique identifier associated with the request.	string	False

About *conversationID*: There are situations where the sender's BA may not receive back or may fail to durably store the returned message ID, for example in case of failure of the Endpoint, of the network or of the BA itself. So the BA doesn't know if the message was or was not correctly transferred to the sender's Endpoint. There are two subsequent issues:

- If the message was actually correctly transferred and stored in the Endpoint, the BA doesn't know the message ID needed for further processing, such as checking the delivery status of the message.
- Considering that losing a message is a non acceptable risk, the BA will send the message again when the connection with the Endpoint is restored. The drawback is that the same message may then be sent twice with two different IDs.

So, resending the message using the same *conversationID* value solves both issues. Indeed, when an Endpoint is requested to send a message with a *conversationID* value that has already been used for an existing stored and sent message, it shall not send the message again but return the caller BA with the ID of the already existing message. The recommendation is that *conversationID := senderApplication + baMessageID.*

Service response elements

Element name	Description	Element type
messageID	The UUID (Universal Unique ID) of the message composed and stored by the Endpoint — see § 3.2.	string

Additional[14] error elements for the service

Element name	Description	Element type
receiverCode	The component ID of the requested recipient's Endpoint for the message.	string

[14] In addition to the elements described in § 5.1.1.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.2.2.2 RECEIVEMESSAGE SERVICE

The *ReceiveMessage* service is used by a BA to download a message from the Endpoint.

Service request elements

Element name	Description	Element type	Required
businessType	The business-type of the requested message — see § 3.3. Pattern: [A-Za-z0-9]+ [15]	string	True
downloadMessage	The service returns, if any, the first received and pending message having the requested business-type. “First” means according to the priority defined in § 3.13. The content (or document) of the message is or is not returned according to the value of the element: true := returned; false := not returned.	boolean	True

Service response elements

Element name	Description	Element type
receivedMessage	Sending context and possibly content of a message.	ReceivedMessage (see § 5.5.19)
remainingMessagesCount	The number of remaining messages received by the Endpoint, matching the requested business-type and waiting for delivery. In case the service returns the content of a message, the message is not included in the count of the remaining messages.	integer

Additional error elements for the service

Element name	Description	Element type
businessType	The business-type that was requested.	string

Note: until the recipient’s BA confirms to the recipient’s Endpoint that the message is correctly transferred using the *ConfirmReceiveMessage* service, the Endpoint shall consider that the message has not been transferred, but is still pending and shall be transferred again next time a BA requests for the business-type. This ensures that no message may be lost. As a consequence the BAs must be aware that, in some failure or recovery situations, they may possibly receive an already delivered message (i.e. having a known message ID).

[15] Pattern is the « regular expression » that the element value shall match.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.2.2.3 CHECKMESSAGESTATUS SERVICE

1242 The *CheckMessageStatus* service is used to check the current delivery status of a message.

1243 Service request elements

Element name	Description	Element type	Required
messageID	The UUID (Universal Unique ID) of the message whose status is requested — see § 3.2.	string	True

1244 Service response elements

Element name	Description	Element type
messageStatus	All Information about the message delivery.	MessageStatus (see § 5.5.13)

1245 Additional error elements for the service

Element name	Description	Element type
messageID	The requested message ID.	string

5.2.2.4 CONNECTIVITYTEST SERVICE

1246 The *ConnectivityTest* service can be used to check if another Endpoint can be reached. The
1247 service just sends a tracing message whose delivery-status can further be requested using
1248 the *CheckMessageStatus* service. The connectivity is successful, i.e. the tracing-message
1249 has reached the recipient's Endpoint, when the status is DELIVERED.

1250 Service request elements

Element name	Description	Element type	Required
receiverCode	The component ID of the recipient's Endpoint whose connectivity is checked. Pattern: [A-Za-z0-9-@]+	string	True

1251 Service response elements

Element name	Description	Element type
messageID	The message ID of the tracing-message.	string

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Additional error elements for the service

Element name	Description	Element type
receiverCode	The component ID of the recipient's Endpoint whose connectivity check was requested.	string

5.2.2.5 CONFIRMRECEIVEMESSAGE SERVICE

The *ConfirmReceiveMessage* service is used by a recipient's BA to confirm the download transfer of a message from the recipient's Endpoint.

A BA cannot reject a message; the business functional acceptance (i.e. compliance with business rules) is another issue. If a message is not confirmed back, for example in case of failure, the Endpoint will provide it again at the next *ReceiveMessage* call.

In case a BA encounters an unrecoverable error when processing a transferred message, and when the error comes from the message itself (e.g. inconsistent elements) and not from the application (e.g. file system full), the BA should confirm the transfer, log the error and possibly alert, otherwise the message will indefinitely be retransferred by the Endpoint until it is confirmed.

Service request elements

Element name	Description	Element type	Required
messageID	The UUID (Universal Unique ID) of the message whose transfer to the BA is being confirmed — see § 3.2.	string	True

Service response elements

Element name	Description	Element type
messageID	The UUID (Universal Unique ID) of the message whose transfer has be confirmed.	string

Additional error elements for the service

Element name	Description	Element type
messageID	The requested message ID.	string

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.2.3 FILE SYSTEM SHARED FOLDERS (FSSF)

5.2.3.1 INTRODUCTION

The FSSF interface is the way for a BA to exchange documents as files with the Endpoint. The file system where the files are written is accessed by the Endpoint as local file system. The principles are the followings:

- All the sender's BAs write in a common and unique OUT-folder the documents that the Endpoint shall send.
- The recipient's BAs read in an IN-folder the documents that the Endpoint has received.
- Additional information that is necessary for the message delivery is included in the filenames. Such information is the request/reply elements of the *SendMessage* and *ReceiveMessage* services.
- The organisation of the directories is local to each Endpoint and is configurable.

When implemented, a file interface with the Endpoint shall comply with the FSSF interface as described in the current section.

Note: There are differences between interfacing the Endpoint using FSSF and using the webservice interface.

- *CheckMessageStatus* and *ConnectivityTest* services are not supported.
- *ConfirmReceiveMessage* is implicit; i.e. a message is moved to the RECEIVED state in the recipient's Endpoint when the content has been successfully written into a file in the IN-folder.
- Actually FSSF, i.e. the processing of sending and receiving documents using files, may be considered —and also probably built— as a Business Application (BA) embedded with the Endpoint.

5.2.3.2 USED FILES AND FILE NAMING CONVENTION

There are 4 types of files used by the FSSF interface. Each file type is written into a separate folder.

The filenames are built from several parts joined by underscores ("_").

- Each part is limited to alphanumeric or hyphen characters. Accented characters, white spaces, and special characters shall not be used.
- Joining underscores must be present even when optional part is missing or empty.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Type	Folder / Writer	Description and Filename format
Files to be sent	OUT / BAs	• The folder contains the files written by BAs to be sent by the Endpoint. • The sender's Endpoint removes from the folder the files that it has processed correctly (i.e. accepted files are deleted) or not (i.e. rejected files are moved to the OUT_ERROR folder). • Filenames: <SenderBA>_<Recipient>_<BusType>_<BAmessageID>.<Ext>
Failed files	OUT_ERROR / Sender's Endpoint	• The folder contains the files that the sender's Endpoint did not process correctly. They have been moved from the OUT-folder to the OUT-ERROR-folder without changing their names. • It's up to the Endpoint Administrator to analyse and clean up the folder. • The filenames can match or not the "files to send" filename format. Note that not matching the filename format is a reason for the file not to be processed correctly.
Received files	IN / Recipient's Endpoint	• Each file in the folder contains the content of a message that has been received by the recipient's Endpoint. The filename is built from the header information of the received message. • The files should be removed from the folder when processed, correctly or not, by the recipient's BAs. • Filenames: <SenderBA>_<Sender>_<BusType>_<BAmessageID>_<MessageID>.<Ext>
Log files	OUT_LOG / Sender's Endpoint	• The folder contains one log file for each message accepted by the Endpoint. • The file contains English readable text. Each line reports an event about the message delivery, and is the concatenation of the *MessageTraceItem* structure, as provided in the *CheckMessageStatus* service response. • The file is appended with a new line each time a new event for the message is notified to the sender's Endpoint. • It's up to the Endpoint Administrator to clean up the OUT_LOG folder. • The filename is the exact name of the sent file with an added ".log" extension: <SenderBA>_<Recipient>_<BusType>_<BAmessageID>.<Ext>.log

Filename parts	Type	Description
<BAmessageID>	Optional	An identifier of the document provided by the sender's BA. Information is transported "as is" to the recipient's BA. — Pattern: [A-Za-z0-9-]*
<BusType>	Mandatory	The business type for the message (see § 3.3). — Pattern: [A-Za-z0-9-]*
<Ext>	Optional	The file extension — Pattern: [A-Za-z0-9-]*
<MessageID>	Mandatory	The UUID (Universal Unique ID) of the message composed by the sender's Endpoint (see § 3.2) — Pattern: [A-Za-z0-9-]+
<Sender>	Mandatory	The component code of the sender's Endpoint. — Pattern: [A-Za-z0-9-@]+
<SenderBA>	Optional	The identifier of the sender's BA. — Pattern: [A-Za-z0-9-]*
<Receiver>	Mandatory	The component code of the recipient's Endpoint. — Pattern: [A-Za-z0-9-@]+

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Additional rules:

- The to-be-sent filenames without extension shall not end with the dot character (“.”).
- The sender’s Endpoint shall ignore files in the OUT-folder with extension “TMP” (or “tmp”).
- The sender’s Endpoint shall fail to send the files that matches one of the following conditions:
  1. Filename does not match the “files to-be-sent” Filename format.
  2. Filename is longer than 200 characters.
  3. File is empty.

5.2.3.3 CONCURRENT ACCESS TO FILES

As the BAs and the Endpoint concurrently access to the files, special attention is required to avoid access conflicts and data losses.

Access conflicts

To avoid access conflicts between the writer and a reader of the file:

- The file-reader shall ignore files whose extension is “TMP” (or “tmp”).
- The file-writer shall write the data first in a temporary file whose extension is “TMP” (or “tmp”), and then rename it changing the extension (note: “rename” is an atomic operation on every file system).

Data losses

Data may be lost if a file is overridden by another file having the same filename. To avoid this, each file should have a unique filename:

- OUT – OUT_ERROR – OUT_LOG: It is highly recommended that the BAs uses <SenderBA> and <BAmessageID> to ensure they cannot use the same filename.
- IN: the use of <MessageID> in the filename ensures that the content of two different messages will always be written in 2 different files.

5.2.3.4 CONFIGURING FSSF

The Administrator shall be able to configure the Endpoint with following information:

- OUT folder name;
- OUT_ERROR folder name;
- OUT_LOG folder name;
- A list of business-types, and for each one an associated folder name and a default extension.

The recipient’s Endpoint shall write in files the content of the business-messages whose business-type is in the configured list:

- The file shall be written in the IN folder which is associated with the message’s business-type.
- The file shall have the extension provided in the “extension” attribute of the message header — see § 5.5.10. When none, the extension shall be the default extension for the message’s business-type.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.3 GATEWAY & NODE INTERFACES

5.3.1 OVERVIEW

The Gateway interface provides the Endpoint with the access to the network. MADES specifies the interface exposed by the Gateway using Web services — over SOAP/HTTPS protocol.

The Node interface provides the Gateway access to the Node. MADES specifies the interface exposed by the Node using Web services – over SOAP/HTTPS protocol.

Both Node and Gateway Interfaces expose the same services grouped as follows:

- Authentication service — see § 5.3.2.
- Messaging services — see § 5.3.3.
- Directory services — see § 5.3.4.

