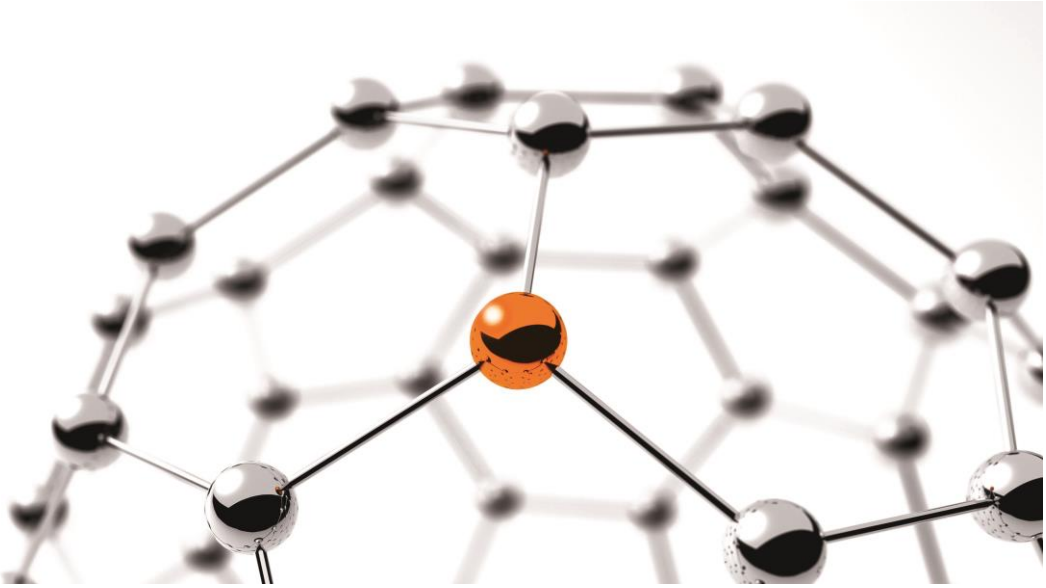
Specification

Smart Home IP (SHIP)

Version 1.0.1 (RC1)
2019-09-24



Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Terms of use for publications of EEBus Initiative e.V.

General information
The specifications, particulars, documents, publications and other information provided by the EEBus Initiative e.V. are solely for general informational purposes. Particularly specifications that have not been submitted to national or international standardisation organisations by EEBus Initiative e.V. (such as DKE/DIN-VDE or IEC/CENELEC/ETSI) are versions that have not yet undergone complete testing and can therefore only be considered as preliminary information. Even versions that have already been published via standardisation organisations can contain errors and will undergo further improvements and updates in future.

Liability
EEBus Initiative e.V. does not assume liability or provide a guarantee for the accuracy, completeness or up-to-date status of any specifications, data, documents, publications or other information provided and particularly for the functionality of any developments based on the above.

Copyright, rights of use and exploitation
The specifications provided are protected by copyright. Parts of the specifications have been submitted to national or international standardisation organisations by EEBus Initiative e.V., such as DKE/DIN-VDE or IEC/CENELEC/ETSI, etc. Furthermore, all rights to use and/or exploit the specifications belong to the EEBus Initiative e.V., Butzweilerhofallee 4, 50829 Cologne, Germany and can be used in accordance with the following regulations:

The use of the specifications for informational purposes is allowed. It is therefore permitted to use information evident from the contents of the specifications. In particular, the user is permitted to offer products, developments and/or services based on the specifications.

Any respective use relating to standardisation measures by the user or third parties is prohibited. In fact, the specifications may only be used by EEBus Initiative e.V. for standardisation purposes. The same applies to their use within the framework of alliances and/or cooperations that pursue the aim of determining uniform standards.

Any use not in accordance with the purpose intended by EEBus Initiative e.V. is also prohibited.

Furthermore, it is prohibited to edit, change or falsify the content of the specifications. The dissemination of the specifications in a changed, revised or falsified form is also prohibited. The same applies to the publication of extracts if they distort the literal meaning of the specifications as a whole.

It is prohibited to pass on the specifications to third parties without reference to these rights of use and exploitation.

It is also prohibited to pass on the specifications to third parties without informing them of the authorship or source.

Without the prior consent of EEBus Initiative e.V., all forms of use and exploitation not explicitly stated above are prohibited.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Introduction

Over the past decades, different home automation technologies have been created, which connect devices using digital communication technologies. Most of these solutions bring along an infrastructure of their own, like dedicated home automation wires. These approaches are acceptable for commercial and industrial buildings, but they are too complex for private homes, especially if retrofitting into already existing infrastructure is necessary. For these cases, wireless technologies have been introduced to make installation easier.

In the meantime, even private homes have been expanded with IP (Internet Protocol) based installations by home or flat owners. IP based devices fitting different consumer needs have become increasingly popular over the past years. This means that most likely, a communication infrastructure is already available in private households. Additionally, there are a lot of potential extensions to other domains than just home automation, since smart phones, PCs, cloud communication, etc. continuously broaden the horizon of possible applications.

However, there is a need for a secure standardized TCP/IP protocol based on requirements for the next generation network within the Internet of Things (IoT). Things, in the IoT, can refer to a wide variety of devices and will bring a lot of additional possibilities, e.g. within home automation, Smart Grid, Smart Home or Ambient Assisted Living (AAL).

This specification describes an IP based approach for plug and play home automation, which can easily be extended to additional domains. The solution is called SHIP (Smart Home IP), with the communicating devices being called SHIP nodes.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

CONTENTS

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

List of Figures

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

List of Tables

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

1 Scope

This document describes an IP (Internet Protocol) based approach for machine-to-machine communication.

It describes all relevant mechanisms between the Network Layer (layer 3) and Application Layer (layer 7) based on the seven-layer ISO-OSI model.

The goal is to obtain a secure TCP/IP-based solution that allows interoperable connectivity between different implementers and vendors.

Communication security is in line with the Smart Meter Gateway HAN (Home Area Network) interface as described by the Federal Office for Information Security Germany (BSI) in TR-03109 Version 1.0, while also providing scalability, a high degree of usability, and efficient mechanisms for simple devices.

Scalability an important design principle for SHIP, as a wide variety of devices should be addressed within the SHIP-protocol. Simple devices with limited connection capabilities (worst case assumption: only one simultaneous connection) or no or simple user interfaces (e.g. push button) shall be enabled, as well as gateway or cloud solutions with enhanced capabilities.

To provide a future-proof solution, this specification also defines different mechanisms for downward compatible extensibility.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

2 Normative References

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IETF RFC 768: 1981, User Datagram Protocol

IETF RFC 793: 1981, Transmission Control Protocol

IETF RFC 1035: 1987, Domain Names

IETF RFC 2104: 1997, HMAC, Keyed-Hashing for Message Authentication

IETF RFC 2119: 1997, Key words for use in RFCs to indicate requirement levels

IETF RFC 3279: 2002, Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure certificate and Certificate Revocation List (CRL) Profile

IETF RFC 3280: 2002, Internet X.509 Public Key Infrastructure Certificate Revocation List (CRL) Profile

IETF RFC 4055: 2005, The Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

IETC RFC 4627: 2006, The application/JSON Media Type for JavaScript Object Notation (JSON)

IETF RFC 5077: 2008, Transport Layer Security (TLS) Session Resumption without Server-Side State

IETF RFC 5234: 2008, Augmented BNF for Syntax Specifications: ABNF

IETF RFC 5246: 2008, The Transport Layer Security (TLS) Protocol Version 1.2

IETF RFC 5280: 2008, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

IETF RFC 5289: 2008, TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)

IETF RFC 5480: 2009, Elliptic Curve Cryptography Subject Public Key Information

IETF RFC 5758: 2010, Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA

IETF RFC 6066: 2011, Transport Layer Security (TLS) Extensions

IETF RFC 6090: 2011, Fundamental Elliptic Curve Cryptography Algorithms

IETF RFC 6298: 2011, Computing TCP's Retransmission Timer

IETF RFC 6455: 2011, The WebSocket Protocol

IETF RFC 6762: 2013, Multicast DNS

IETF RFC 6763: 2013, DNS-Based Service Discovery

IETF RFC 7320: 2014, URI Design and Ownership

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

213 ISO/IEC 18004:2015: Information technology — Automatic identification and data capture
214 techniques — QR Code bar code symbology specification

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

3 Terms and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in IETF RFC 2119.

3.1 CA

Certificate Authority or Certification Authority. A CA can provide a digital signature for certificates. Other SHIP nodes can check this digital signature with the certificate from the CA itself, the "CA-certificate".