The Node synchronization interface is used by the Nodes to synchronize their directory data each other. MADES specifies the interface using Web services – over SOAP/HTTPS protocol — see § 5.3.5.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

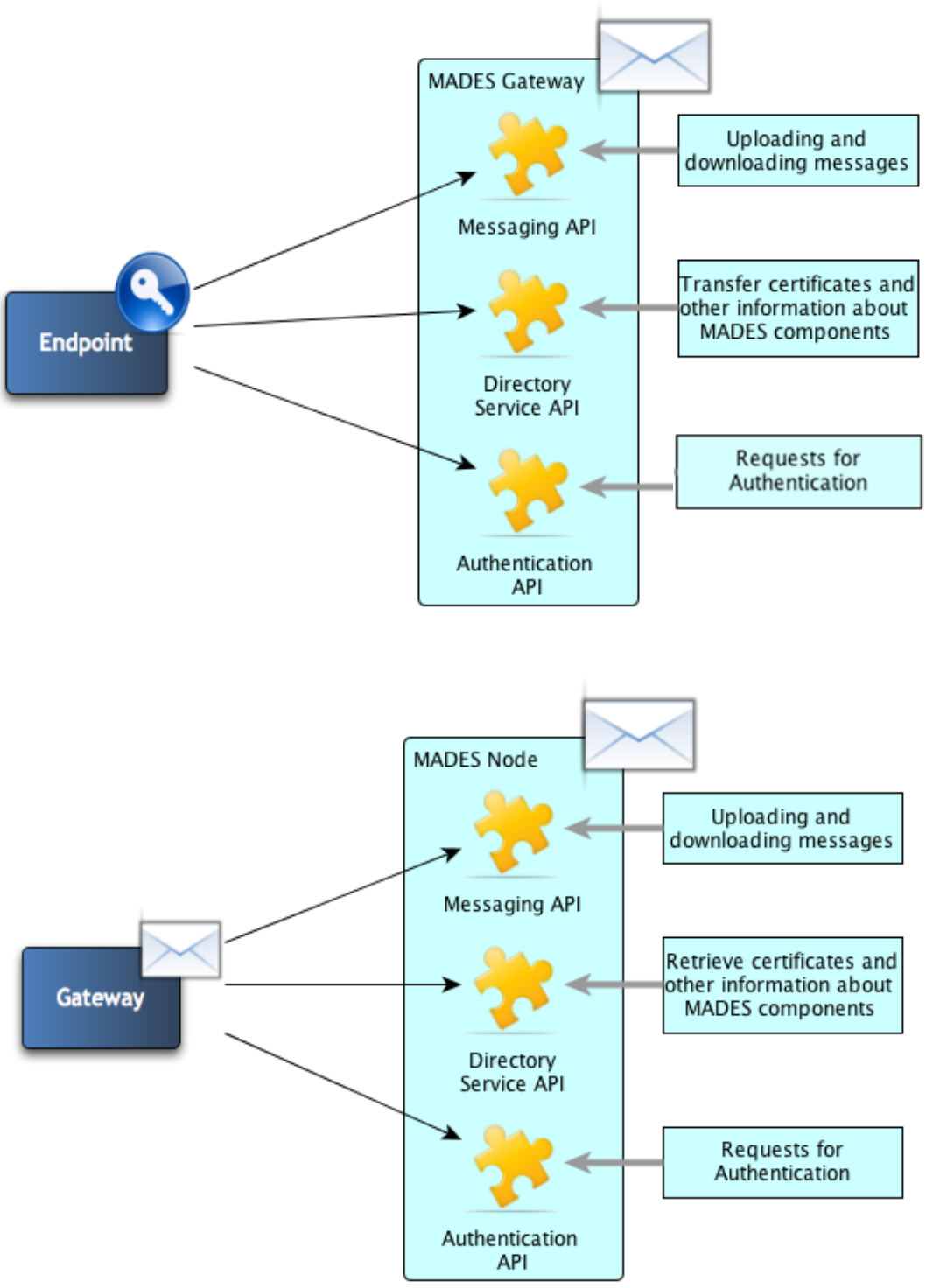


Figure 27: Gateway & Node Interfaces – Overview

5.3.2 AUTHENTICATION SERVICE

There is one authentication service, named *GetAuthenticationToken*, used by a client component to retrieve a token supplied by a server component, which is referred as the "authentication-token". The token has an expiration time (i.e. date and time). Such a token can be generated as a UUID.

The client shall then return the authentication-token signed with the authentication certificate for every following request — see § 3.5.2. The client has to renew the authentication-token using the same service when expired.

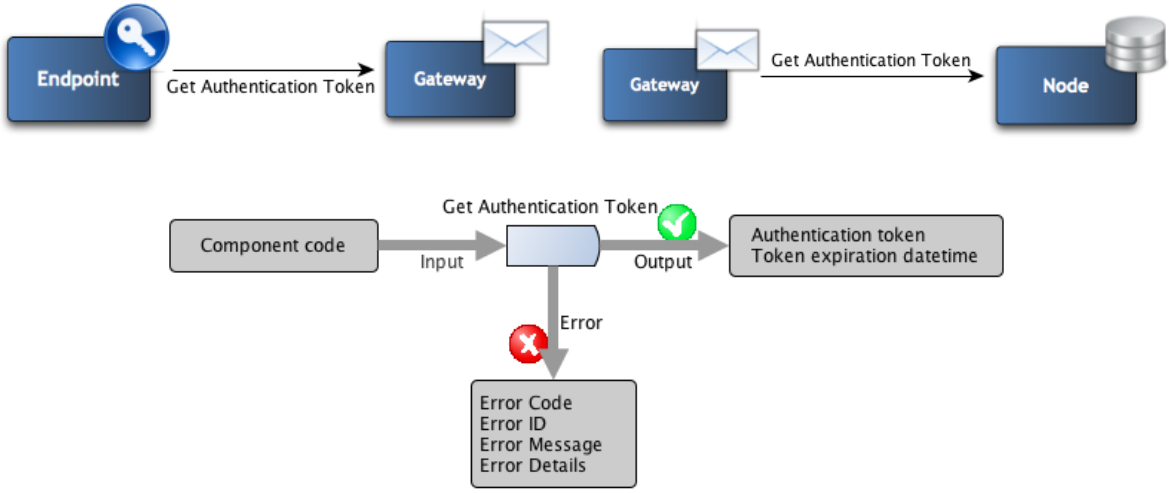


Figure 28: Gateway & Node Interfaces – Authentication Service

Service request elements

Element name	Type	Description	Required
componentCode	string	The component ID of the connecting client requesting for an authentication-token.	True
serviceMversion	integer	The MADES version of the current service that is requested by the client – see § 4.1.2.	True

Service response elements

Element name	Type	Description	Required
authToken	string	The requested authentication token.	True
expiration	timestamp	The expiration date and time of the provided authentication-token.	True

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.3.3 MESSAGING SERVICES

Messaging services are operations for bulk upload and download of messages.

The download process is a two-phase operation: first the client downloads messages from server; then it confirms that the download was successful — see § 3.8.

Figure 29: Gateway & Node Interfaces – Messaging Services

Two limits shall be configurable within each source component regarding the bulk transfer mechanism (defined by the network governance):

- the maximum number of messages in one transfer.
- the maximum allowed size for the request (upload) or the reply (download), which contains the messages.

5.3.3.1 "TRANSFER CONFIRMATION" VERSUS "ACCEPTANCE"

The transfer confirmation is a technical mechanism to notify a source component that the target component has taken responsibility for the message. If the source component does not receive the confirmation, it remains responsible for the message delivery and shall then transfer it again.

The acceptance of a message by a component generally means more, i.e. that the message has passed additional validation checks. Moreover, acceptance always leads to an event notification (whether delivery or failure).

Upload:
When a Node accepts an uploaded message, it delegates the event notification of the event n°4 to the sender's Gateway and uses the upload response to do so. Other components shall not use the upload response to reject a business-message. The interpretation of the possible responses by the sender's Gateway is:

1. No confirmation is received; the message shall be transferred again.
2. The message is accepted; a delivery acknowledgement shall be issued if the message is a business-message.
3. The message is rejected (it shall only on a business-message); a failure acknowledgement shall be issued and the message shall not be transferred again.

Download:

The target component of a download transfer (recipient's Gateway or recipient' Endpoint) always issues the acknowledgement. So there is no need to accept or reject a message when confirming the transfer (see *ConfirmDownload* § 5.3.3.4).

5.3.3.2 UPLOADMESSAGES SERVICE

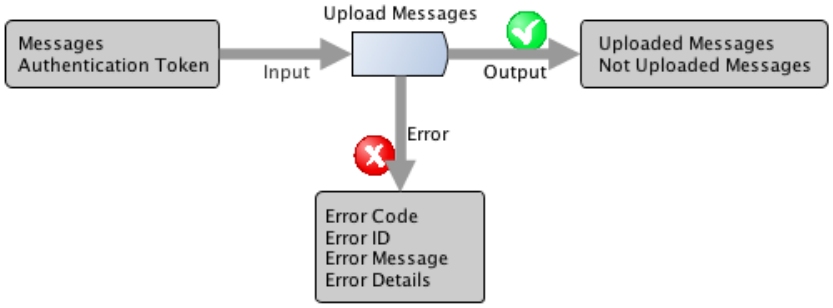


Figure 30: Gateway & Node Interfaces – Messaging Services – UploadMessages Service

Service request elements

Element name	Element type	Description	Required
messages	InternalMessage[] (see § 5.5.10)	The collection of the messages to be uploaded ordered according to priority rule in the client.	True
authToken	AuthenticationToken (see § 5.5.2)	The authentication token provided by the server which is signed back by the client using the authentication certificate.	True
serviceMversion	integer	The MADES version of the current service that is requested by the client – see § 4.1.2.	True

Service response elements

Element name	Element type	Description	Required
uploadedMessages	string[]	The collection of the IDs of the messages which are confirmed as transferred, or accepted.	False
notUploadedMessages	notUploadedMessageResponse[] (see § 5.5.17)	The collection of {ID, error details} for each non-accepted (i.e. rejected) message.	False

Each ID of the uploaded messages provided in the response shall belong to one and only one of these two collections.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.3.3.3 DOWNLOADMESSAGES SERVICE

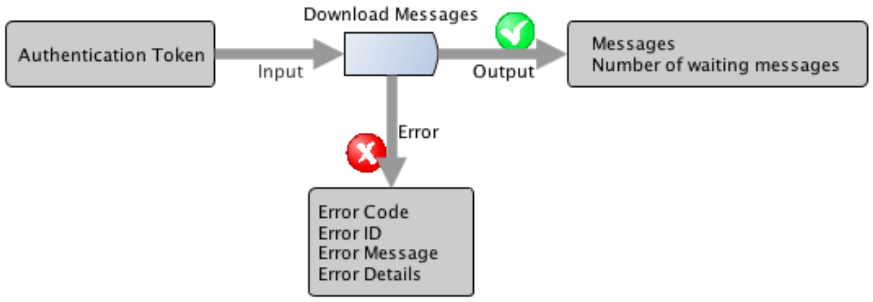


Figure 31: Gateway & Node Interfaces – Messaging Services – DownloadMessages Service

A client component shall present the signed component IDs of the Endpoints for which it requests messages. The certificate used for signing an Endpoint ID shall be the authentication certificate of the Endpoint.

A Gateway can verify the signed ID from an Endpoint, and shall use it when requesting for downloading messages from the Node.

The Gateway should simultaneously download messages all the connected Endpoints, provided it presents the collection of theirs signed IDs. To do so the Gateway has to cache the signed IDs received from the Endpoints. However a Gateway shall only download messages for an Endpoint if the authentication token with the Endpoint is valid (not expired).

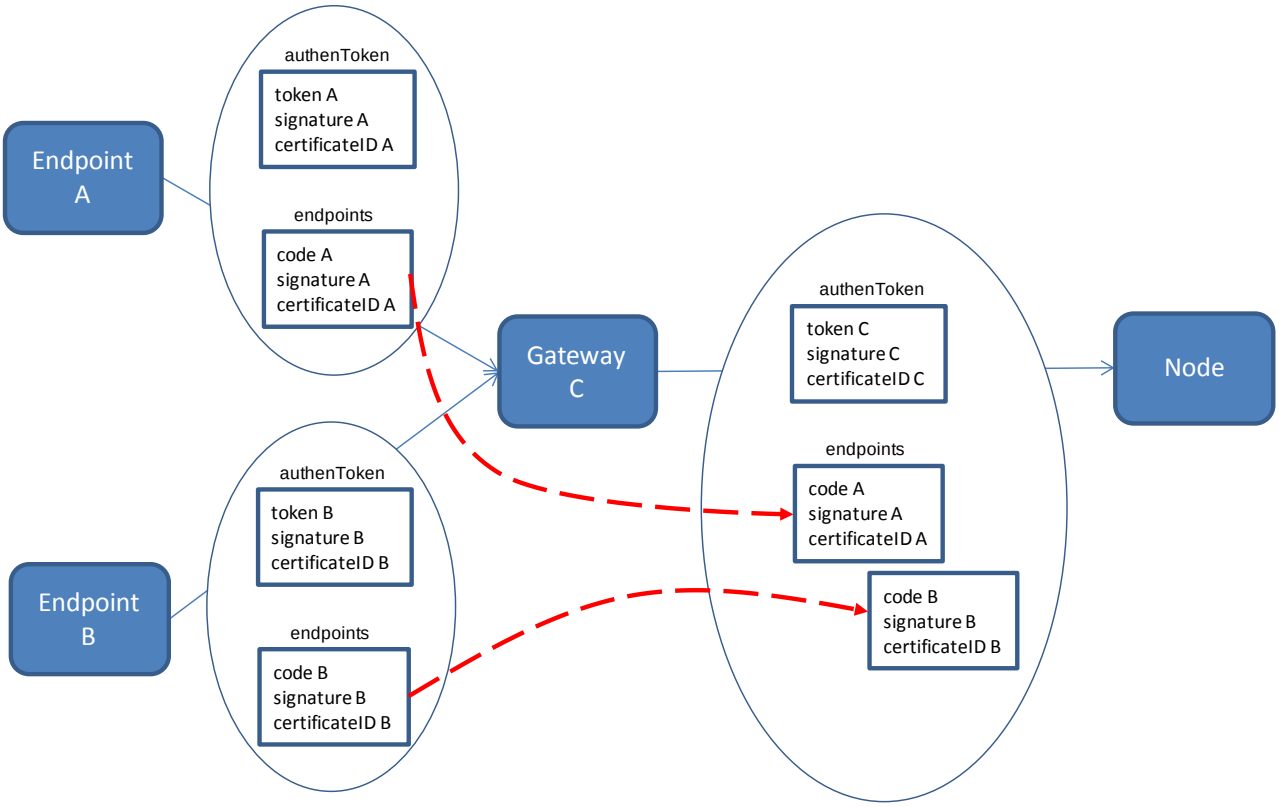


Figure 32: Gateway & Node Interfaces – Messaging Services – Download authorization

The Node shall verify that every Endpoint ID is successfully signed with a non-revoked authentication certificate of the Endpoint, and log an error message when the verification fails.

The Node shall process the request while omitting the failed IDs, and reject it when none succeeded. Thus a failing Endpoint won't block other Endpoints connected to the Gateway.