3.2 Commissioning Tool

In the scope of SHIP, a commissioning tool may be used to establish the trust between different devices in the smart home installation, e.g. distribute trustworthy credentials from some SHIP nodes to other SHIP nodes. E.g. a smart phone, a web server or a dedicated device can embody the role of a commissioning tool. So far, the SHIP specification does not specify a commissioning tool. An interoperable protocol for commissioning can be used on the layer above SHIP. A manufacturer may also use their own solutions.

3.3 DNS

Domain Name System, see IETF RFC 1035.

3.4 DNS host name

Fully qualified domain name used within DNS as host name to get the IP address of the corresponding internet host.

3.5 DNS-SD

Domain Name System – Service discovery, see IETF RFC 6763.

3.6 EUI

Extended Unique Identifier, see **http://standards.ieee.org/develop/regauth/tut/eui64.pdf** .

3.7 Factory Default

A factory default SHALL allow the user to reset the SHIP node to the as-new condition. This means that all data that has been provided and stored by the SHIP node during its operation time SHALL be deleted.

3.8 IANA

Internet Assigned Numbers Authority.

3.9 IP

Internet Protocol.

3.10 LAN

Local Area Network.

3.11 MAC

Media Access Control.

3.12 mDNS, multicast DNS host name

Fully qualified domain name used within mDNS as host name to get the IP address of the corresponding local SHIP node.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

3.13 Numerical representation

0x introduces the hexadecimal representation of an unsigned value. For example, 0xab represents a decimal value of 171.

3.14 PIN

Personal Identification Number. This specification makes use of a PIN as secret for SHIP specific verification procedures.

3.15 PKI

Public Key Infrastructure.

3.16 Push Button

The term Push Button is used to describe a simple trigger mechanism. A Push Button event does not necessarily mean that a real physical button has to be used to trigger this event. A Push Button event may also be generated by other means, e.g. via a smart phone application or a web-interface (secure connection to SHIP node required). A Push Button shall provide a simple mechanism for a user to bring the device to a certain state or start a certain process.

3.17 QR Code

The term "QR Code" is a registered trademark of DENSO WAVE INCORPORATED. "QR Code" is the short form for “Quick Response Code” and used for efficient encoding of data into a small graphic.

Among others, the international standard ISO/IEC 18004:2015 specifies the encoding of QR code symbols.

3.18 RFC

Request for comments.

3.19 SHIP

Abbreviation of "Smart Home IP". This term is used throughout this document to refer to the described communication protocol.

3.20 SHIP ID

Each SHIP node has a globally unique SHIP ID. The SHIP ID is used to uniquely identify a SHIP node, e.g. in its service discovery. This ID is present in the mDNS/DNS-SD local service discovery; see chapter 7.

3.21 SHIP Client

The SHIP client role shall be assigned to the SHIP node that also embodies the TCP client role for a specific peer-to-peer connection.

3.22 SHIP Node

A SHIP node is a logical device which communicates via the described SHIP protocol and can be integrated into a web server or physical device.

Note: One physical device may have more than one logical SHIP node. In this case, each SHIP node MUST use distinct ports (e.g. a physical device provides 3 open ports with 3 different SHIP services).

3.23 SHIP Server

The SHIP server role shall be assigned to the SHIP node that also embodies the TCP server role for a specific peer-to-peer connection.

3.24 SKI

Each SHIP node has a specific public key. The Subject Key Identifier (SKI) is derived from this public key and is used as a cryptographically backed identification and authentication criterion.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

3.25 SPINE

Smart Premises Interoperable Neutral message Exchange: Technical Specification of EEBus Initiative e.V.

3.26 Trusted SHIP Node

A trusted SHIP node is a term which is only applicable from a specific SHIP node point of view.

If SHIP node A has a communication partner and a trusted relationship to SHIP node B, SHIP node B is called a trusted SHIP node from SHIP node A’s point of view. A trusted relationship can be established in different ways, as described in chapter 12.2.2.

3.27 UCS

Universal Character Set.

3.28 UTF

UCS Transformation Format. A computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.

3.29 WAN

Wide Area Network.

3.30 Web server based SHIP node

A SHIP node that is hosted by a web server.

3.31 WiFi

IP networks based on the IEEE802.11 set of standards, used for wireless IP communication.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

4 Architecture Overview

Smart Home IP (SHIP) describes an IP-based approach for interoperable connectivity of smart home appliances, which covers local SHIP nodes in the smart home as well as web server based SHIP nodes and remote SHIP nodes.

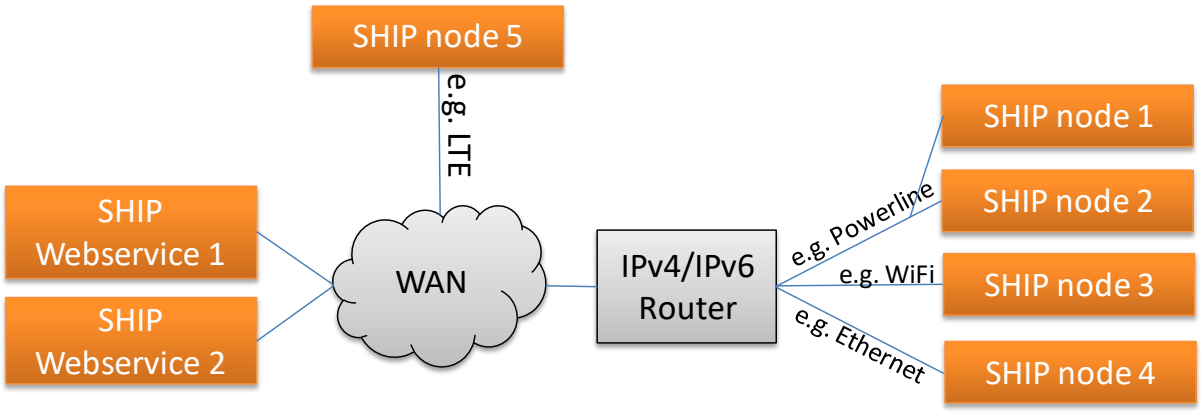


Figure 1: Physical Connections in the Overall System

SHIP nodes MAY base on different physical layer approaches, e.g. WiFi or powerline technologies. An IP router can be used to connect different physical networks and provide access to the internet, but this is out of the scope of the SHIP specification.

On the IP Layer, IPv4 as well as IPv6 are permitted. IP addresses can be preconfigured, assigned via DNS-server, with SLAAC, or by any other appropriate means.

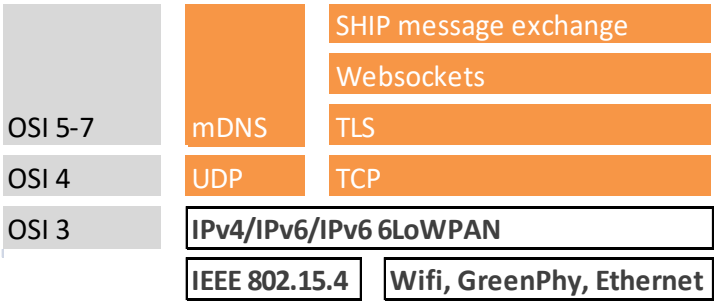


Figure 2: SHIP Stack Overview

A SHIP node SHALL support mDNS/DNS-SD for local device/service discovery. The SHIP protocol is based on TCP, TLS and WebSocket.

Note: Computationally limited SHIP nodes MAY only support a limited number of connections. For further information see chapter 8.1.

Note: A SHIP node MUST always provide a server port. Only a SHIP node that supports only one simultaneous active connection MAY close the server port in order to establish a client connection.

In SHIP, it is not important which SHIP node takes over the server or client role. If two SHIP nodes try to connect to each other virtually simultaneously, double connections are prevented by the mechanism described in chapter 12.2.2.

SHIP specific messages conveyed over the established WebSocket connection shall be encoded using JSON, like described in chapter 11.

As the SHIP specification is defined by members of the EEBus Initiative e.V., it can be used to transport EEBus-specific payload and provide an EEBus IP-Backbone, but also other protocols

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

346 can be used above SHIP. To provide a clean solution, SHIP is defined without dependencies to
347 the EEBus data model.

348 ## 4.1 General Considerations On Closing Communication Channels

349 In general, different manufacturers will favour different strategies on opening and closing
350 communication channels. Some manufacturers will choose to leave communication channels
351 open as long as possible, while others will choose to close communication channels as soon
352 as possible. In practice, limitations of the number (or duration) of supported parallel connections
353 might force connections to be closed at least temporarily. Thus, a termination process is defined
354 in order to distinguish sudden interrupts or failures from (typically) temporary disconnections.
355 Details are explained in section 13.4.7.