Service request elements

Element name	Element type	Description	Required
endpoints	Endpoint[] (see § 5.5.9)	A collection of signed component IDs of some recipient's Endpoints whose messages are requested for download.	True
authToken	AuthenticationToken (see § 5.5.2)	The authentication token provided by the server which is signed back by the client using the authentication certificate.	True
serviceMversion	integer	The MADES version of the current service that is requested by the client – see § 4.1.2.	True

Service response elements

Element name	Element type	Description	Required
messages	InternalMessage[] (see § 5.5.10)	The collection of the IDs of the messages which are confirmed transferred — see § 3.8	False
waitingMessages	integer	The number of messages, matching the request, not included in the current response and still waiting to be downloaded by the client.	True

5.3.3.4 CONFIRMDOWNLOAD SERVICE

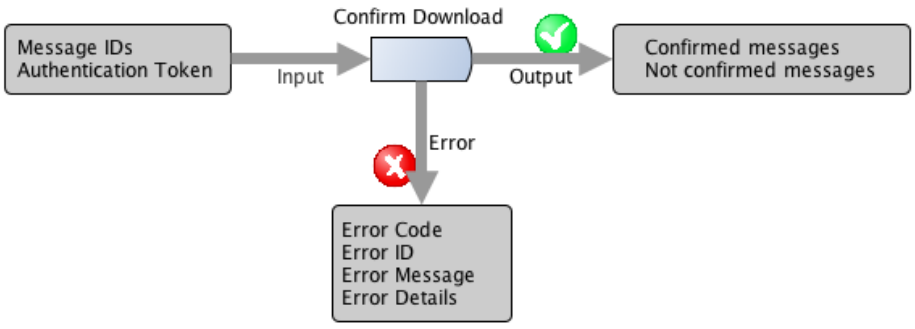


Figure 33: Gateway & Node Interfaces – Messaging Services – ConfirmDownload Service

The client (source) component shall confirm all or none of the messages that were previously transferred during the previous download request.

Service request elements

Element name	Element type	Description	Required
messageIDs	string[]	The collection of the IDs of the messages whose download transfer is confirmed.	False

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Element name	Element type	Description	Required
authToken	AuthenticationToken (see § 5.5.2)	The authentication token provided by the server which is signed back by the client using the authentication certificate.	True
serviceMversion	integer	The MADES version of the current service that is requested by the client – see § 4.1.2.	True

Service response elements

Element name	Element type	Description	Required
confirmedMessages	string[]	*(Unused)*	False
notConfirmedMessages	NotConfirmedMessageResponse [] (see § 5.5.17)	*(Unused)*	False

5.3.4 DIRECTORY SERVICES

Figure 34: Gateway & Node Interfaces – Directory Services

5.3.4.1 SETCOMPONENTMVERSION SERVICE

SetComponentMversion is used by a component to be accepted in the network — see § 4.2.2.

To prevent that a component sends wrong data which could disrupt the network behaviour, the component ID shall be signed.

Service request elements

Element name	Element type	Description	Required
componentCode	string	The ID of the component requesting for network acceptance.	True
signature	string	The RSA encoding of the SHA-1 hash of the component ID (componentCode) of the component requesting for network acceptance. The certificate used for encoding shall be the authentication certificate of the component.	True
certificateID	string	The ID of the certificate used to encode “signature”.	True
componentMVersion	integer	The installed MADES version of the component requesting for network acceptance.	True

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Element name	Element type	Description	Required
authToken	AuthenticationToken (see § 5.5.2)	The authentication token provided by the server which is signed back by the client using the authentication certificate.	True
serviceMversion	integer	The MADES version of the current service that is requested by the client – see § 4.1.2.	True

Service response elements

Element name	Element type	Description	Required
nodeMversion	integer	The installed MADES version of the home Node.	True
acceptance	boolean	True if the component is accepted in the network.	True

5.3.4.2 GETCERTIFICATE SERVICE

GetCertificate is used to retrieve a certificate of a given type (signing, encryption, or authentication), owned by the given Endpoint and possibly having the requested ID.

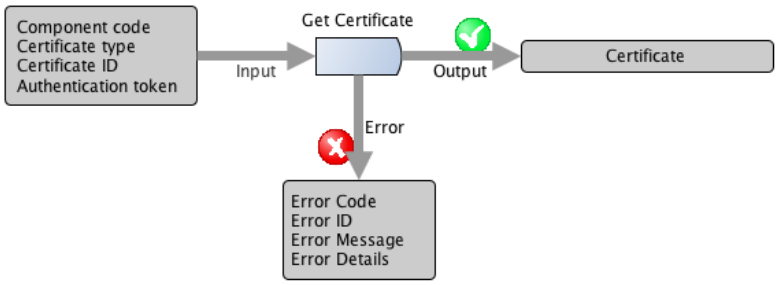


Figure 35: Gateway & Node Interfaces – Directory Services – GetCertificate Service

Service request elements

Element name	Element type	Description	Required
componentCode	string	The ID of the component that owns the requested certificate.	True
type	CertificateType (see § 5.5.4)	The type of the requested certificate,	True
certificateID	string	The ID of the requested certificate.	False
authToken	AuthenticationToken (see § 5.5.2)	The authentication token provided by the server which is signed back by the client using the authentication certificate.	True
serviceMversion	integer	The MADES version of the current service that is requested by the client – see § 4.1.2.	True

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

1415 Service response elements

Element name	Element type	Description	Required
certificate	Certificate (See § 5.5.3)	The returned certificate shall match the requested "*type*" and shall be owned by *componentCode*. If *certificateID* is also requested, the returned certificate shall also match the ID. Additional conditions about the validity and the revocation of the certificate are provided further. If no certificate matches, the response is empty.	False

1416 Additional conditions:

Certificate type	*certificateID* is requested	*certificateID* is not requested
Authentication	The returned certificate shall be valid (not expired) and not revoked.	Request Error: this situation should never occur. A component shall only request for an authentication certificate to check a signed token or a signed component ID, and thus knowing the certificate ID.
Encryption	The returned certificate can be expired or revoked, for it may be requested to decrypt a message that was composed before the expiration time or the revocation time.	The returned certificate shall be valid and not revoked. When several certificates are possible, the service shall return the certificate that expires first.
Signing	The returned certificate can be expired or revoked, for it may be requested to check the signature of a message that was composed before the expiration time or the revocation time.	Request Error: this situation should never occur. A component shall only request for a signing certificate to check a signature and thus knowing the certificate ID.

5.3.4.3 GETCOMPONENT SERVICE

1417 *GetComponent* is used for retrieving descriptive and routing information on a component.

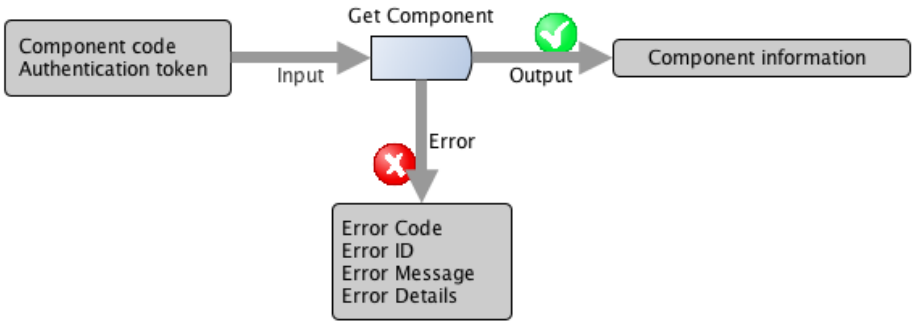


Figure 36: Gateway & Node Interfaces – Directory Services – GetComponent Service

1418 Service request elements

Element name	Element type	Description	Required
componentCode	string	The ID (or code) of the requested component.	True

Element name	Element type	Description	Required
authToken	AuthenticationToken (see § 5.5.2)	The authentication token provided by the server which is signed back by the client using the authentication certificate.	True
serviceMversion	integer	The MADES version of the current service that is requested by the client – see § 4.1.2.	True

1419 Service response elements

Element name	Element type	Description	Required
component	ComponentInformation (see § 5.5.7)	The directory Information about the component. If the requested component does not exist, the response is empty.	False

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.3.5 NODE SYNCHRONIZATION INTERFACE

5.3.5.1 GETNODEMVERSION SERVICE

1420 The *GetNodeMversion* service is used by a Node to get the Mversion of another Node — see
1421 § 4.2.1.

1422 Service request elements

Element name	Element type	Description	Required
mversion	integer	The installed MADES version of the requesting client Node.	True

1423 Service response elements

Element name	Element type	Description	Required
mversion	integer	The installed MADES version of the replying server Node.	True
nodeCode	string	The component ID of the replying server Node.	True

5.3.5.2 GETALLDIRECTORYDATA SERVICE

1424 The *GetAllDirectoryData* service is used by the Nodes to synchronize each other.

1425 Service request elements

Element name	Element type	Description	Required
dversion	integer	The version of the directory data of the server Node that the client Node already owns. Note: No version shall be provided if the client synchronizes for the first time with the server.	False
serviceMversion	integer	The MADES version of the current service that is requested by the client – see § 4.1.2.	True

1426 Service response elements

Element name	Element type	Description	Required
dversion	integer	The current version of the directory reference data of the replying server Node.	True
nodeCode	string	The component ID of the replying server Node.	True
components	ComponentDescription[] (see § 5.5.6)	The collection of the descriptions of all components registered to the replying server Node, plus the description of the Node itself. The collection shall only be provided when the current directory version of the server Node is strictly higher than the version already owned by the client Node.	False

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.4 FORMAT OF THE NODE-LIST FILE

1427 The Node-list file shall be in UTF-8 encoding.

1428 The records in the file shall be delimited by the new line character (LF U+000A).

1429 Each record provides a list of attributes for one Node. Attributes are delimited by the empty
1430 space character (SPACE U+0020) and shall appear in the following order:

Attribute	Description	Required
Node component ID	The component ID of the Node	True
Primary Node URL	The primary URL to access the Node, formatted according to RFC 1738	True
Secondary Node URL	The secondary URL to access the Node, formatted according to RFC 1738	False

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.5 TYPED ELEMENTS USED BY THE INTERFACES

5.5.1 USING MTOM

MTOM (*Message Transmission Optimization Mechanism*) is a W3C recommendation for handling binary data in SOAP messages — http://www.w3.org/TR/soap12-mtom/

MTOM shall be used to optimise the size of the messages sent. Binary data in the SOAP message have to be encoded as text because SOAP uses XML. The base64 text encoding increases the size of the data by about 33%. MTOM provides a way to send the binary data in the original binary form. MTOM optimizes the element content that is in the canonical lexical representation of the *xsd:base64Binary* type.

5.5.2 AUTHENTICATIONTOKEN

Element name	Element type	Description	Required
token	string	The token received by the client (or requesting) component when it authenticated against the server (see the *GetAuthenticationToken* service).	True
signature	string	The signed token, i.e. the RSA encoding of the SHA-1 hash of the token. The encoding certificate shall be the authentication certificate of the client.	True
certificateID	string	The ID of the certificate used for signing the token.	True

5.5.3 CERTIFICATE

Element name	Element type	Description	Required
certificateID	string	The ID of the certificate.	True
certificate	base64Binary	The binary data of the certificate in DER (Distinguished Encoding Rules) format.	True
expiration	timestamp	The expiration date-time of the provided certificate in the caller cache — see § 3.17.5. *Do not confuse with the expiration date of the certificate as defined by the certificate issuer and included in the certificate itself.* *The element is required for directory service, but not for directory synchronization*	

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.5.4 CERTIFICATETYPE (STRING ENUMERATION)

String value	Description
AUTHENTICATION	A certificate used for TLS and token authentication
ENCRYPTION	A certificate used for encryption
SIGNING	A certificate used for signing

5.5.5 COMPONENTCERTIFICATE

Element name	Element type	Description	Required
type	CertificateType (see § 5.5.4)	The type of the certificate (e.g. encryption, signing, authentication),	True
revoked	boolean	'true' if the certificate has been revoked.	False
certificate	Certificate (see § 5.5.3)	The certificate.	True

5.5.6 COMPONENTDESCRIPTION

Element name	Element type	Description	Required
information	ComponentInformation (see § 5.5.7)	All about the component: ID, type, contact information, routing information.	True
certificates	ComponentCertificate[] (see § 5.5.5)	The collection of the certificates owned by the component, whatever type (signing, encryption and authentication), and possibly more than one for a type.	True

5.5.7 COMPONENTINFORMATION

Element name	Element type	Description	Required
code	string	The component ID to which is associated all information of this current data structure.	True
type	ComponentType (see § 5.5.8)	The type of component.	True
organization	string	The organization responsible for the component.	True
person	string	The technical contact person.	True
email	string	The email of the technical contact person.	True
phone	string	The phone number of the technical contact person.	True
routing	RoutingInformation (see § 5.5.20)	The routing information to access to the component (ex: URLs)	True

Element name	Element type	Description	Required
expiration	timestamp	The expiration date-time of the provided information in the caller cache — see § 3.17.5. *The element is required for directory service, but not for directory synchronization*	
codeMversion	integer	The MADES version to which the component complies – see § 4.1.1. Note: Information should be initialized at registration time. Otherwise it remains unknown until the component first connects to the network, and before that time, no message can be sent to the component.	False

5.5.8 ComponentType (string enumeration)

String value	Description
NODE	The component is a Node.
GATEWAY	The component is a Gateway.
ENDPOINT	The component is an Endpoint.

5.5.9 Endpoint

Element name	Element type	Description	Required
code	string	The component ID of an Endpoint.	True
signature	string	The RSA encoding of the SHA-1 hash of the component ID of the Endpoint. The certificate used for encoding is the authentication certificate of the Endpoint.	True
certificateID	string	The ID of the certificate used to encode the signature.	True

5.5.10 InternalMessage

Element name	Element type	Description	Required
messageID	string	The ID of the message.	True
receiverCode	string	*Business-message, Tracing-message* → The component ID of the recipient's Endpoint. *Acknowledgement* → The *senderCode* of the original message.	True
businessType	string	*Business-message* → The business-type as provided by the sender's BA. *Tracing-message* → Irrelevant, but at least one character needed. *Acknowledgement* → The business-type of the original message.	True

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Element name	Element type	Description	Required
content	base64Binary	*Business-message* → The content of the message which is possibly compressed and then encrypted. *Tracing-message* → A non empty (at least one character) irrelevant and not compressed but encrypted content. *Acknowledgement* → see § 3.10.4.	True
extension	string	*Business-message* → The file extension for the document — only used if the content was transferred to the sender's Endpoint through a file and by the FSSF interface (see § 5.2.3). *Acknowledgement, Tracing-message* → Not used.	False
generated	dateTime	*Business-message, Tracing-message* → The date and time when the message was created by the sender's Endpoint. *Acknowledgement* → The date and time of the notified event, i.e. when the acknowledgement was created in the sending component.	True
expirationTime	timestamp	*Business-message, Tracing-message* → The expiration date and time of the message — set by the sender's Endpoint when the message was accepted (see § 3.9). *Acknowledgement* → The *expirationTime* of the original message.	True
senderCode	string	*Business-message, Tracing-message* → The component ID of the sender's Endpoint. *Acknowledgement* → The ID of the component sending the acknowledgement.	True
senderDescription	string	The display name of the *senderCode* component.	True
internalType	InternalMessageType (see § 5.5.11)	The technical type of the message.	True
relatedMessageID	string	*Business-message, Tracing-message* → Not used. *Acknowledgement* → The message ID of the original message..	False
SenderApplication	string	*Business-message, Tracing-message* → The identifier of the sender's BA as provided when sending the document. *Acknowledgement* → Not used.	False
baMessageID	string	*Business-message, Tracing-message* → An identifier of the document as provided by the sender's BA. *Acknowledgement* → Not used.	False
metadata	MessageMetadata (see § 5.5.12)	The metadata added to the message by compression, signature or encryption – see § 3.14.	False
messageMversion	integer	The MADES version to which the message complies – see § 4.1.3	True

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Note:

- The underlined attributes are those included in the manifest used to generate the message signature – see § 3.14.2.
- The “message header” refers to the set of all elements except “content”.

5.5.11 INTERNALMESSAGETYPE (STRING ENUMERATION)

String Value	Description
STANDARD_MESSAGE	A business-message but not a tracing-message.
DELIVERY_ACKNOWLEDGEMENT	An acknowledgement notifying that the original STANDARD_MESSAGE has been accepted by a component (Gateway, Node, or Endpoint).
RECEIVE_ACKNOWLEDGEMENT	An acknowledgement notifying that the original STANDARD_MESSAGE has been transferred to a recipient’s BA.
FAILURE_ACKNOWLEDGEMENT	A failure-acknowledgement.
TRACING_MESSAGE	A tracing-message – see § 3.12.
TRACING_ACKNOWLEDGEMENT	An acknowledgement notifying that the original TRACING_MESSAGE has been accepted by a component (Gateway, Node, or Endpoint)..

5.5.12 MESSAGEMETADATA

Element name	Element type	Description	Required
messageProcessors	MessageProcessor[]	A collection of metadata, each from a used message processor (collection count may range from 1 to 3).	False

MessageProcessor

Element name	Element type	Description	Required
processorID	string	The unique ID of the message processor. There are 3 processors whose IDs are: • “signature” • “encryption” • “compressor”	True
processorData	Map	A collection of named values.	True

Map

Element name	Element type	Description	Required
entries	MapEntry[]	A collection of data, each provided with a name and a value, i.e. a set composed of a key (name), a type (format) and a value (according to the type).	False

1444 MapEntry

Element name	Element type	Description	Required
key	string	The name of the metadata	True
type	ValueType	The type/format of the metadata.	True
value	string	The value of the metadata.	True

1445 ValueType (enumeration)

String Value	Description
STRING	String
LONG	A 64bit number expressed as string. Example number 42 is represented as string "42" (without quotes)
BYTE_ARRAY	⇔ base64Binary type
BOOLEAN	A string equal to “true” or “false”.

5.5.13 MESSAGESTATE (STRING ENUMERATION)

1446 The possible values of the string are the ones of *MessageTraceState* — see § 5.5.16.

5.5.14 MESSAGESTATUS

Element name	Element type	Description
messageID	string	The UUID (Universal Unique ID) of the message whose status is reported in this data structure — see § 3.2.
state	MessageState (see § 5.5.13)	The delivery-status of the requested message. (see values in § 3.4 - uppercase)
receiverCode	string	The component ID of the recipient’s Endpoint of the message.
senderCode	string	The component ID of the sender’s Endpoint of the message.
businessType	string	The business-type of the message.
senderApplication	string	The identifier of sender’s BA, if any and as provided by the sender’s BA in the *SendMessage* service.
baMessageID	string	The identifier of the message assigned by the sending BA, if any and as provided by the sender’s BA in the *SendMessage* service.
sendTimestamp	dateTime	The time when the message was created by the sender’s Endpoint (The *generated* element of the *InternalMessage* type – see § 5.5.10).

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Element name	Element type	Description
receiveTimestamp	dateTime	The “reception time” of the message in the sender’s Endpoint. It is the time when the message state was set to DELIVERED in the sender’s Endpoint, which is also the time when the acknowledgement with the DELIVERED status (event n°6) was sent.
trace	MessageTraceItem [] (see § 5.5.15)	The collection of the traces reporting the events about the message delivery.

5.5.15 MESSAGETRACEITEM

Element name	Element type	Description	Required
timestamp	dateTime	The date and time of the reported event.	True
state	MessageTraceState (see § 5.5.16)	The reported event	True
component	string	The ID of the component (see § 3.2) where the event happened.	True
Component description	string	The display name of the component where the event happened.	True
Details	string	The English readable details about the event.	False

5.5.16 MESSAGETRACESTATE (STRING ENUMERATION)

String value	Description
VERIFYING	The acceptance of the message by the sender’s Endpoint is pending due to connectivity problem between the sender’s Endpoint and the directory services. (*internal event reported by the sender’s Endpoint*).
ACCEPTED	The message has been accepted by the sender’s Endpoint. (*internal event reported by the sender’s Endpoint*).
TRANSPORTED	The message has been accepted by an intermediate component (except the recipient’s Endpoint or a recipient’s BA). *(event reported using a DELIVERY_ACKNOWLEDGEMENT or a TRACING_ACKNOWLEDGEMENT – see § 5.5.11)*
DELIVERED	The message has been accepted by the recipient’s Endpoint. *(event reported using a DELIVERY_ACKNOWLEDGEMENT or a TRACING_ACKNOWLEDGEMENT – see § 5.5.11)*
RECEIVED	The message has been accepted by a recipient’s BA. *(event reported using an RECEIVE_ACKNOWLEDGEMENT – see § 5.5.11)*
FAILED	The processing of the message has failed and the delivery is stopped. *(internal event reported by the sender’s Endpoint or event reported using a FAILURE_ACKNOWLEDGEMENT – see § 5.5.11)*

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.5.17 NOTCONFIRMEDMESSAGERESPONSE

Element name	Element type	Description	Required
messageID	string	The ID of the message whose upload failed.	True
errorCode	string	A code representing the type of the error (e.g. validation error, unexpected error)	True
errorID	string	Unique identification of the error and for the component implementation. The error ID shall be always written in the logs.	True
errorMessage	string	An English readable text describing the error.	True
errorDetails	string	(optional) Additional English readable details about the error context.	False

5.5.18 NOTUPLOADEDMESSAGERESPONSE

Element name	Element type	Description	Required
messageID	string	The ID of the message whose upload failed.	True
fatal	boolean	Set to true if the error is not recoverable. The message shall then be set in the failed state by the client source component, which shall never try to upload it again.	True
businessErrorMessage	string	An English readable description of the error.	False
errorCode	string	A code representing the type of the error (e.g. validation error, unexpected error)	True
errorID	string	Unique identification of the error and for the component implementation. The error ID shall be always written in the logs.	True
errorMessage	string	An English readable text describing the error.	True
errorDetails	string	(optional) Additional English readable details about the error context.	False

5.5.19 RECEIVEDMESSAGE

Element name	Element type	Description
messageID	string	The UUID (Universal Unique ID) of the message — see § 3.2.
receiverCode	string	The component ID of the recipient's Endpoint of the message — see § 3.2.
senderCode	string	The component ID of the sender's Endpoint of the message — see § 3.2.

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

Element name	Element type	Description
businessType	string	The business-type of the message currently transferred to the BA.
content	base64Binary	The content of the message as provided by the sender's BA in the *SendMessage* service.
senderApplication	string	The identifier of sender's BA, if any and as provided by the sender's BA in the *SendMessage* service.
baMessageID	string	The identifier of the message assigned by the sending BA, if any and as provided by the sender's BA in the *SendMessage* service.

5.5.20 ROUTINGINFORMATION

Element name	Element type	Description	Required
node	string	The component ID of the component's home Node.	True
primaryURL	string	The primary URL of the Node according to RFC 1738	True
secondaryURL	string	The secondary URL of the Node according to RFC 1738	False
nodeMversion	integer	The installed MADES version of the component's home Node – see § 4.1.1	True

5.5.21 SENTMESSAGE

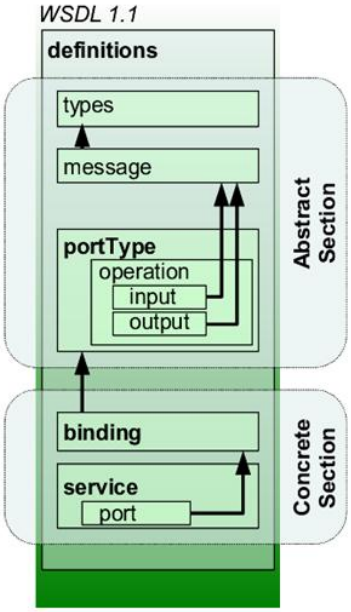
Element name	Element type	Description	Required
receiverCode	string	The component ID of the recipient's Endpoint (see § 3.2) — Pattern: [A-Za-z0-9-@]+	True
businessType	string	The business-type of the message (see § 3.3) — Pattern: [A-Za-z0-9]+	True
baMessageID	string	An identifier of the document provided by the sender's BA. Information is transported "as is" to the recipient's BA — Pattern: [A-Za-z0-9]*	False
senderApplication	string	The identifier of the sender's BA. Information is transported "as is" to the recipient's BA — Pattern: [A-Za-z0-9]*	False
content	base64Binary	The content of the message, i.e. the business document. Note: There is no constraint about the structure of the document which is processed as a stream of bytes. E.g. it can be a human-readable XML document, multiples files compressed in ZIP format.	True

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.6 DESCRIPTION OF THE SERVICES USING WSDL

The services are described using the Web Services Description Language (WSDL) 1.1[16]
See http://www.w3.org/TR/wsdl

The SOAP 1.1 and SOAP 1.2 bindings allow using the interfaces via SOAP 1.1 and SOAP 1.2 protocols.



[16] Figure from http://en.wikipedia.org/wiki/Web_Services_Description_Language

5.6.1 ENDPOINT INTERFACE

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ECPEndpoint" targetNamespace="http://mades.entsoe.eu/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ecp="http://mades.entsoe.eu/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">

 <wsdl:types>
  <xsd:schema targetNamespace="http://mades.entsoe.eu/">

   <xsd:element name="SendMessageRequest">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="message" type="ecp:SentMessage"/>
      <xsd:element minOccurs="0" name="conversationID" nillable="true" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="SendMessageResponse">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="messageID" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="SendMessageError">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="errorCode" type="xsd:string"/>
      <xsd:element name="errorID" type="xsd:string"/>
      <xsd:element name="errorMessage" type="xsd:string"/>
      <xsd:element minOccurs="0" name="receiverCode" nillable="true" type="xsd:string"/>
      <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="ReceiveMessageRequest">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="businessType" type="xsd:string"/>
      <xsd:element name="downloadMessage" type="xsd:boolean"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="ReceiveMessageResponse">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element minOccurs="0" name="receivedMessage" nillable="true" type="ecp:ReceivedMessage"/>
      <xsd:element name="remainingMessagesCount" type="xsd:long"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="ReceiveMessageError">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="errorCode" type="xsd:string"/>
      <xsd:element name="errorID" type="xsd:string"/>
      <xsd:element name="errorMessage" type="xsd:string"/>
```