356 ## 4.2 SHIP Node Parameters

357 Throughout this specification, different parameters are defined to provide an exact description
358 of the SHIP behaviour where needed. The different parameters are summarized in the following
359 list. Please go to the corresponding chapter for a detailed description.

Description	Chapter	Default value range	Default value recommendation
Initial TCP retransmission count	8.3		2
Initial TCP retransmission timeout	8.4		1s
Maximum TCP retransmission timeout	8.4		120s
MTU (Maximum Transmission Unit)	8	1500 bytes	
Maximum fragment length	9.2	512 bytes	
Connection Keepalive "ping" min interval	10.4	50s	
Connection Keepalive "pong" timeout	10.4	10s	
SKI length	12.2	20 bytes (40 digit hexadecimal string)	
PIN length	12.5	8-16 digit hexadecimal string	
Maximum "auto_accept" time window	12.3.1.1	≤120s	60s
“User trust level” necessary for general SHIP communication	12.3.2	≥8	
“User trust level” or “second factor trust level” necessary for commissioning	12.3.2	≥32	
CmiTimeout	13.4	10-30s	30s
Wait-For-Ready-Timer initial	13.4.4.1	60-240s	120s
Wait-For-Ready-Timer prolongation	13.4.4.1	60-240s	120s
PIN entry penalty after the 3rd invalid attempt	13.4.4.3	10-15s	15s
PIN entry penalty after the 6th invalid attempt	13.4.4.3	60-90s	90s

360 *Table 1: SHIP Parameters Default Values*

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

5 Registration

The registration of a SHIP node can be triggered by different mechanisms, e.g., a push button (“auto accept”, as described in chapter 12.3.1.1), a commissioning tool (“commissioning” as described in chapter 12.3.1.3) or if an SKI is entered or verified by the user (“user input” or “user verify” as described in chapters 12.3.1.4 and 12.3.1.2).

Note: Details on how to find appropriate SHIP nodes are explained in chapter 7 where you can also find details on "register" flags and "SKI" values.

Registration with secure verification:

Sections 12.3.1.2 (user verification), 12.3.1.3 (commissioning), and 12.3.1.4 (user input) describe verification modes with the possibility to trust the SKI value of another SHIP node. For these modes, a SHIP node SHALL search for SHIP nodes with trusted SKI values and connect to them to establish the registration. If the registration with a SHIP node does not complete successfully (see section 5.1 for successful registration), the SHIP node SHALL cyclically retry to connect to the SHIP node with the corresponding SKI.

If the other local SHIP node aborts the SME “hello”, as described in chapter 13.4.4.1, a SHIP node SHOULD NOT retry to connect to this SHIP node again as long as the register flag of the other SHIP node is set to “false”. If the register flag of the other SHIP node is set to “true”, a SHIP node that already has the trusted SKI from the other SHIP node SHALL retry to connect to the other SHIP node again.

Registration with auto accept:

If the “auto accept” mode is active, as described in chapter 12.3.1.1, a SHIP node SHALL set its own register flag for the service discovery to true. Additionally, the SHIP node SHALL start a service discovery for other SHIP nodes that have set the register flag to true.

Note: If both SHIP nodes use the “auto accept” mode to connect to each other, this is called mutual “auto accept”. In the case of mutual “auto accept”, no side verifies any SKI and therefore a so-called “man-in-the-middle” attack cannot be excluded. In the “man-in-the-middle” case, a third device (the "man in the middle", a potentially harmful device) is able to secretly read and even manipulate the communication between two SHIP nodes. Therefore, it is STRONGLY RECOMMENDED to support at least one of the other verification modes to avoid mutual “auto accept” and potential “man-in-the-middle” attacks!

If a SHIP node discovers more than one other SHIP node with a “register” flag set to true, it SHALL pick one SHIP node by any means appropriate (e.g., it could interpret additional information contained in the service discovery).

If the “auto accept” mode is inactive, a SHIP node SHALL set its own “register” flag for the service discovery to false and SHALL stop searching for other SHIP nodes that also have set the “register” flag to “true”.

Note: Only “auto accept” affects the register flag. The other verification modes (“user verify”, “user input” and “commissioning”) SHALL have no effect on the register flag.

The registration between a SHIP node and a web server based SHIP node is established in the following order:

1. Retrieve IP address and port number from DNS (only if URL/ DNS host name is used; if IP address is used, this step can be skipped)

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

2. Connect to IP address and port number
3. Verify public key of the web server-based SHIP node, as described in chapter 12.3
4. SHIP message exchange (SME) Connection Mode Initialization
5. SHIP message exchange (SME) Connection Data Preparation

The registration with another local SHIP node is established in the following order:

1. If “auto accept” is active, set register flag in service discovery to “true”; otherwise, it must be set on “false”.
2. If “auto accept” is used and the other SHIP node has set the register flag in service discovery to true, it is strongly recommended to switch to another verification mode to prevent mutual “auto accept”. If “user verify”, “user input” or “commissioning” is used, search for SHIP nodes with corresponding SKI values in the service discovery.
3. Connect to IP address and port number retrieved via service discovery or accept incoming connection.
4. Verify public key of the communication partner as described in chapter 12.3
5. SHIP message exchange (SME) Connection Mode Initialization
6. SHIP message exchange (SME) Connection Data Preparation

With the SME “hello” message, a SHIP node can confirm the trustworthiness of the communication partner as described in chapter 13.4.4.1. If a SHIP node trusts the communication partner, the SHIP node SHALL store the credentials of the communication partner.

Note: If a SHIP node only supports one simultaneous active connection, it MAY close the server port during the registration phase in order to be able to establish a client connection. In this case, the constrained SHIP node SHALL wait a time of X milliseconds before it closes the server port and tries to establish a connection to another SHIP node that has the register flag in service discovery set to “true”. X SHOULD be a random value between 0-30000 milliseconds.

5.1 Successful Registration

When both sides have confirmed trustworthiness of each other with an SME “hello” message, the registration is successfully completed. Every new connection between these two devices MUST now be viewed as reconnection and not a registration until one of both SHIP nodes purposely aborts the SME “hello” handshake.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

6 Reconnection

If two SHIP nodes have successfully established a connection before, both nodes can reconnect to each other at any time. It does not matter if the register flag is “true” or “false” during reconnection – the register flag is only important for the registration process when connecting to new SHIP nodes (see chapter 5).

If the public key still matches the previously (during registration) provided, verified and trusted public key, the SHIP node SHALL be accepted again without any delay.

A reconnection between a SHIP node and a web server-based SHIP node is established in the following order:

1. Retrieve IP address and port number from DNS (only if URL / DNS host name is used; if IP address is used, this step can be skipped)
2. Connect to IP address and port number
3. Check if the public key of the communication partner is still the same as during registration
4. SHIP message exchange (SME)

A reconnection with a local SHIP node is typically established in the following order:

1. Connect to IP address and port number retrieved via service discovery or accept incoming connection
2. Check if the public key of the communication partner is trusted and still the same as during registration
3. SHIP message exchange (SME)

In the reconnection scenario, both SHIP nodes should already trust each other, so no user interaction is necessary. With the SME “hello” message, a SHIP node can directly confirm the trustworthiness of the communication partner as described in chapter 13.4.4.1 and continue with SME protocol handshake, optional PIN verification, and data exchange as described in chapter 13.4.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

7 Discovery

A discovery mechanism is used to find available SHIP nodes and their services in the local network without knowing their multicast DNS host names or IP addresses. For this purpose, mDNS/DNS-SD SHALL be used.

DNS-SD records SHOULD have a TTL of 2 minutes.

mDNS/DNS-SD provides methods for local service discovery, resource discovery, and multicast DNS host name to IP address resolution. Detailed information on mDNS can be found in RFC 6762; information on DNS-SD can be found in reference RFC 6763.

A SHIP node that uses mDNS SHALL offer a service named “`ship`”.

7.1 Service Instance

The SHIP node SHALL assign an <Instance> label of up to 63 bytes in UTF-8 format for each DNS SRV/TXT record pair that it advertises. In accordance with RFC 6763 and in order to avoid name conflicts, this label SHALL use user-friendly and meaningful names, for example the device type, brand and model. Using a hypothetical company "ExampleCompany", an example <Instance> of a product with a SHIP node could be "`Dishwasher ExampleCompany EEB01M3EU`".

Should a name conflict still occur, a node SHALL assign itself a new name until the conflicts are resolved. A conflict SHOULD be resolved by appending a decimal integer in parentheses to the <Instance> (for example, "`Name(2)`" for the first conflict, "`Name(3)`" for the second conflict, etc.).

7.2 Service Name

The service name used with DNS-SD SHALL be “`ship`”.