```
1513            <xsd:element minOccurs="0" name="businessType" nillable="true" type="xsd:string"/>
1514            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
1515          </xsd:sequence>
1516        </xsd:complexType>
1517      </xsd:element>
1518
1519      <xsd:element name="ConfirmReceiveMessageRequest">
1520        <xsd:complexType>
1521          <xsd:sequence>
1522            <xsd:element name="messageID" type="xsd:string"/>
1523          </xsd:sequence>
1524        </xsd:complexType>
1525      </xsd:element>
1526
1527      <xsd:element name="ConfirmReceiveMessageResponse">
1528        <xsd:complexType>
1529          <xsd:sequence>
1530            <xsd:element name="messageID" type="xsd:string"/>
1531          </xsd:sequence>
1532        </xsd:complexType>
1533      </xsd:element>
1534
1535      <xsd:element name="ConfirmReceiveMessageError">
1536        <xsd:complexType>
1537          <xsd:sequence>
1538            <xsd:element name="errorCode" type="xsd:string"/>
1539            <xsd:element name="errorID" type="xsd:string"/>
1540            <xsd:element name="errorMessage" type="xsd:string"/>
1541            <xsd:element minOccurs="0" name="messageID" nillable="true" type="xsd:string"/>
1542            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
1543          </xsd:sequence>
1544        </xsd:complexType>
1545      </xsd:element>
1546
1547      <xsd:complexType name="SentMessage">
1548        <xsd:sequence>
1549          <xsd:element name="receiverCode" type="xsd:string"/>
1550          <xsd:element name="businessType" type="xsd:string"/>
1551          <xsd:element name="content" type="xsd:base64Binary"/>
1552          <xsd:element minOccurs="0" name="senderApplication" nillable="true" type="xsd:string"/>
1553          <xsd:element minOccurs="0" name="baMessageID" nillable="true" type="xsd:string"/>
1554        </xsd:sequence>
1555      </xsd:complexType>
1556
1557      <xsd:complexType name="ReceivedMessage">
1558        <xsd:sequence>
1559          <xsd:element name="messageID" type="xsd:string"/>
1560          <xsd:element name="receiverCode" type="xsd:string"/>
1561          <xsd:element name="senderCode" type="xsd:string"/>
1562          <xsd:element name="businessType" type="xsd:string"/>
1563          <xsd:element name="content" type="xsd:base64Binary"/>
1564          <xsd:element minOccurs="0" name="senderApplication" nillable="true" type="xsd:string"/>
1565          <xsd:element minOccurs="0" name="baMessageID" nillable="true" type="xsd:string"/>
1566        </xsd:sequence>
1567      </xsd:complexType>
1568
1569      <xsd:complexType name="MessageStatus">
1570        <xsd:sequence>
1571          <xsd:element name="messageID" type="xsd:string"/>
1572          <xsd:element name="state" type="ecp:MessageState"/>
1573          <xsd:element name="receiverCode" type="xsd:string"/>
1574          <xsd:element name="senderCode" type="xsd:string"/>
1575          <xsd:element name="businessType" type="xsd:string"/>
1576          <xsd:element minOccurs="0" name="senderApplication" nillable="true" type="xsd:string"/>
1577          <xsd:element minOccurs="0" name="baMessageID" nillable="true" type="xsd:string"/>
1578          <xsd:element name="sendTimestamp" type="xsd:dateTime"/>
```


ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

```
1579          <xsd:element minOccurs="0" name="receiveTimestamp" nillable="true" type="xsd:dateTime"/>
1580          <xsd:element name="trace" nillable="true" type="ecp:MessageTrace"/>
1581        </xsd:sequence>
1582      </xsd:complexType>
1583
1584      <xsd:complexType name="MessageTrace">
1585        <xsd:sequence>
1586          <xsd:element maxOccurs="unbounded" name="trace" type="ecp:MessageTraceItem"/>
1587        </xsd:sequence>
1588      </xsd:complexType>
1589
1590      <xsd:complexType name="MessageTraceItem">
1591        <xsd:sequence>
1592          <xsd:element name="timestamp" type="xsd:dateTime"/>
1593          <xsd:element name="state" type="ecp:MessageTraceState"/>
1594          <xsd:element name="component" type="xsd:string"/>
1595          <xsd:element name="componentDescription" type="xsd:string"/>
1596          <xsd:element name="details" nillable="true" type="xsd:string"/>
1597        </xsd:sequence>
1598      </xsd:complexType>
1599
1600      <xsd:element name="ConnectivityTestRequest">
1601        <xsd:complexType>
1602          <xsd:sequence>
1603            <xsd:element name="receiverCode" type="xsd:string"/>
1604          </xsd:sequence>
1605        </xsd:complexType>
1606      </xsd:element>
1607
1608      <xsd:element name="ConnectivityTestResponse">
1609        <xsd:complexType>
1610          <xsd:sequence>
1611            <xsd:element name="messageID" type="xsd:string"/>
1612          </xsd:sequence>
1613        </xsd:complexType>
1614      </xsd:element>
1615
1616      <xsd:element name="ConnectivityTestError">
1617        <xsd:complexType>
1618          <xsd:sequence>
1619            <xsd:element name="errorCode" type="xsd:string"/>
1620            <xsd:element name="errorID" type="xsd:string"/>
1621            <xsd:element name="errorMessage" type="xsd:string"/>
1622            <xsd:element minOccurs="0" name="receiverCode" nillable="true" type="xsd:string"/>
1623            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
1624          </xsd:sequence>
1625        </xsd:complexType>
1626      </xsd:element>
1627
1628      <xsd:element name="CheckMessageStatusRequest">
1629        <xsd:complexType>
1630          <xsd:sequence>
1631            <xsd:element name="messageID" type="xsd:string"/>
1632          </xsd:sequence>
1633        </xsd:complexType>
1634      </xsd:element>
1635
1636      <xsd:element name="CheckMessageStatusResponse">
1637        <xsd:complexType>
1638          <xsd:sequence>
1639            <xsd:element name="messageStatus" type="ecp:MessageStatus"/>
1640          </xsd:sequence>
1641        </xsd:complexType>
1642      </xsd:element>
1643
1644      <xsd:element name="CheckMessageStatusError">
```

```
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element name="errorCode" type="xsd:string"/>
        <xsd:element name="errorID" type="xsd:string"/>
        <xsd:element name="errorMessage" type="xsd:string"/>
        <xsd:element minOccurs="0" name="messageID" nillable="true" type="xsd:string"/>
        <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>

     <xsd:simpleType name="MessageState">
      <xsd:restriction base="xsd:string">
       <xsd:enumeration value="VERIFYING"/>
       <xsd:enumeration value="ACCEPTED"/>
       <xsd:enumeration value="DELIVERING"/>
       <xsd:enumeration value="DELIVERED"/>
       <xsd:enumeration value="RECEIVED"/>
       <xsd:enumeration value="FAILED"/>
      </xsd:restriction>
     </xsd:simpleType>

     <xsd:simpleType name="MessageTraceState">
      <xsd:restriction base="xsd:string">
       <xsd:enumeration value="VERIFYING"/>
       <xsd:enumeration value="ACCEPTED"/>
       <xsd:enumeration value="TRANSPORTED"/>
       <xsd:enumeration value="DELIVERED"/>
       <xsd:enumeration value="RECEIVED"/>
       <xsd:enumeration value="FAILED"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="SendMessageRequest">
   <wsdl:part name="parameters" element="ecp:SendMessageRequest"/>
  </wsdl:message>

  <wsdl:message name="SendMessageResponse">
   <wsdl:part name="parameters" element="ecp:SendMessageResponse"/>
  </wsdl:message>

  <wsdl:message name="ConnectivityTestFault">
   <wsdl:part name="fault" element="ecp:ConnectivityTestError"/>
  </wsdl:message>

  <wsdl:message name="ReceiveMessageRequest">
   <wsdl:part name="parameters" element="ecp:ReceiveMessageRequest"/>
  </wsdl:message>

  <wsdl:message name="ConfirmReceiveMessageRequest">
   <wsdl:part name="parameters" element="ecp:ConfirmReceiveMessageRequest"/>
  </wsdl:message>

  <wsdl:message name="ConnectivityTestRequest">
   <wsdl:part name="parameters" element="ecp:ConnectivityTestRequest"/>
  </wsdl:message>

  <wsdl:message name="CheckMessageStatusResponse">
   <wsdl:part name="parameters" element="ecp:CheckMessageStatusResponse"/>
  </wsdl:message>

  <wsdl:message name="ConfirmReceiveMessageResponse">
   <wsdl:part name="parameters" element="ecp:ConfirmReceiveMessageResponse"/>
  </wsdl:message>
```

```
1711
1712    <wsdl:message name="ReceiveMessageFault">
1713      <wsdl:part name="fault" element="ecp:ReceiveMessageError"/>
1714    </wsdl:message>
1715
1716    <wsdl:message name="CheckMessageStatusFault">
1717      <wsdl:part name="fault" element="ecp:CheckMessageStatusError"/>
1718    </wsdl:message>
1719
1720    <wsdl:message name="CheckMessageStatusRequest">
1721      <wsdl:part name="parameters" element="ecp:CheckMessageStatusRequest"/>
1722    </wsdl:message>
1723
1724    <wsdl:message name="ConfirmReceiveMessageFault">
1725      <wsdl:part name="fault" element="ecp:ConfirmReceiveMessageError"/>
1726    </wsdl:message>
1727
1728    <wsdl:message name="SendMessageFault">
1729      <wsdl:part name="fault" element="ecp:SendMessageError"/>
1730    </wsdl:message>
1731
1732    <wsdl:message name="ReceiveMessageResponse">
1733      <wsdl:part name="parameters" element="ecp:ReceiveMessageResponse"/>
1734    </wsdl:message>
1735
1736    <wsdl:message name="ConnectivityTestResponse">
1737      <wsdl:part name="parameters" element="ecp:ConnectivityTestResponse"/>
1738    </wsdl:message>
1739
1740    <wsdl:portType name="ECPEndpoint">
1741      <wsdl:operation name="SendMessage">
1742        <wsdl:input message="ecp:SendMessageRequest"/>
1743        <wsdl:output message="ecp:SendMessageResponse"/>
1744        <wsdl:fault name="SendMessageError" message="ecp:SendMessageFault"/>
1745      </wsdl:operation>
1746      <wsdl:operation name="ReceiveMessage">
1747        <wsdl:input message="ecp:ReceiveMessageRequest"/>
1748        <wsdl:output message="ecp:ReceiveMessageResponse"/>
1749        <wsdl:fault name="ReceiveMessageError" message="ecp:ReceiveMessageFault"/>
1750      </wsdl:operation>
1751      <wsdl:operation name="ConfirmReceiveMessage">
1752        <wsdl:input message="ecp:ConfirmReceiveMessageRequest"/>
1753        <wsdl:output message="ecp:ConfirmReceiveMessageResponse"/>
1754        <wsdl:fault name="ConfirmReceiveMessageError" message="ecp:ConfirmReceiveMessageFault"/>
1755      </wsdl:operation>
1756      <wsdl:operation name="ConnectivityTest">
1757        <wsdl:input message="ecp:ConnectivityTestRequest"/>
1758        <wsdl:output message="ecp:ConnectivityTestResponse"/>
1759        <wsdl:fault name="ConnectivityTestError" message="ecp:ConnectivityTestFault"/>
1760      </wsdl:operation>
1761      <wsdl:operation name="CheckMessageStatus">
1762        <wsdl:input message="ecp:CheckMessageStatusRequest"/>
1763        <wsdl:output message="ecp:CheckMessageStatusResponse"/>
1764        <wsdl:fault name="CheckMessageStatusError" message="ecp:CheckMessageStatusFault"/>
1765      </wsdl:operation>
1766    </wsdl:portType>
1767
1768    <wsdl:binding name="ECPEndpointSOAP12" type="ecp:ECPEndpoint">
1769      <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
1770      <wsdl:operation name="SendMessage">
1771        <soap12:operation soapAction="http://mades.entsoe.eu/SendMessage"/>
1772        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
1773        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
1774        <wsdl:fault name="SendMessageError"> <soap12:fault name="SendMessageError" use="literal"/>
1775  </wsdl:fault>
1776      </wsdl:operation>
```


ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

```
1777      <wsdl:operation name="ReceiveMessage">
1778        <soap12:operation soapAction="http://mades.entsoe.eu/ReceiveMessage"/>
1779        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
1780        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
1781        <wsdl:fault name="ReceiveMessageError"> <soap12:fault name="ReceiveMessageError" use="literal"/>
1782    </wsdl:fault>
1783      </wsdl:operation>
1784      <wsdl:operation name="ConfirmReceiveMessage">
1785        <soap12:operation soapAction="http://mades.entsoe.eu/ConfirmReceiveMessage"/>
1786        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
1787        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
1788        <wsdl:fault name="ConfirmReceiveMessageError"> <soap12:fault name="ConfirmReceiveMessageError"
1789    use="literal"/> </wsdl:fault>
1790      </wsdl:operation>
1791      <wsdl:operation name="ConnectivityTest">
1792        <soap12:operation soapAction="http://mades.entsoe.eu/ConnectivityTest"/>
1793        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
1794        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
1795        <wsdl:fault name="ConnectivityTestError"> <soap12:fault name="ConnectivityTestError" use="literal"/>
1796    </wsdl:fault>
1797      </wsdl:operation>
1798      <wsdl:operation name="CheckMessageStatus">
1799        <soap12:operation soapAction="http://mades.entsoe.eu/CheckMessageStatus"/>
1800        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
1801        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
1802        <wsdl:fault name="CheckMessageStatusError"> <soap12:fault name="CheckMessageStatusError"
1803    use="literal"/> </wsdl:fault>
1804      </wsdl:operation>
1805    </wsdl:binding>
1806
1807    <wsdl:binding name="ECPEndpointSOAP11" type="ecp:ECPEndpoint">
1808      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
1809      <wsdl:operation name="SendMessage">
1810        <soap:operation soapAction="http://mades.entsoe.eu/SendMessage"/>
1811        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
1812        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
1813        <wsdl:fault name="SendMessageError"> <soap:fault name="SendMessageError" use="literal"/> </wsdl:fault>
1814      </wsdl:operation>
1815      <wsdl:operation name="ReceiveMessage">
1816        <soap:operation soapAction="http://mades.entsoe.eu/ReceiveMessage"/>
1817        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
1818        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
1819        <wsdl:fault name="ReceiveMessageError"> <soap:fault name="ReceiveMessageError" use="literal"/>
1820    </wsdl:fault>
1821      </wsdl:operation>
1822      <wsdl:operation name="ConfirmReceiveMessage">
1823        <soap:operation soapAction="http://mades.entsoe.eu/ConfirmReceiveMessage"/>
1824        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
1825        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
1826        <wsdl:fault name="ConfirmReceiveMessageError"> <soap:fault name="ConfirmReceiveMessageError"
1827    use="literal"/> </wsdl:fault>
1828      </wsdl:operation>
1829      <wsdl:operation name="ConnectivityTest">
1830        <soap:operation soapAction="http://mades.entsoe.eu/ConnectivityTest"/>
1831        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
1832        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
1833        <wsdl:fault name="ConnectivityTestError"> <soap:fault name="ConnectivityTestError" use="literal"/>
1834    </wsdl:fault>
1835      </wsdl:operation>
1836      <wsdl:operation name="CheckMessageStatus">
1837        <soap:operation soapAction="http://mades.entsoe.eu/CheckMessageStatus"/>
1838        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
1839        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
1840        <wsdl:fault name="CheckMessageStatusError"> <soap:fault name="CheckMessageStatusError"
1841    use="literal"/> </wsdl:fault>
1842      </wsdl:operation>
```

```
1843      </wsdl:binding>
1844
1845      <wsdl:service name="ECPEndpointService">
1846        <wsdl:port name="ECPEndpointSOAP12" binding="ecp:ECPEndpointSOAP12">
1847          <soap12:address location="http://mades.entsoe.eu"/>
1848        </wsdl:port>
1849        <wsdl:port name="ECPEndpointSOAP11" binding="ecp:ECPEndpointSOAP11">
1850          <soap:address location="http://mades.entsoe.eu"/>
1851        </wsdl:port>
1852      </wsdl:service>
1853    </wsdl:definitions>
```


ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.6.2 GATEWAY & NODE INTERFACE

5.6.2.1 AUTHENTICATION SERVICE

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ECPAuthenticationService" targetNamespace="http://mades.entsoe.eu/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ecp="http://mades.entsoe.eu/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">

  <wsdl:types>
    <xsd:schema targetNamespace="http://mades.entsoe.eu/">

      <xsd:element name="GetAuthenticationTokenRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="componentCode" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="GetAuthenticationTokenResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="authToken" type="xsd:string"/>
            <xsd:element name="expiration" type="xsd:long"/>
            <xsd:element minOccurs="0" name="serviceMversion" nillable="true" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="GetAuthenticationTokenError">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="errorCode" type="xsd:string"/>
            <xsd:element name="errorID" type="xsd:string"/>
            <xsd:element name="errorMessage" type="xsd:string"/>
            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="GetAuthenticationTokenResponse">
    <wsdl:part name="parameters" element="ecp:GetAuthenticationTokenResponse"/>
  </wsdl:message>

  <wsdl:message name="GetAuthenticationTokenFault">
    <wsdl:part name="fault" element="ecp:GetAuthenticationTokenError"/>
  </wsdl:message>

  <wsdl:message name="GetAuthenticationTokenRequest">
    <wsdl:part name="parameters" element="ecp:GetAuthenticationTokenRequest"/>
  </wsdl:message>

  <wsdl:portType name="ECPAuthenticationService">
    <wsdl:operation name="GetAuthenticationToken">
      <wsdl:input message="ecp:GetAuthenticationTokenRequest"/>
      <wsdl:output message="ecp:GetAuthenticationTokenResponse"/>
      <wsdl:fault name="GetAuthenticationTokenError" message="ecp:GetAuthenticationTokenFault"/>
    </wsdl:operation>
  </wsdl:portType>
```


ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

```
1913
1914    <wsdl:binding name="ECPAuthenticationServiceSOAP12" type="ecp:ECPAuthenticationService">
1915      <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
1916      <wsdl:operation name="GetAuthenticationToken">
1917        <soap12:operation soapAction="http://mades.entsoe.eu/GetAuthenticationToken"/>
1918        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
1919        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
1920        <wsdl:fault name="GetAuthenticationTokenError"> <soap12:fault name="GetAuthenticationTokenError"
1921  use="literal"/> </wsdl:fault>
1922      </wsdl:operation>
1923    </wsdl:binding>
1924
1925    <wsdl:binding name="ECPAuthenticationServiceSOAP11" type="ecp:ECPAuthenticationService">
1926      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
1927      <wsdl:operation name="GetAuthenticationToken">
1928        <soap:operation soapAction="http://mades.entsoe.eu/GetAuthenticationToken"/>
1929        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
1930        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
1931        <wsdl:fault name="GetAuthenticationTokenError"> <soap:fault name="GetAuthenticationTokenError"
1932  use="literal"/> </wsdl:fault>
1933      </wsdl:operation>
1934    </wsdl:binding>
1935
1936    <wsdl:service name="ECPAuthenticationService">
1937      <wsdl:port name="ECPAuthenticationServiceSOAP12" binding="ecp:ECPAuthenticationServiceSOAP12">
1938        <soap12:address location="http://mades.entsoe.eu"/>
1939      </wsdl:port>
1940      <wsdl:port name="ECPAuthenticationServiceSOAP11" binding="ecp:ECPAuthenticationServiceSOAP11">
1941        <soap:address location="http://mades.entsoe.eu"/>
1942      </wsdl:port>
1943    </wsdl:service>
1944  </wsdl:definitions>
```


ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.6.2.2 MESSAGING SERVICES

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ECPInternalMessaging" targetNamespace="http://mades.entsoe.eu/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ecp="http://mades.entsoe.eu/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">

  <wsdl:types>
    <xsd:schema targetNamespace="http://mades.entsoe.eu/">

      <xsd:element name="UploadMessagesRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="unbounded" name="messages" type="ecp:InternalMessage"/>
            <xsd:element name="authToken" type="ecp:AuthenticationToken"/>
            <xsd:element minOccurs="0" name="serviceMversion" nillable="true" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="UploadMessagesResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="uploadedMessages" type="xsd:string"/>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="notUploadedMessages"
type="ecp:NotUploadedMessageResponse"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="UploadMessagesError">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="errorCode" type="xsd:string"/>
            <xsd:element name="errorID" type="xsd:string"/>
            <xsd:element name="errorMessage" type="xsd:string"/>
            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="DownloadMessagesRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="unbounded" name="endpoints" type="ecp:Endpoint">
            <xsd:element name="authToken" type="ecp:AuthenticationToken"/>
            <xsd:element minOccurs="0" name="serviceMversion" nillable="true" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="DownloadMessagesResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="messages" type="ecp:InternalMessage"/>
            <xsd:element name="waitingMessages" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="DownloadMessagesError">
        <xsd:complexType>
          <xsd:sequence>
```


ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

```
2007            <xsd:element name="errorCode" type="xsd:string"/>
2008            <xsd:element name="errorID" type="xsd:string"/>
2009            <xsd:element name="errorMessage" type="xsd:string"/>
2010            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2011          </xsd:sequence>
2012        </xsd:complexType>
2013      </xsd:element>
2014
2015      <xsd:element name="ConfirmDownloadRequest">
2016        <xsd:complexType>
2017          <xsd:sequence>
2018            <xsd:element maxOccurs="unbounded" minOccurs="0" name="messageIDs" type="xsd:string"/>
2019            <xsd:element name="authToken" type="ecp:AuthenticationToken"/>
2020            <xsd:element minOccurs="0" name="serviceMversion" nillable="true" type="xsd:int"/>
2021          </xsd:sequence>
2022        </xsd:complexType>
2023      </xsd:element>
2024
2025      <xsd:element name="ConfirmDownloadResponse">
2026        <xsd:complexType>
2027          <xsd:sequence>
2028            <xsd:element maxOccurs="unbounded" minOccurs="0" name="confirmedMessages" type="xsd:string"/>
2029            <xsd:element maxOccurs="unbounded" minOccurs="0" name="notConfirmedMessages"
2030    type="ecp:NotConfirmedMessageResponse"/>
2031          </xsd:sequence>
2032        </xsd:complexType>
2033      </xsd:element>
2034
2035      <xsd:element name="ConfirmDownloadError">
2036        <xsd:complexType>
2037          <xsd:sequence>
2038            <xsd:element name="errorCode" type="xsd:string"/>
2039            <xsd:element name="errorID" type="xsd:string"/>
2040            <xsd:element name="errorMessage" type="xsd:string"/>
2041            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2042          </xsd:sequence>
2043        </xsd:complexType>
2044      </xsd:element>
2045
2046      <xsd:complexType name="InternalMessage">
2047        <xsd:sequence>
2048          <xsd:element name="messageID" type="xsd:string"/>
2049          <xsd:element name="receiverCode" type="xsd:string"/>
2050          <xsd:element name="businessType" type="xsd:string"/>
2051          <xsd:element name="content" type="xsd:base64Binary"/>
2052          <xsd:element minOccurs="0" name="extension" nillable="true" type="xsd:string"/>
2053          <xsd:element name="generated" type="xsd:dateTime"/>
2054          <xsd:element minOccurs="0" name="expirationTime" nillable="true" type="xsd:long"/>
2055          <xsd:element name="senderCode" type="xsd:string"/>
2056          <xsd:element name="senderDescription" type="xsd:string"/>
2057          <xsd:element name="internalType" type="ecp:InternalMessageType"/>
2058          <xsd:element minOccurs="0" name="relatedMessageID" nillable="true" type="xsd:string"/>
2059          <xsd:element minOccurs="0" name="senderApplication" nillable="true" type="xsd:string"/>
2060          <xsd:element minOccurs="0" name="baMessageID" nillable="true" type="xsd:string"/>
2061          <xsd:element name="metadata" type="ecp:MessageMetadata"/>
2062          <xsd:element minOccurs="0" name="messageMversion" nillable="true" type="xsd:int"/>
2063        </xsd:sequence>
2064      </xsd:complexType>
2065
2066      <xsd:simpleType name="InternalMessageType">
2067        <xsd:restriction base="xsd:string">
2068          <xsd:enumeration value="STANDARD_MESSAGE"/>
2069          <xsd:enumeration value="DELIVERY_ACKNOWLEDGEMENT"/>
2070          <xsd:enumeration value="RECEIVE_ACKNOWLEDGEMENT"/>
2071          <xsd:enumeration value="FAILURE_ACKNOWLEDGEMENT"/>
2072          <xsd:enumeration value="TRACING_MESSAGE"/>
```

```
2073            <xsd:enumeration value="TRACING_ACKNOWLEDGEMENT"/>
2074          </xsd:restriction>
2075        </xsd:simpleType>
2076
2077        <xsd:complexType name="MessageMetadata">
2078          <xsd:sequence>
2079            <xsd:element maxOccurs="unbounded" minOccurs="0" name="messageProcessors"
2080    type="ecp:MessageProcessor"/>
2081          </xsd:sequence>
2082        </xsd:complexType>
2083
2084        <xsd:complexType name="MessageProcessor">
2085          <xsd:sequence>
2086            <xsd:element name="processorID" type="xsd:string"/>
2087            <xsd:element name="processorData" type="ecp:Map"/>
2088          </xsd:sequence>
2089        </xsd:complexType>
2090
2091        <xsd:complexType name="NotUploadedMessageResponse">
2092          <xsd:sequence>
2093            <xsd:element name="messageID" type="xsd:string"/>
2094            <xsd:element name="fatal" type="xsd:boolean"/>
2095            <xsd:element minOccurs="0" name="businessErrorMessage" nillable="true" type="xsd:string"/>
2096            <xsd:element name="errorCode" type="xsd:string"/>
2097            <xsd:element name="errorID" type="xsd:string"/>
2098            <xsd:element name="errorMessage" type="xsd:string"/>
2099            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2100          </xsd:sequence>
2101        </xsd:complexType>
2102
2103        <xsd:complexType name="NotConfirmedMessageResponse">
2104          <xsd:sequence>
2105            <xsd:element name="messageID" type="xsd:string"/>
2106            <xsd:element name="errorCode" type="xsd:string"/>
2107            <xsd:element name="errorID" type="xsd:string"/>
2108            <xsd:element name="errorMessage" type="xsd:string"/>
2109            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2110          </xsd:sequence>
2111        </xsd:complexType>
2112
2113        <xsd:complexType name="Endpoint">
2114          <xsd:sequence>
2115            <xsd:element name="code" type="xsd:string"/>
2116            <xsd:element name="signature" type="xsd:string"/>
2117            <xsd:element name="certificateID" type="xsd:string"/>
2118          </xsd:sequence>
2119        </xsd:complexType>
2120
2121        <xsd:complexType name="AuthenticationToken">
2122          <xsd:sequence>
2123            <xsd:element name="token" type="xsd:string"/>
2124            <xsd:element name="signature" type="xsd:string"/>
2125            <xsd:element name="certificateID" type="xsd:string"/>
2126          </xsd:sequence>
2127        </xsd:complexType>
2128
2129        <xsd:complexType name="Map">
2130          <xsd:sequence>
2131            <xsd:element maxOccurs="unbounded" minOccurs="0" name="entries" type="ecp:MapEntry"/>
2132          </xsd:sequence>
2133        </xsd:complexType>
2134
2135        <xsd:complexType name="MapEntry">
2136          <xsd:sequence>
2137            <xsd:element name="key" type="xsd:string"/>
2138            <xsd:element name="type" type="ecp:ValueType"/>
```