The <Service> portion of a service instance name consists of the service name preceded by an underscore '_' and followed by a period ‘.’ plus a second DNS label specified by SHIP as "_tcp".

Thus, a valid service instance name example would be:

"`Dishwasher ExampleCompany EEB01M3EU._ship._tcp.local.`"

where “`Dishwasher ExampleCompany EEB01M3EU`” is the <Instance> portion (described in previous section), “`ship`” is the service name, “`tcp`” is the transport protocol, and “`local`” is the <Domain> portion.

7.3 Multicast DNS Name

A local SHIP node SHALL assign a unique multicast DNS host name of up to 63 bytes. In order to avoid name conflicts, names SHOULD use the unique ID as specified in the TXT record.

Thus, a complete multicast DNS host name example would be:

"`EXAMPLEBRAND-EEB01M3EU-001122334455.local.`"

7.3.1 Default Records

DNS-SD defines several records by default. This information MUST NOT be included in other records.

The A record includes the IPv4 address and the AAAA record includes the IPv6 address of the node. The SRV record SHALL include the service name, multicast DNS host name and port.

Note: A SHIP node MAY freely choose its port for the SHIP TCP server, but MUST state it correctly in the SRV record.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

7.3.2 TXT Record

This sub-section specifies the format of the TXT record to be used in conjunction with DNS-SD. A SHIP node SHALL use a single TXT record format. The TXT record SHALL NOT exceed 400 bytes in length. The following table contains additional service parameters that SHALL be included in the TXT record.

Key	Value	Example	Runtime Behavior	Required
txtvers	Version number	txtvers=1	Static	Mandatory
id	Identifier	id=EXAMPLEBRAND-EEB01M3EU-001122334455	Static	Mandatory
path	String with wss path	path=/ship/	Static	Mandatory
ski	40 byte hexadecimal digits representing the 160 bit SKI value	ski=1234AAAAFFFF1111 CCCC3333EEEEDDDD 99992222	Static	Mandatory
register	Boolean	register=true	Static	Mandatory

Table 2: Mandatory Parameters in the TXT Record

The TXT record can include other optional key-values as long as the TXT record does not exceed 400 bytes in length. The following optional keys are defined by this specification:

Key	Value	Example	Runtime Behaviour	Required
brand	String with brand	brand=ExampleBrand	Static	Optional
type	String with device type	type=Dishwasher	Static	Optional
model	String with model	model=EEB01M3EU	Static	Optional

Table 3: Optional Parameters in the TXT Record

txtvers SHALL be the first key in the TXT record. For this specification, the value of the `txtvers` key SHALL be 1. If it is found in a response to be other than 1, the TXT record SHALL be ignored. The `txtvers` key SHALL be present with a non-empty value. Clients SHALL silently discard TXT records with `txtvers` keys that are not present or have a different value than 1.

Unknown key pairs in a response SHALL be ignored.

The `id`, `ski`, `brand`, `type` and `model` values SHALL be in UTF-8 format.

The value of the `id` key contains a globally unique ID of the SHIP node and has a maximum length of 63 bytes. The first part of the unique ID SHOULD be an abbreviation of the manufacturer name. Behind the abbreviation, the manufacturer defines a unique identifier. The id value (SHIP ID) shall be unique. Note: The presence of two SHIP nodes with identical id values in a local network is not considered a regular setup within this specification as it may disrupt regular SHIP communications.

The maximum length of the `brand`, `type` and `model` values will be 32 byte of UTF-8 data each.

The maximum length of the path value will be 32 bytes of UTF-8 data. The minimum length is 1, where the path key contains the value “/”.

The `ski` key allows other SHIP nodes to directly identify a SHIP node by its SKI. This is very helpful for other SHIP nodes that were provided with one or more trustworthy SKI values from other SHIP nodes via “commission tool”, “user verification” or “user input”. Otherwise, trial-and-error TLS handshakes with all nodes would be necessary to find the nodes with the fitting public key / SKI. Also, SHIP nodes that support “user verify” do not need to gather SKIs from local SHIP nodes over a TLS handshake, but can gather the SKIs simply via service discovery.

An SKI with the value 0x1234AAAAFFFF1111CCCC3333EEEEDDDD99992222

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

538 SHALL be encoded as ski=1234AAAAFFFF1111CCCC3333EEEEDDDD99992222.

539 The `register` key is used to indicate whether auto accept is active in the SHIP node.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

8 TCP

TCP SHALL be used for communication. A communication over UDP, apart from mDNS for service discovery, is not specified at the moment, but might be added later for multicast and group communication scenarios.

The MTU size SHALL NOT exceed 1500 bytes.

For a local server, the port SHALL be set according to the DNS-SD SRV record, as described in chapter 7.3.1.

8.1 Limited Connection Capabilities

A SHIP node MUST support a minimum of “1” simultaneously active connection.

If a local SHIP node is limited to “1” simultaneously active connection, it SHALL provide a listening TCP server and SHALL only close the TCP server port if it wants to establish a connection to another SHIP node as a TCP client. After using a TCP client connection, it SHALL close the TCP client connection as soon as possible and start listening on the TCP server port again.

If a SHIP node supports more than “1” simultaneously active connection, it SHALL always reserve one connection for the TCP server port. This means that when the SHIP node is limited to “x” simultaneously active connections, it SHALL only use a maximum of “x-1” connections for TCP client connections.

If a SHIP node supports more than “1” simultaneously active connection, it SHALL always reserve one connection for TCP client connections. This means that when the SHIP node is limited to “x” simultaneously active connection, it SHALL only use a maximum of “x-1” connections for TCP server connections.

In general, a SHIP node MAY close the TCP server port when it has reached its connection limit. In this case, the SHIP node SHALL reopen the TCP server port as soon as possible. If a SHIP node has not reached its connection limit, it SHALL always have an open TCP server port.

8.2 Online Detection

Before a local SHIP node can try to establish a connection over TCP to another local SHIP node, the other SHIP node SHOULD be detected as “online”.

If the TTL of the mDNS service announcement of a local SHIP node is not valid, this SHIP node SHOULD be interpreted as “offline”. If the mDNS service announcement of the corresponding local SHIP node is updated and the TTL is valid, the SHIP node SHALL be interpreted as “online” again.

In addition, a local SHIP node MAY send ICMP echo requests (Pings) to another local SHIP node to check whether the other side is “online” or “offline”.

Note: In certain environments or devices, ICMP echo requests/replies MAY be blocked. If a local SHIP node is unable to receive an ICMP echo reply, but mDNS service announcements are received from the other local SHIP node, the SHIP node SHALL consider the other SHIP node as subject to ICMP blocking. In this case, the local SHIP node SHALL NOT use the ICMP echo requests as an indicator for the “offline” state and SHOULD only rely on the TTL of the mDNS service announcement as “offline” indicator.

Note: The “offline” detection is especially important for local SHIP nodes with limited connection capabilities. Trying to reach a SHIP node B that is “offline” can cost SHIP node A 10 seconds of connection time, see 8.3. This means that other SHIP nodes may not be able to reach this SHIP node A for about 10 seconds while it is trying to establish a connection with SHIP node B that is offline.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

8.3 TCP Connection Establishment

As described in RFC 793, an initial SYN packet is sent from the client to the server to initiate a TCP connection. When a server accepts the incoming connection, it responds with an acknowledgment of the SYN packet (SYN ACK). When a SHIP server receives a SYN packet for a closed port, it SHALL respond with a reset (RST) packet as described in chapter 3.4 / page 36 of RFC 793. Furthermore, the RST packet SHOULD not be blocked or filtered out, e.g. by a firewall, on the SHIP node.

The usage of the RST packet allows SHIP clients to very quickly detect whether the server port of the other SHIP node is closed. In that case, the connecting SHIP node can immediately abort the connection attempt. This also reduces the usage time of TCP connections, which can be of high importance for constrained devices, as TCP connections may be a limited resource, as described in section 8.1.

As the SYN packet as well as the RST packet may get lost, the initial SYN packet SHOULD be retransmitted twice if no response (e.g. an ACK or RST) is received. If the recommended timeouts from section 8.4 are used, this results in a maximum connection establishment duration of:

1 + 3 + 6 = 10 seconds

8.4 Retransmission Timeout

A SHIP node SHOULD maintain a retransmission timer as defined by RFC 6298. For that, round-trip times (RTT) of transmitted packets are measured. The RTT means the time from sending a packet to receiving the corresponding acknowledge (ACK). Retransmitted packets MUST NOT be used for measuring, because in that case an ACK cannot be uniquely assigned to a sent packet anymore.

From the measured values, a retransmission timeout (RTO) is calculated, which is used for subsequent transmissions of packets. RFC 6298 recommends a minimum RTO value of 1 second. Nonetheless, it also points out that this is a conservative value and it will most likely make sense to reduce it in the future. Thus, a SHIP node MAY reduce the minimum RTO value.

For the initial SYN packet, the value of the RTO SHOULD be initialized with 1 second as also recommended by RFC 6298. RFC 6298 appendix explains why this has been reduced from the historical value of 3 seconds. A SHIP node MAY choose to increase this value for lossy networks.

After the first retransmission of a packet, RFC 6298 demands to set the RTO to a minimum of the historical value of 3 seconds. Then, an exponential back-off is applied, which doubles the RTO with every retransmission. A SHIP node SHOULD apply a maximum RTO value of 120 seconds.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

9 TLS

TLS 1.2 is MANDATORY. Apart from the maximum fragment length, see 9.2, TLS MUST be used as specified in RFC 5246.

SHIP nodes MUST use mutual authentication during the TLS 1.2 Handshake, hence the public key / certificate MUST be verified on the client and server side, as described in chapter 12.2.2.

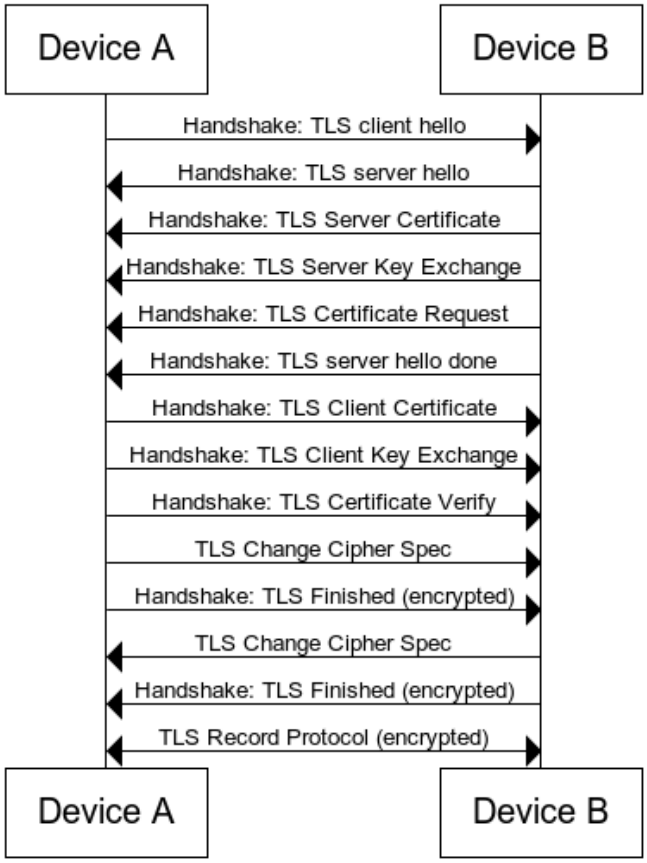


Figure 3: Full TLS 1.2 Handshake with mutual authentication

9.1 Cipher Suites

The TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 cipher suite, as specified in RFC 5289, MUST be supported.

OPTIONAL cipher suites are:

- TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

Hence, ECDSA is used for authentication and ECDHE for key exchange with perfect forward secrecy. The algorithms are based on ECC because the computational costs are lower than for e.g. RSA with similar security.

As ECC curve, secp256r1 curve MUST be used; other curve sets are not allowed at this time. Secp256r1 is widely supported by different solutions and libraries; other curves might be added later.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

9.2 Maximum Fragment Length

Maximum Fragment Length Negotiation Extension, as specified in RFC 6066, SHOULD be supported. If used, Maximum Fragment Length Negotiation Extension SHALL only support a length of 1024 bytes. This keeps the required buffer size for embedded devices low.

Some TLS implementations currently do not support Maximum Fragment Length Negotiation Extension. Therefore, a SHIP node SHALL ensure that the fragment length (TLSPlaintext.length) of outgoing packets does not exceed 1024 bytes, even if Fragment Length Negotiation Extension is not supported.

9.3 TLS Compression

TLS Compression MUST NOT be used.

9.4 Server Name Indication

As specified by WebSocket in RFC 6455, the server name indication (SNI) extension MUST be supported However, local SHIP nodes SHALL ignore the information provided by the SNI extension. Web server-based SHIP nodes MAY evaluate the SNI extension if they have a fixed DNS host name.

For local connections, the server name SHALL be equal to the mDNS host name of the local server. For web server connections, the server name SHALL be the DNS hostname of the webserver.

Note: As described in RFC6066, the server name for SNI is represented as a byte string using ASCII encoding without a trailing dot. This means that even if the server name in mDNS might have a trailing dot, this trailing dot should not be used for SNI. However, some web browsers seem to use the trailing dot for SNI in the client hello due to an incorrect implementation. Therefore, a SHIP server implementation SHOULD ignore the last trailing dot if it is mistakenly inserted by the client.

9.5 Renegotiation

As the usage of TLS Renegotiation is not defined within SHIP, a SHIP node SHALL NOT support TLS renegotiation and refuse TLS renegotiation requests in general.

9.6 Session Resumption

A full TLS 1.2 handshake introduces large computational costs and additional round trips. From a user perspective, these computational costs can lead to delays in reaction time > 1 second for constrained devices. To allow fast reconnections over TLS without the need for a full TLS handshake, session resumption SHOULD be supported. This means that the session state holding the master secret and a session id SHOULD be stored and reused during reconnections.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

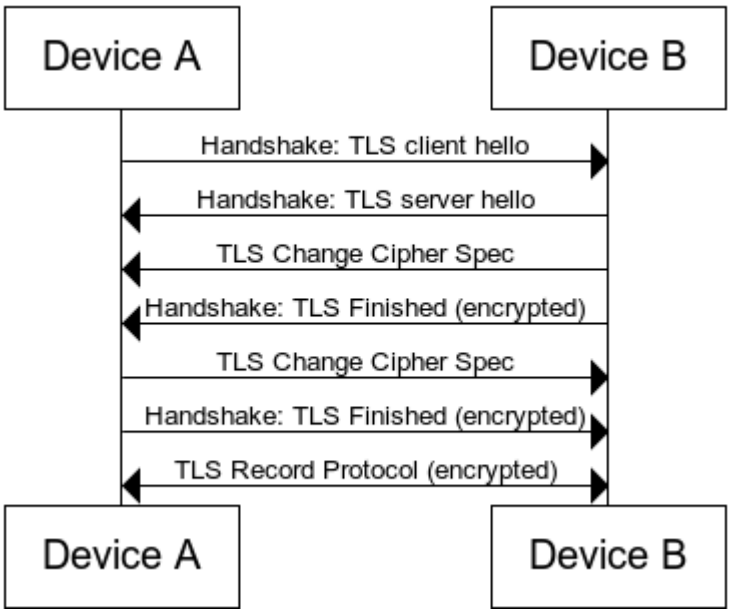


Figure 4: Quick TLS Handshake with Session Resumption

A SHIP node MAY discard a session state, e.g. if the connection has low requirements regarding the latency and reaction time, if the connection was not used for a certain amount of time, or if there is no more space for storage left and a new connection is established that is rated with a higher priority.

Note that discarding the session state always forces a full TLS handshake when the connection is re-established.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

10 WebSocket

On top of TCP and TLS, WebSocket MUST be used. One of the goals achieved by using WebSocket is the ability of the protocol to maintain connections through local network gateways such as Network Address Translation (NAT) devices or firewalls.

Note that a number of “draft versions” of the WebSocket standard exist that are incompatible with the current standard. Therefore, this specification requires strict compliance with RFC 6455.

10.1 TLS Dependencies

A SHIP node client MUST use the Server Name Indication extension in the TLS handshake (RFC 6066). This is especially important for large-scale service providers such as cloud installations to be able to provide services for various server names. Please see section 9.4 for more details.

10.2 Opening Handshake

This section refers to sections 1.3 (non-normative) and 4 in RFC 6455.

Valid WebSocket Request-URIs for use with SHIP MUST follow the wss scheme (i.e., a valid SHIP URI will always start with “wss://”) if TLS is used. This specification does not make any assumptions on the host, port and resource name properties of the request. A SHIP node will learn these properties via the SHIP discovery process as described in chapter 7. If a SHIP node decides to connect to another SHIP node, it SHALL present these properties in the exact same fashion as previously discovered.

The origin property MAY not be present in the request.

Each WebSocket request conforming to this specification MUST include the Sec-WebSocket-Version header with a fixed value of 13. Earlier WebSocket draft standard versions are not allowed. Additionally, the value “`ship`” MUST be included in the Sec-WebSocket-Protocol header.