```
2139            <xsd:element name="value" type="xsd:string"/>
2140          </xsd:sequence>
2141        </xsd:complexType>
2142
2143        <xsd:simpleType name="ValueType">
2144          <xsd:restriction base="xsd:string">
2145            <xsd:enumeration value="STRING"/>
2146            <xsd:enumeration value="LONG"/>
2147            <xsd:enumeration value="BYTE_ARRAY"/>
2148            <xsd:enumeration value="BOOLEAN"/>
2149          </xsd:restriction>
2150        </xsd:simpleType>
2151      </xsd:schema>
2152    </wsdl:types>
2153
2154    <wsdl:message name="ConfirmDownloadResponse">
2155      <wsdl:part name="parameters" element="ecp:ConfirmDownloadResponse"/>
2156    </wsdl:message>
2157
2158    <wsdl:message name="UploadMessagesFault">
2159      <wsdl:part name="fault" element="ecp:UploadMessagesError"/>
2160    </wsdl:message>
2161
2162    <wsdl:message name="UploadMessagesRequest">
2163      <wsdl:part name="parameters" element="ecp:UploadMessagesRequest"/>
2164    </wsdl:message>
2165
2166    <wsdl:message name="DownloadMessagesFault">
2167      <wsdl:part name="fault" element="ecp:DownloadMessagesError"/>
2168    </wsdl:message>
2169
2170    <wsdl:message name="UploadMessagesResponse">
2171      <wsdl:part name="parameters" element="ecp:UploadMessagesResponse"/>
2172    </wsdl:message>
2173
2174    <wsdl:message name="DownloadMessagesRequest">
2175      <wsdl:part name="parameters" element="ecp:DownloadMessagesRequest"/>
2176    </wsdl:message>
2177
2178    <wsdl:message name="ConfirmDownloadFault">
2179      <wsdl:part name="fault" element="ecp:ConfirmDownloadError"/>
2180    </wsdl:message>
2181
2182    <wsdl:message name="DownloadMessagesResponse">
2183      <wsdl:part name="parameters" element="ecp:DownloadMessagesResponse"/>
2184    </wsdl:message>
2185
2186    <wsdl:message name="ConfirmDownloadRequest">
2187      <wsdl:part name="parameters" element="ecp:ConfirmDownloadRequest"/>
2188    </wsdl:message>
2189
2190    <wsdl:portType name="ECPInternalMessaging">
2191      <wsdl:operation name="UploadMessages">
2192        <wsdl:input message="ecp:UploadMessagesRequest"/>
2193        <wsdl:output message="ecp:UploadMessagesResponse"/>
2194        <wsdl:fault name="UploadMessagesError" message="ecp:UploadMessagesFault"/>
2195      </wsdl:operation>
2196      <wsdl:operation name="DownloadMessages">
2197        <wsdl:input message="ecp:DownloadMessagesRequest"/>
2198        <wsdl:output message="ecp:DownloadMessagesResponse"/>
2199        <wsdl:fault name="DownloadMessagesError" message="ecp:DownloadMessagesFault"/>
2200      </wsdl:operation>
2201      <wsdl:operation name="ConfirmDownload">
2202        <wsdl:input message="ecp:ConfirmDownloadRequest"/>
2203        <wsdl:output message="ecp:ConfirmDownloadResponse"/>
2204        <wsdl:fault name="ConfirmDownloadError" message="ecp:ConfirmDownloadFault"/>
```

```
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="ECPInternalMessagingSOAP11" type="ecp:ECPInternalMessaging">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="UploadMessages">
      <soap:operation soapAction="http://mades.entsoe.eu/UploadMessages"/>
      <wsdl:input> <soap:body use="literal"/> </wsdl:input>
      <wsdl:output> <soap:body use="literal"/> </wsdl:output>
      <wsdl:fault name="UploadMessagesError"> <soap:fault name="UploadMessagesError" use="literal"/>
</wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="DownloadMessages">
      <soap:operation soapAction="http://mades.entsoe.eu/DownloadMessages"/>
      <wsdl:input> <soap:body use="literal"/> </wsdl:input>
      <wsdl:output> <soap:body use="literal"/> </wsdl:output>
      <wsdl:fault name="DownloadMessagesError"> <soap:fault name="DownloadMessagesError" use="literal"/>
</wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="ConfirmDownload">
      <soap:operation soapAction="http://mades.entsoe.eu/ConfirmDownload"/>
      <wsdl:input> <soap:body use="literal"/> </wsdl:input>
      <wsdl:output> <soap:body use="literal"/> </wsdl:output>
      <wsdl:fault name="ConfirmDownloadError"> <soap:fault name="ConfirmDownloadError" use="literal"/>
</wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:binding name="ECPInternalMessagingSOAP12" type="ecp:ECPInternalMessaging">
    <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="UploadMessages">
      <soap12:operation soapAction="http://mades.entsoe.eu/UploadMessages"/>
      <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
      <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
      <wsdl:fault name="UploadMessagesError"> <soap12:fault name="UploadMessagesError" use="literal"/>
</wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="DownloadMessages">
      <soap12:operation soapAction="http://mades.entsoe.eu/DownloadMessages"/>
      <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
      <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
      <wsdl:fault name="DownloadMessagesError"> <soap12:fault name="DownloadMessagesError" use="literal"/>
</wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="ConfirmDownload">
      <soap12:operation soapAction="http://mades.entsoe.eu/ConfirmDownload"/>
      <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
      <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
      <wsdl:fault name="ConfirmDownloadError"> <soap12:fault name="ConfirmDownloadError" use="literal"/>
</wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="ECPInternalMessagingService">
    <wsdl:port name="ECPInternalMessagingSOAP11" binding="ecp:ECPInternalMessagingSOAP11">
      <soap:address location="http://mades.entsoe.eu"/>
    </wsdl:port>
    <wsdl:port name="ECPInternalMessagingSOAP12" binding="ecp:ECPInternalMessagingSOAP12">
      <soap12:address location="http://mades.entsoe.eu"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```


ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.6.2.3 DIRECTORY SERVICES

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ECPDirectoryService" targetNamespace="http://mades.entsoe.eu/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ecp="http://mades.entsoe.eu/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">

 <wsdl:types>
  <xsd:schema targetNamespace="http://mades.entsoe.eu/">

   <xsd:element name="GetCertificateRequest">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="componentCode" type="xsd:string"/>
      <xsd:element name="type" type="ecp:CertificateType"/>
      <xsd:element minOccurs="0" name="certificateID" nillable="true" type="xsd:string"/>
      <xsd:element name="authToken" type="ecp:AuthenticationToken"/>
      <xsd:element minOccurs="0" name="serviceMversion" nillable="true" type="xsd:int"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="GetCertificateResponse">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element minOccurs="0" name="certificate" nillable="true" type="ecp:Certificate"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="GetCertificateError">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="errorCode" type="xsd:string"/>
      <xsd:element name="errorID" type="xsd:string"/>
      <xsd:element name="errorMessage" type="xsd:string"/>
      <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="GetComponentRequest">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="componentCode" type="xsd:string"/>
      <xsd:element name="authToken" type="ecp:AuthenticationToken"/>
      <xsd:element minOccurs="0" name="serviceMversion" nillable="true" type="xsd:int"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="GetComponentResponse">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element minOccurs="0" name="component" nillable="true" type="ecp:ComponentInformation"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="GetComponentError">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="errorCode" type="xsd:string"/>
```

```
2329            <xsd:element name="errorID" type="xsd:string"/>
2330            <xsd:element name="errorMessage" type="xsd:string"/>
2331            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2332          </xsd:sequence>
2333        </xsd:complexType>
2334      </xsd:element>
2335
2336      <xsd:element name="SetComponentMversionRequest">
2337        <xsd:complexType>
2338          <xsd:sequence>
2339            <xsd:element name="componentCode" type="xsd:string"/>
2340            <xsd:element name="signature" type="xsd:string"/>
2341            <xsd:element name="certificateID" type="xsd:string"/>
2342            <xsd:element name="componentMVersion" type="xsd:int"/>
2343            <xsd:element name="authToken" type="ecp:AuthenticationToken"/>
2344            <xsd:element minOccurs="0" name="serviceMversion" nillable="true" type="xsd:int"/>
2345          </xsd:sequence>
2346        </xsd:complexType>
2347      </xsd:element>
2348
2349      <xsd:element name="SetComponentMversionResponse">
2350        <xsd:complexType>
2351          <xsd:sequence>
2352            <xsd:element name="nodeMversion" type="xsd:int"/>
2353            <xsd:element name="acceptance" type="xsd:boolean"/>
2354          </xsd:sequence>
2355        </xsd:complexType>
2356      </xsd:element>
2357
2358      <xsd:element name="SetComponentMversionError">
2359        <xsd:complexType>
2360          <xsd:sequence>
2361            <xsd:element name="errorCode" type="xsd:string"/>
2362            <xsd:element name="errorID" type="xsd:string"/>
2363            <xsd:element name="errorMessage" type="xsd:string"/>
2364            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2365          </xsd:sequence>
2366        </xsd:complexType>
2367      </xsd:element>
2368
2369      <xsd:complexType name="Certificate">
2370        <xsd:sequence>
2371          <xsd:element name="certificateID" type="xsd:string"/>
2372          <xsd:element name="certificate" type="xsd:base64Binary"/>
2373          <xsd:element name="expiration" type="xsd:long"/>
2374        </xsd:sequence>
2375      </xsd:complexType>
2376
2377      <xsd:simpleType name="CertificateType">
2378        <xsd:restriction base="xsd:string">
2379          <xsd:enumeration value="AUTHENTICATION"/>
2380          <xsd:enumeration value="ENCRYPTION"/>
2381          <xsd:enumeration value="SIGNING"/>
2382        </xsd:restriction>
2383      </xsd:simpleType>
2384
2385      <xsd:complexType name="ComponentInformation">
2386        <xsd:sequence>
2387          <xsd:element name="code" type="xsd:string"/>
2388          <xsd:element name="type" type="ecp:ComponentType"/>
2389          <xsd:element name="organization" type="xsd:string"/>
2390          <xsd:element name="person" type="xsd:string"/>
2391          <xsd:element name="email" type="xsd:string"/>
2392          <xsd:element name="phone" type="xsd:string"/>
2393          <xsd:element name="routing" type="ecp:RoutingInformation"/>
2394          <xsd:element minOccurs="0" name="expiration" nillable="true" type="xsd:long"/>
```

```
2395          <xsd:element minOccurs="0" name="codeMversion" nillable="true" type="xsd:int"/>
2396        </xsd:sequence>
2397      </xsd:complexType>
2398
2399      <xsd:complexType name="RoutingInformation">
2400        <xsd:sequence>
2401          <xsd:element name="node" type="xsd:string"/>
2402          <xsd:element name="primaryURL" type="xsd:string"/>
2403          <xsd:element minOccurs="0" name="secondaryURL" nillable="true" type="xsd:string"/>
2404          <xsd:element minOccurs="0" name="nodeMversion" nillable="true" type="xsd:int"/>
2405        </xsd:sequence>
2406      </xsd:complexType>
2407
2408      <xsd:simpleType name="ComponentType">
2409        <xsd:restriction base="xsd:string">
2410          <xsd:enumeration value="NODE"/>
2411          <xsd:enumeration value="GATEWAY"/>
2412          <xsd:enumeration value="ENDPOINT"/>
2413        </xsd:restriction>
2414      </xsd:simpleType>
2415
2416      <xsd:complexType name="AuthenticationToken">
2417        <xsd:sequence>
2418          <xsd:element name="token" type="xsd:string"/>
2419          <xsd:element name="signature" type="xsd:string"/>
2420          <xsd:element name="certificateID" type="xsd:string"/>
2421        </xsd:sequence>
2422      </xsd:complexType>
2423    </xsd:schema>
2424  </wsdl:types>
2425
2426  <wsdl:message name="GetComponentRequest">
2427    <wsdl:part name="parameters" element="ecp:GetComponentRequest"/>
2428  </wsdl:message>
2429
2430  <wsdl:message name="GetCertificateResponse">
2431    <wsdl:part name="parameters" element="ecp:GetCertificateResponse"/>
2432  </wsdl:message>
2433
2434  <wsdl:message name="GetComponentResponse">
2435    <wsdl:part name="parameters" element="ecp:GetComponentResponse"/>
2436  </wsdl:message>
2437
2438  <wsdl:message name="GetCertificateRequest">
2439    <wsdl:part name="parameters" element="ecp:GetCertificateRequest"/>
2440  </wsdl:message>
2441
2442  <wsdl:message name="GetCertificateFault">
2443    <wsdl:part name="fault" element="ecp:GetCertificateError"/>
2444  </wsdl:message>
2445
2446  <wsdl:message name="GetComponentFault">
2447    <wsdl:part name="fault" element="ecp:GetComponentError"/>
2448  </wsdl:message>
2449
2450  <wsdl:message name="SetComponentMversionRequest">
2451    <wsdl:part name="parameters" element="ecp:SetComponentMversionRequest"/>
2452  </wsdl:message>
2453
2454  <wsdl:message name="SetComponentMversionResponse">
2455    <wsdl:part name="parameters" element="ecp:SetComponentMversionResponse"/>
2456  </wsdl:message>
2457
2458  <wsdl:message name="SetComponentMversionFault">
2459    <wsdl:part name="fault" element="ecp:SetComponentMversionError"/>
2460  </wsdl:message>
```