With this procedure, a SHIP node SHALL detect whether it is talking to another SHIP node at the earliest stage possible, preventing the overhead of useless communication with other SHIP nodes that implement the WebSocket protocol, but without the SHIP payload. The request MUST NOT contain any other subprotocol names.

The current version of this document does not specify any WebSocket extensions. Therefore, the request MUST NOT contain a Sec-WebSocket-Extensions header.

10.3 Data Framing

This section refers to section 5 in RFC 6455.

SHIP protocol messages are at least partially binary. Therefore, all data frames (i.e. non-control frames) used with this specification MUST be of type 0x2 (binary frames). A SHIP node that receives a data frame with another type (0x1) MUST terminate the connection with status code 1003 (unacceptable data). A SHIP node that receives a data frame with type (0x3–0x7, 0xB-0xF) MUST terminate the connection with status code 1002 (protocol error).

Since this specification does not allow any extensions, the reserved bits of the base framing protocol MUST be set accordingly (to value 0) and the reserved opcodes in the framing header MUST NOT be used.

For clarity, please note that while RFC 6455 requires clients to support fragmentation of messages and to support handling control frames in the middle of a fragmented message, it explicitly forbids interleaving of fragments belonging to different messages. The absence of SHIP protocol message interleaving (“multiplexing”) is not considered a relevant issue at the moment since SHIP protocol messages are expected to be relatively small (i.e., their transmission on a typical IP layer will only take a few milliseconds).

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

10.4 Connection Keepalive

A SHIP WebSocket connection SHALL be left established whenever local resource usage on the SHIP node permits this behaviour to reduce delay and reaction times of SHIP nodes.

In wide-area networking scenarios (e.g. for Cloud services), connections can typically only be established from a local SHIP node towards a remote one and not vice versa (i.e., only from the local device towards the Cloud, not vice versa because of a local firewall or NAT gateway). In this case, keeping up the connection is vital to be able to receive messages from the Cloud at any given time.

Furthermore, large-scale deployments might need to deploy fail-safe algorithms to detect server failures and re-route traffic to other nodes. A server failure may be detected quickly when using keep-alive connections, and re-routing will then usually occur before the connection is really needed for the next payload message, improving overall protocol resilience and user experience.

In addition to keeping connections alive whenever possible, a SHIP node SHALL make use of the ping and pong control frames to ensure that the underlying transport is really operational.

A SHIP node MUST NOT send ping messages at intervals smaller than 50 seconds on a single connection. The typical timeout waiting period for a pong message after sending a ping message SHALL be 10 seconds.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

11 Message Representation Using JSON Text Format

11.1 Introduction

Many SHIP messages are sent using a JSON-UTF8 representation as a basis. However, the SHIP protocol is prepared to allow other formats, such as JSON-UTF16 or ASN.1.

For different reasons, which are beyond the scope of this document, some parts of the messages are defined using XSD (XML Schema Definition). This language permits the description of XML structures and content for specific purposes. Several tools can be found that permit creation of XMLs or even so-called “data binders” from XSDs.

JSON, or Java Script Object Notation, like XML, is an interchange format to describe data objects. It has beendescribed in RFC 4627 since 2006. Because of its small set of formatting rules, it is easy to implement. Code for parsing and generating JSON is available in most programming languages.

In order to benefit from the advantages of XSD and JSON, this chapter describes which JSON types must be used and how to generate JSON text representations from the XSD. In general, it is rather difficult to map every feature of an XML to a corresponding JSON representation. However, there are some ways to retain the semantics of most XSD elements.

11.2 Definitions

JSON consists of six basic types.

1. Number
2. String (double-quoted)
3. Boolean
4. Array (ordered sequence)
5. Object (unordered collection)
6. Null

The data representation consists of key:value pairs and is built on two structures: An unordered collection surrounded by left and right curly brackets or an ordered sequence surrounded by square brackets. The members inside the structures are separated by commas.

11.3 Examples For Each Type

1. Number
   {”age” : 12, ”height” : 1.73}

2. String (double-quoted)
   {”name” : ”JSON in WebSocket”}

3. Boolean
   {”checked” : true}

4. Array (ordered sequence)
   {”item” : [”one”, ”two”, ”three”]}

5. Object (unordered collection)
   {”Name” : ”Crockford”, ”First name” : ”Douglas”}

6. Null
   {”nillable” : null}

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

11.4 XML to JSON Transformation

11.4.1 Scope

The transformation rules in this chapter apply to all data structures in section 13.4 that are explicitly defined using XSD. These structures are called “explicit SHIP structures”.

The SHIP Message Exchange permits conveying payload of an external (i.e. non-SHIP) specification within the element “data.payload” (see section 13.4.5.2). The corresponding specification is announced using the element “protocolId”. An external specification MAY apply the specific conversion rules of this chapter as well. For each protocolId, it is permitted to define deviating rules for the content of “payload”.

11.4.2 XSD Types

The mapping of XSD types is described in Table 4.

XSD types	**JSON types**
xs:boolean	Boolean
xs:double, xs:byte, xs:unsignedByte, xs:short, xs:unsignedShort, xs:integer, xs:unsignedInt, xs:nonNegativeInteger, xs:unsignedLong	Number
xs:dateTime, xs:duration, xs:time	String
xs:language, xs:string	String
xs:hexBinary	String
xs:anyType	Results in corresponding types
xs:simpleType	See next chapter 11.4.4.
xs:complexType	See next chapter 11.4.5.

Table 4: Mapping from the XSD Types to JSON Types.

11.4.3 Element Occurrences

Elements with a specified type can contain the attributes “minOccurs” and “maxOccurs”. These attributes specify how often the field can or, respectively, must be added. E.g. “minOccurs=0” means the field is optional and may be omitted. If the “minOccurs” attribute is omitted for an element, it is implicitly set to 1, which means the field is mandatory.

If “maxOccurs” is set to a value greater than 1, the element is transformed to a JSON array, which contains items of the corresponding type. In that case, “maxOccurs” defines the maximum length of the array, where “unbounded” means there is no upper limit. If the “maxOccurs” attribute is omitted, it is implicitly set to 1, which means no JSON array is generated for the element, but of course the corresponding JSON type.

If “minOccurs” and “maxOccurs” are both set to 0, the element is ignored.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

11.4.4 Simple Types

Simple types are specified with the <xs:simpleType> item and always refer to simple types like <xs:integer> or <xs:string>. Simple types can specify restrictions on the value or can define a list or a union of one or more simple types.

XSD item in xs:simpleType	JSON types
xs:restriction	Type corresponding to the base type. Restricts the possible values.
xs:list	Array
xs:union	String

Table 5: Transformation of a simple type.

Restrictions contain XSD items like <xs:minLength>, <xs:maxLength>, <xs:enumeration>, etc. These items limit the possible values of the type and apply to JSON and XML in the same way.

11.4.5 Complex Types

Complex types consist of a combination of sub-elements (“particles” in the XML specification), which can be arranged in different ways. These combinations are called compositors, which are: sequence, choice, and all. Some of them can also be nested. Depending on their usage, these compositors result in different JSON representations:

Used in: / Compositor:	xs:complexType	xs:sequence	xs:choice	xs:all
xs:sequence	Array	+	+	Not allowed
xs:choice	Array	+	+	Not allowed
xs:all	Object	Not allowed	Not allowed	+
None	Array	-	-	-

Table 6 Mapping from the XSD compositors to JSON Types.

“+“ means that the sub-elements are only integrated in the superset without creating a new hierarchy level. “Not allowed” means that XML Schema 1.0 prohibits this combination. “-“ means that the item is omitted.

Furthermore, complex types can be derived from other simple or complex types with further extensions or restrictions. For that, the complex type consists of a <xs:simpleContent> or <xs:complexContent> item with a nested <xs:restriction> or <xs:extension>. These items themselves are not transformed into JSON components.

In a complex content, restrictions delimit the base type to a set of sub-elements and/or delimit the possible values of elements. Extensions add elements to a sub-type. With that, the JSON structure MUST be transformed from the derived version of the type. This means that the JSON values MUST follow the restrictions and extensions in the same way as for XML. Elements added by an extension MUST also be transformed to corresponding JSON items.

In a simple content, restrictions delimit the possible values of the element like for simple types. Extensions can only add attributes to simple types, so they are omitted from the JSON transformation.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

11.4.6 Rules

Generating a JSON representation based upon an XSD is defined as follows. In addition, different coding styles have to be considered, e.g., in XML, closing angle brackets are protected.