```
2461
2462    <wsdl:portType name="ECPDirectoryService">
2463      <wsdl:operation name="GetCertificate">
2464        <wsdl:input message="ecp:GetCertificateRequest"/>
2465        <wsdl:output message="ecp:GetCertificateResponse"/>
2466        <wsdl:fault name="GetCertificateError" message="ecp:GetCertificateFault"/>
2467      </wsdl:operation>
2468      <wsdl:operation name="GetComponent">
2469        <wsdl:input message="ecp:GetComponentRequest"/>
2470        <wsdl:output message="ecp:GetComponentResponse"/>
2471        <wsdl:fault name="GetComponentError" message="ecp:GetComponentFault"/>
2472      </wsdl:operation>
2473      <wsdl:operation name="SetComponentMversion">
2474        <wsdl:input message="ecp:SetComponentMversionRequest"/>
2475        <wsdl:output message="ecp:SetComponentMversionResponse"/>
2476        <wsdl:fault name="SetComponentMversionError" message="ecp:SetComponentMversionFault"/>
2477      </wsdl:operation>
2478    </wsdl:portType>
2479
2480    <wsdl:binding name="ECPDirectoryServiceSOAP11" type="ecp:ECPDirectoryService">
2481      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2482      <wsdl:operation name="GetCertificate">
2483        <soap:operation soapAction="http://mades.entsoe.eu/GetCertificate"/>
2484        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2485        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2486        <wsdl:fault name="GetCertificateError"> <soap:fault name="GetCertificateError" use="literal"/> </wsdl:fault>
2487      </wsdl:operation>
2488      <wsdl:operation name="GetComponent">
2489        <soap:operation soapAction="http://mades.entsoe.eu/GetComponent"/>
2490        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2491        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2492        <wsdl:fault name="GetComponentError"> <soap:fault name="GetComponentError" use="literal"/>
2493    </wsdl:fault>
2494      </wsdl:operation>
2495      <wsdl:operation name="SetComponentMversion">
2496        <soap:operation soapAction="http://mades.entsoe.eu/SetComponentMversion"/>
2497        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2498        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2499        <wsdl:fault name="SetComponentMversionError"> <soap:fault name="SetComponentMversionError"
2500    use="literal"/> </wsdl:fault>
2501      </wsdl:operation>
2502    </wsdl:binding>
2503
2504    <wsdl:binding name="ECPDirectoryServiceSOAP12" type="ecp:ECPDirectoryService">
2505      <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2506      <wsdl:operation name="GetCertificate">
2507        <soap12:operation soapAction="http://mades.entsoe.eu/GetCertificate"/>
2508        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2509        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2510        <wsdl:fault name="GetCertificateError"> <soap12:fault name="GetCertificateError" use="literal"/>
2511    </wsdl:fault>
2512      </wsdl:operation>
2513      <wsdl:operation name="GetComponent">
2514        <soap12:operation soapAction="http://mades.entsoe.eu/GetComponent"/>
2515        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2516        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2517        <wsdl:fault name="GetComponentError"> <soap12:fault name="GetComponentError" use="literal"/>
2518    </wsdl:fault>
2519      </wsdl:operation>
2520      <wsdl:operation name="SetComponentMversion">
2521        <soap12:operation soapAction="http://mades.entsoe.eu/SetComponentMversion"/>
2522        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2523        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2524        <wsdl:fault name="SetComponentMversionError"> <soap12:fault name="SetComponentMversionError"
2525    use="literal"/> </wsdl:fault>
2526      </wsdl:operation>
```


ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

```
2527      </wsdl:binding>
2528
2529      <wsdl:service name="ECPDirectoryService">
2530        <wsdl:port name="ECPDirectoryServiceSOAP11" binding="ecp:ECPDirectoryServiceSOAP11">
2531          <soap:address location="http://mades.entsoe.eu"/>
2532        </wsdl:port>
2533        <wsdl:port name="ECPDirectoryServiceSOAP12" binding="ecp:ECPDirectoryServiceSOAP12">
2534          <soap12:address location="http://mades.entsoe.eu"/>
2535        </wsdl:port>
2536      </wsdl:service>
2537    </wsdl:definitions>
```


ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu

5.6.2.4 NODE SYNCHRONIZATION INTERFACE

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ECPNodeSynchronizationService" targetNamespace="http://mades.entsoe.eu/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ecp="http://mades.entsoe.eu/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">

  <wsdl:types>
    <xsd:schema targetNamespace="http://mades.entsoe.eu/">

      <xsd:element name="GetAllDirectoryDataRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element minOccurs="0" name="dversion" nillable="true" type="xsd:int"/>
            <xsd:element minOccurs="0" name="serviceMversion" nillable="true" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="GetAllDirectoryDataResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="dversion" type="xsd:int"/>
            <xsd:element name="nodeCode" type="xsd:string"/>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="components" nillable="true"
type="ecp:ComponentDescription"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="GetAllDirectoryDataError">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="errorCode" type="xsd:string"/>
            <xsd:element name="errorID" type="xsd:string"/>
            <xsd:element name="errorMessage" type="xsd:string"/>
            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="GetNodeMversionRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="mversion" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="GetNodeMversionResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="mversion" type="xsd:int"/>
            <xsd:element name="nodeCode" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="GetNodeMversionError">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="errorCode" type="xsd:string"/>
            <xsd:element name="errorID" type="xsd:string"/>
```

```
2600            <xsd:element name="errorMessage" type="xsd:string"/>
2601            <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2602          </xsd:sequence>
2603        </xsd:complexType>
2604      </xsd:element>
2605
2606      <xsd:complexType name="ComponentDescription">
2607        <xsd:sequence>
2608          <xsd:element name="information" type="ecp:ComponentInformation"/>
2609          <xsd:element maxOccurs="unbounded" minOccurs="0" name="certificates"
2610  type="ecp:ComponentCertificate"/>
2611        </xsd:sequence>
2612      </xsd:complexType>
2613
2614      <xsd:complexType name="ComponentCertificate">
2615        <xsd:sequence>
2616          <xsd:element name="certificate" type="ecp:Certificate"/>
2617          <xsd:element minOccurs="0" name="revoked" nillable="true" type="xsd:boolean"/>
2618          <xsd:element name="type" type="ecp:CertificateType"/>
2619        </xsd:sequence>
2620      </xsd:complexType>
2621
2622      <xsd:complexType name="Certificate">
2623        <xsd:sequence>
2624          <xsd:element name="certificateID" type="xsd:string"/>
2625          <xsd:element name="certificate" type="xsd:base64Binary"/>
2626        </xsd:sequence>
2627      </xsd:complexType>
2628
2629      <xsd:simpleType name="CertificateType">
2630        <xsd:restriction base="xsd:string">
2631          <xsd:enumeration value="AUTHENTICATION"/>
2632          <xsd:enumeration value="ENCRYPTION"/>
2633          <xsd:enumeration value="SIGNING"/>
2634        </xsd:restriction>
2635      </xsd:simpleType>
2636
2637      <xsd:complexType name="ComponentInformation">
2638        <xsd:sequence>
2639          <xsd:element name="code" type="xsd:string"/>
2640          <xsd:element name="type" type="ecp:ComponentType"/>
2641          <xsd:element name="organization" type="xsd:string"/>
2642          <xsd:element name="person" type="xsd:string"/>
2643          <xsd:element name="email" type="xsd:string"/>
2644          <xsd:element name="phone" type="xsd:string"/>
2645          <xsd:element name="routing" type="ecp:RoutingInformation"/>
2646          <xsd:element minOccurs="0" name="expiration" nillable="true" type="xsd:long"/>
2647          <xsd:element minOccurs="0" name="codeMversion" nillable="true" type="xsd:int"/>
2648        </xsd:sequence>
2649      </xsd:complexType>
2650
2651      <xsd:complexType name="RoutingInformation">
2652        <xsd:sequence>
2653          <xsd:element name="node" type="xsd:string"/>
2654          <xsd:element name="primaryURL" type="xsd:string"/>
2655          <xsd:element minOccurs="0" name="secondaryURL" nillable="true" type="xsd:string"/>
2656          <xsd:element minOccurs="0" name="nodeMversion" nillable="true" type="xsd:int"/>
2657        </xsd:sequence>
2658      </xsd:complexType>
2659
2660      <xsd:simpleType name="ComponentType">
2661        <xsd:restriction base="xsd:string">
2662          <xsd:enumeration value="NODE"/>
2663          <xsd:enumeration value="GATEWAY"/>
2664          <xsd:enumeration value="ENDPOINT"/>
2665        </xsd:restriction>
```

```
      </xsd:simpleType>

    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="GetAllDirectoryDataResponse">
    <wsdl:part name="parameters" element="ecp:GetAllDirectoryDataResponse"/>
  </wsdl:message>

  <wsdl:message name="GetAllDirectoryDataFault">
    <wsdl:part name="fault" element="ecp:GetAllDirectoryDataError"/>
  </wsdl:message>

  <wsdl:message name="GetAllDirectoryDataRequest">
    <wsdl:part name="parameters" element="ecp:GetAllDirectoryDataRequest"/>
  </wsdl:message>

  <wsdl:message name="GetNodeMversionResponse">
    <wsdl:part name="parameters" element="ecp:GetNodeMversionResponse"/>
  </wsdl:message>

  <wsdl:message name="GetNodeMversionFault">
    <wsdl:part name="fault" element="ecp:GetNodeMversionError"/>
  </wsdl:message>

  <wsdl:message name="GetNodeMversionRequest">
    <wsdl:part name="parameters" element="ecp:GetNodeMversionRequest"/>
  </wsdl:message>

  <wsdl:portType name="ECPNodeSynchronizationService">
    <wsdl:operation name="GetAllDirectoryData">
      <wsdl:input message="ecp:GetAllDirectoryDataRequest"/>
      <wsdl:output message="ecp:GetAllDirectoryDataResponse"/>
      <wsdl:fault name="GetAllDirectoryDataError" message="ecp:GetAllDirectoryDataFault"/>
    </wsdl:operation>
    <wsdl:operation name="GetNodeMversion">
      <wsdl:input message="ecp:GetNodeMversionRequest"/>
      <wsdl:output message="ecp:GetNodeMversionResponse"/>
      <wsdl:fault name="GetNodeMversionError" message="ecp:GetNodeMversionFault"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="ECPNodeSynchronizationServiceSOAP12" type="ecp:ECPNodeSynchronizationService">
    <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetAllDirectoryData">
      <soap12:operation soapAction="http://mades.entsoe.eu/GetAllDirectoryData"/>
      <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
      <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
      <wsdl:fault name="GetAllDirectoryDataError"> <soap12:fault name="GetAllDirectoryDataError" use="literal"/>
</wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="GetNodeMversion">
      <soap12:operation soapAction="http://mades.entsoe.eu/GetNodeMversion"/>
      <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
      <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
      <wsdl:fault name="GetNodeMversionError"> <soap12:fault name="GetNodeMversionError" use="literal"/>
</wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:binding name="ECPNodeSynchronizationServiceSOAP11" type="ecp:ECPNodeSynchronizationService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetAllDirectoryData">
      <soap:operation soapAction="http://mades.entsoe.eu/GetAllDirectoryData"/>
      <wsdl:input> <soap:body use="literal"/> </wsdl:input>
      <wsdl:output> <soap:body use="literal"/> </wsdl:output>
```

```
2732        <wsdl:fault name="GetAllDirectoryDataError"> <soap:fault name="GetAllDirectoryDataError" use="literal"/>
2733    </wsdl:fault>
2734      </wsdl:operation>
2735      <wsdl:operation name="GetNodeMversion">
2736        <soap:operation soapAction="http://mades.entsoe.eu/GetNodeMversion"/>
2737        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2738        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2739        <wsdl:fault name="GetNodeMversionError"> <soap:fault name="GetNodeMversionError" use="literal"/>
2740    </wsdl:fault>
2741      </wsdl:operation>
2742    </wsdl:binding>
2743
2744    <wsdl:service name="ECPNodeSynchronizationService">
2745      <wsdl:port name="ECPNodeSynchronizationServiceSOAP12"
2746    binding="ecp:ECPNodeSynchronizationServiceSOAP12">
2747        <soap12:address location="http://mades.entsoe.eu"/>
2748      </wsdl:port>
2749      <wsdl:port name="ECPNodeSynchronizationServiceSOAP11"
2750    binding="ecp:ECPNodeSynchronizationServiceSOAP11">
2751        <soap:address location="http://mades.entsoe.eu"/>
2752      </wsdl:port>
2753    </wsdl:service>
2754    </wsdl:definitions>
```

5.6.3 XML SIGNATURE EXAMPLE

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha512"/>
    <Reference URI="">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha512"/>
      <DigestValue>eVpInNsCIWzEjdrxxvongO2rnQ4=</DigestValue>
    </Reference>
  </SignedInfo>
     <SignatureValue>aD9HNiTmVxW+HnDOpSjzwDB+MypGTC7yb3/HUpAZKmEhRwQC0eBwYcZSRTqF8VdzmneH6abq2P+m
vHNXPC53i3mF58XDR5JFHHWLHq8B9HZm6/IYxcNy2cGW9yAVyQKe3uJXeV/95u9qMEwJhbOjvPIx
ZdbXqcCSorWqih7hdB86Nv2SIBfXMvWdIinwZfU/44RUptNyxQpP/Pw91Dd8YnMTNVwm2ax5oL1W
akIGKToS/yYid/Cgyb1xhGdfXEp30bqLusaLMYkbctpZ2WDn2w5I4mmOO78jndPUnaMT5gyFaonz
+K84xD+1/tZbTQ0adc9LE7XgAkpiiNjf2LW9tw==</SignatureValue>
  <KeyInfo>
    <KeyName>yyyyyyyyyyyyyyy</KeyName>
  </KeyInfo>
</Signature>
```

Where:

- DigestValue is the non encoded hash of the message.
- SignatureValue is the encoded hash of the message.
- KeyName is the ID of the signer component.

~°~°~°~°~°~°~°~°~° End of document ~°~°~°~°~°~°~°~°~°~°

ENTSO-E AISBL • Avenue de Cortenbergh, 100 • 1000 Brussels • Belgium • Tel +32 2 741 09 50 • Fax +32 2 741 09 51 • info@entsoe.eu • www.entsoe.eu