1. Element names become usual names, which are part of objects.
2. Any numbers are recognized and used as a JSON number.
3. Booleans are recognized and used as JSON booleans.
4. Empty elements get an empty JSON array as value.
5. “nil” elements get a JSON null value.
6. Elements which may occur in the same place more than once become a JSON array.
7. Attributes get absorbed.
8. Groups are integrated in the used places without creating additional representations. When the “maxOccurs” attribute of the group is greater than 1, it can be integrated several times.
9. The following rules apply on the use of namespaces and namespace prefixes of “explicit SHIP structures” (see section 11.4.1). These rules have an impact on the use of element names:
   a. SHIP namespace definitions of XMLs are not transformed into a JSON representation.
   b. SHIP namespace prefixes (including the colon, e.g. “ship:”) of XMLs are omitted for the transformation into JSON. I.e. element names of explicit SHIP structures do not contain a SHIP namespace prefix in a JSON representation.
10. The following rules apply to the use of namespaces and namespace prefixes of “external specifications” used within the element “data.payload” (see section 11.4.1). These rules have an impact on the use of element names. Whether namespace definitions or namespace prefixes of external specifications are transformed into JSON can be specified per protocolId (see section 11.4.1):
    a. By default, it is assumed that no such transformation is required.
    b. If a namespace definition is to be be transformed into JSON, it is RECOMMENDED to transform it to a JSON object as follows:

       {”@xmlns:namespacePrefix” : ”schemaReference”}

       where “namespacePrefix” is a dedicated namespace prefix and “schemaReference” contains the reference from the XML.
    c. If a namespace prefix is to be transformed into JSON, it is RECOMMENDED to transform it to a JSON object as follows: The element names of the XML shall be used including the namespace prefix and the colon, if present.

       Example: If an XML contains a tag “xyz:foo”, where “xyz” is the prefix, the proper JSON element name would also be “xyz:foo”.

11.4.7 Example Transformations

The following table shows and compares examples for the corresponding XML and JSON representations of the XSD elements:

XSD element	XML representation	JSON representation

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

<xs:element name="height" type="xs:double"/>	<height>1.73</height>	{ "height":1.73 }
<xs:element name="checked" type="xs:boolean"/>	<checked>true</checked>	{ "checked":**true** }
<xs:element name="empty"> <xs:complexType> </xs:complexType> </xs:element>	<empty></empty> or <empty/>	{ "empty":[] }
<xs:element name="nillable" nillable="true"/>	<nillable xsi:nil="true"/>	{ "nillable":null }
<xs:element name="items"> <xs:complexType> <xs:sequence> <xs:element maxOccurs="unbounded" name="item" type="xs:unsignedInt"/> </xs:sequence> </xs:complexType> </xs:element>	<items> <item>1</item> <item>2</item> <item>3</item> </items>	{ "items": [{ "item": [1, 2, 3] }] }
<xs:element name="items"> <xs:complexType> <xs:sequence> <xs:element name="item1" type="xs:unsignedInt"/> <xs:element name="item2" type="xs:unsignedInt"/> <xs:element name="item3" type="xs:unsignedInt"/> </xs:sequence> </xs:complexType> </xs:element>	<items> <item1>1</item1> <item2>2</item2> <item3>3</item3> </items>	{ "items": [{"item1": 1}, {"item2": 2}, {"item3": 3}] }
<xs:element name="items"> <xs:complexType> <xs:choice> <xs:element name="item1" type="xs:unsignedInt"/> <xs:element name="item2" type="xs:unsignedInt"/> <xs:element name="item3" type="xs:unsignedInt"/> </xs:choice> </xs:complexType> </xs:element>	<items> <item1>1</item1> </items> or <items> <item2>1</item2> </items> …	{ "items": [{"item1": 1},] } or { "items": [{"item2": 1 },] }

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

<xs:element name="element">
<xs:complexType>
<xs:sequence>
<xs:element name="item" type="xs:unsignedInt"/>
</xs:sequence>
<xs:attribute name="min" type="xs:unsignedInt"/>
</xs:complexType>
</xs:element>
	<element min="3">
	<item>5</item>
	</element>
		{
		"element":[
		{"item": 5 }
		]
		}

<xs:element name="items">
<xs:simpleType>
<xs:list itemType="xs:unsignedInt"/>
</xs:simpleType>
</xs:element>
	<items>1 2 3</items>	{
		"items": [1, 2, 3]
		}

<xs:element name="items">
<xs:complexType>
<xs:all>
<xs:element name="item1" type="xs:unsignedInt"/>
<xs:element name="item2" type="xs:unsignedInt"/>
<xs:element name="item3" type="xs:unsignedInt"/>
</xs:all>
</xs:complexType>
</xs:element>
	<items>
	<item3>3</item3>
	<item2>2</item2>
	<item1>1</item1>
	</items>
	
	…
		{
		"items": {
		"item3": 3,
		"item2": 2,
		"item1": 1
		}
		}
		
		…

877 *Table 7 Examples for XML and JSON representations.*

878 The following example shows the transformation of an XSD that combines several item types:

XSD

```
  <xs:complexType name="ComplexDataType">
    <xs:sequence>
      <xs:element name="itemDouble" type="xs:double" minOccurs="0">
      </xs:element>
      <xs:element name="itemString" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="complexData" type="ComplexDataType"></xs:element>

  <xs:complexType name="ComplexListDataType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" ref="complexData"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="complexListData" type="ComplexListDataType"/>

  <xs:group name="TestGroup">
    <xs:sequence>
      <xs:element name="itemEmpty">
        <xs:complexType></xs:complexType>
      </xs:element>
      <xs:element name="itemFixed" type="xs:string"
                  fixed="predefined value"></xs:element>
      <xs:element name="optionalItem" type="xs:boolean" minOccurs="0"/>
    </xs:sequence>
  </xs:group>

  <xs:element name="example">
    <xs:complexType>
```

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

```
      <xs:sequence>
        <xs:element ref="complexListData"/>
        <xs:group ref="TestGroup"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

XML representation

```
<example>
  <complexListData>
    <complexData>
      <itemDouble>1.6</itemDouble>
      <itemString>abc</itemString>
    </complexData>
    <complexData>
      <itemDouble>2.4</itemDouble>
      <itemString>def</itemString>
    </complexData>
  </complexListData>
  <itemEmpty/>
  <itemFixed>predefined value</itemFixed>
</example>
```

JSON representation

```
{
  "example":
  [
    {
      "complexListData":
      [
        {
          "complexData":
          [
            [
              {
                "itemDouble": 1.6
              },
              {
                "itemString": "abc"
              }
            ],
            [
              {
                "itemDouble": 2.4
              },
              {
                "itemString": "def"
              }
            ]
          ]
        }
      ]
    },
    {
      "itemEmpty":[]
    },
    {
      "itemFixed":"predefined value"
    }
  ]
}
```

879 *Table 8 Example transformation of several combined XSD item types.*

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

11.5 JSON to XML Transformation

11.5.1 Scope

Converting JSON to a corresponding XML representation can be ambiguous as the expressiveness of the two formats differs. E.g. JSON allows to model empty arrays while XML does not. The transformation rules in this chapter aim to reduce these ambiguities.

11.5.2 Rules

Generating an XML representation based upon a JSON is defined by reversing the rules defined in chapter 11.4. Additionally, the following rules apply:

1. Empty JSON arrays where the corresponding XSD element may occur more than once MUST be ignored.

11.5.3 Example Transformation

The following table shows examples for particular transformations from JSON to XML for given XSD elements.

Table 9 compares two JSON representations "a" and "b" leading to the same XML representation according to the rules in section 11.5.2 for a given XSD permitting multiple occurrences of an element. Representation "a" is the regular representation as it follows the transformation principles from section 11.4 for an XML with absent list items: As the XML contains no list item at all, the same level of information should also be present in the JSON representation. However, as JSON naturally permits representations like "b", it needs to be considered equivalent to "a".

XSD element	JSON representation	XML representation

`<xs:element name="items">`
`  <xs:complexType>`
`  <xs:sequence>`
`  <xs:element`
`maxOccurs="unbounded"`
`name="item"`
`type="xs:unsignedInt"/>`
`  </xs:sequence>`
`  </xs:complexType>`
`</xs:element>`
	a)
	`{`
	`  "items": []`
	`}`
	
	b)
	`{`
	`  "items": [`
	`  {`
	`  "item": []`
	`  }`
	`  ]`
	`}`
		`<items>`
		`</items>`

Table 9: Example for JSON to XML transformation.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

12 Key Management

The essential credentials of the key management in this specification consist of the following key material:

1. **Public keys**: Used by SHIP nodes in the first step of authentication to validate the authenticity of signatures and perform an ECDH key agreement. Also, the corresponding SKI values of the public keys are used for identification and authentication of SHIP nodes.

2. **Private keys**: Each SHIP node also has a private key corresponding to its public key, which is used to generate signatures and perform ECDH key agreements. Note: While a public key has no requirements regarding confidentiality and can be transmitted in band, the private key must be kept secret and should never be transmitted. This is especially important if the private key pair comes along with a corresponding PKI-certificate. In this case, a secure element should be used to protect the stored private key.

3. **Symmetric keys**: Used for mutual authentication and symmetric encryption during efficient reconnection and runtime communication.

4. **PIN**: Optionally used in the second step of authentication (independent from TLS) to improve the trust level. E.g. if only auto accept was used by a SHIP node in the first authentication step, which only offers a relatively low trust level, a PIN can be used to reach mutual authentication and e.g. enable commissioning.

12.1 Certificates

12.1.1 SHIP Node Certificates

A SHIP node MUST have a certificate. No matter if the node acts as SHIP client or SHIP server, a SHIP node MUST always provide a certificate during the TLS handshake for mutual authentication.

A self-signed or PKI based certificate MUST be used.

SHIP node certificates MUST include a SHIP node specific public key.

One public/private key pair SHALL NOT be used for more than one certificate.

One SHIP node certificate SHALL NOT be used for more than one SHIP node.

This SHIP specification does not specify a mandatory PKI.

If a SHIP node does not know the PKI of another SHIP node, the corresponding PKI based certificate is just treated like a self-signed certificate. Hence, interoperability within SHIP does not depend on using a certain PKI.

A SHIP node MAY also receive and manage a revocation list from a web server. If a SHIP node has a synchronized time, it MAY also check whether a certificate is still valid. Other means of certificate evaluation MAY also be used by a SHIP node. However, the additional evaluation of a certificate is only a manufacturer specific topic at the moment.

In general, a SHIP node MUST at least verify the public key of the certificate with one of the four registration modes described in chapter 12.3.1. Any other evaluation of the certificate is optional or manufacturer specific and SHALL NOT affect the general SHIP authentication and communication. This includes certificate lifetime checks, PKI checks, and other checks of the certificate. This means if optional or manufacturer specific checks fail, but the received public key is authenticated correctly, the SHIP node SHOULD still allow communication with the other device. An invalid PKI certificate SHOULD be handled like a self-signed certificate, as trust in SHIP relies on the SHIP node specific public key and not a PKI. E.g. a PKI certificate with an invalid lifetime SHOULD just be handled like a self-signed certificate (no “PKI trust” is given, but if the public key is trusted, this certificate MAY still offer “user trust”). This also applies if a SHIP node does not support a synchronized time to check the lifetime of a PKI certificate. If an

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

optional check fails, a SHIP node MAY inform the user about the reasons for a failed optional check.

Note: If a SHIP node certificate has a lifetime, the manufacturer SHOULD also implement update mechanisms for the certificate.

The certificate of a SHIP node MAY be changed together with the public key, SKI, and the corresponding private key by the user, e.g. via a user interface or commissioning tool. However, via a factory reset the original public key, SKI, private key and certificate SHOULD be restored again, as this is especially important in case the SKI of the public key is printed on a device label.

The CN (common name) field is out of scope for certificates within this SHIP specification. A SHIP node SHOULD ignore the CN (common name field) field of received certificates.

12.1.2 Web Server Based SHIP Node Certificates

A web server-based SHIP node has a special role, as a web server is usually not a local member of the private network and in such cases cannot act as a client. A web server based SHIP node SHOULD have a PKI based certificate.

Therefore, SHIP nodes that want to communicate with a web server-based SHIP node SHOULD have a corresponding CA-certificate for the verification of the web server's certificate.

Note: A CA-certificate has no requirements regarding confidentiality. However, a SHIP node MUST assure that the CA-certificate storage cannot be manipulated.

12.2 SHIP Node Specific Public Key

A SHIP node MUST have a SHIP node specific public key. If the SHIP node also has a SHIP node certificate, this SHIP node specific public key MUST be part of the SHIP node certificate. The SHIP node specific public key has no requirements regarding confidentiality and can be transmitted in-band.

The Subject Key Identifier (SKI) of the SHIP node specific public key MUST be provided to the user.

The Subject Key Identifier SHALL be generated as described in RFC 3280, chapter 4.2.1.2 method (1).

The own SKI value of each SHIP node SHALL be made accessible to the user in full length.

Also, for user verification, the SKI values of other SHIP nodes SHOULD be displayed in full length. The user may decide which parts of the key to compare before accepting the key.

The 20 byte-long SKI of the public key SHALL be provided to the user as 40 hexadecimal digits in the following form. To increase the readability of the SKI and provide interoperability from a user perspective, spaces SHALL be inserted each 4 hexadecimal digits, as shown below:

XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

Example of an SKI user representation on a display or device label:

SKI value: 0x1234AAAAFFFF1111CCCC3333EEEEDDDD99992222

⇨ SKI string: 1234 AAAA FFFF 1111 CCCC 3333 EEEE DDDD 9999 2222

Remark: The SHIP node SKI MAY also be integrated into a QR-code together with the PIN, as described in chapter 12.6.

At least one of the following measures MUST be applied to make the SKI of the public key accessible for the user:

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

- Label: Access to the SKI of the SHIP node specific public key is provided via a label on the SHIP node.

- User interface: Access to the SKI of the SHIP node specific public key is provided via a user interface (e.g. display) of the SHIP node.

- Local communication interface: Access to the SKI of the SHIP node specific public key is provided via a local user communication interface of the SHIP node, e.g. NFC. The user MUST be able to easily access the public key of the SHIP node. Therefore, the local communication must provide easy and user friendly access to the SKI.

- Cloud based user interface: Access to the SKI of the SHIP node specific public key is provided via a user interface in the cloud. The public key must be accessed via the serial number, or some other SHIP node-specific distinctive characteristic.

The public key of a SHIP node MAY be changed by the user along with the SKI, private key and corresponding certificate, e.g. via a user interface or a commissioning tool. However, via a factory reset the original public key, SKI, private key and certificate SHOULD be restored again. This is especially important in case the SKI of the public key is printed on a device label.

12.2.1 Public Key Storage

Each verified public key of another SHIP node SHALL be stored together with the trust level of the verification mode that was used. Public keys that could not be verified MAY be stored as unknown public keys.

Note: To avoid re-verification by user interaction, the trusted public key and its trust level SHOULD be stored persistently.

Unknown public keys SHALL issue a “user trust level” of “0”.

A SHIP node SHOULD offer the user a possibility to remove certain trusted public keys at any time. At least the SHIP node MUST offer a possibility to delete all stored public keys from communication partners (e.g. via factory reset).

12.2.2 Prevent Double Connections with SKI Comparison

The public key SHALL also be used to prevent double connections. If a SHIP node recognizes that there are two or more simultaneous connections to another SHIP node, the SHIP node with the bigger 160 bit SKI value SHALL only keep the most recent connection open and close all other connections to the same SHIP node (a previous release of this SHIP specification may permit a different preference). If an older connection is already in the SME data exchange phase, the SHIP node with the bigger SKI value SHOULD initiate a connection termination as described in section 13.4.7.

In general, each SHIP node may close a connection – even the SHIP node with the smaller SKI value – if a timeout was detected or the SHIP node with the bigger SKI value does not close double or multiple connections to the same SHIP node within 3 seconds. After a timeout of 3 seconds, the device with the smaller SKI value SHALL send a WebSocket ping. Connections that are not pingable (i.e. where no proper Pong frame is received as response) SHOULD be closed. If multiple connections are still pingable, the SHIP node with the smaller SKI value MAY close the older connection. If an older connection is already in the SME data exchange phase, the SHIP node with the smaller SKI value SHOULD initiate a connection termination as described in section 13.4.7.

The SHIP node with the greater SKI SHOULD check for double connections directly during the TLS handshake.

12.3 Verification Procedure

SHIP nodes that possess one or more CA-certificates MAY check whether a received certificate is a PKI or self-signed certificate.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

A communication partner with a matching PKI certificate MAY gain additional PKI trust, depending on the trustworthiness of the corresponding CA.

A SHIP node MUST always verify the public key of the communication partner with one of the following verification modes.

12.3.1 Public Key Verification Modes

Each public key verification mode provides a certain user trust level, the “user trust”. While the verification mode describes the concrete mode, the user trust level maps the different modes on a generic value. In each of the public key verification modes, user interaction is necessary to establish user trust between two SHIP nodes to ensure user consent.

When a stored public key is reused, it MUST be possible to derive the trust level of the public key. The following Public Key verification modes exist:

1. Auto accept: user trust level = 8
2. User verify: user trust level = 32
3. Commissioning: user trust level = 32-96 (depending on the trustworthiness of the “commissioning tool”)
4. User input: user trust level = 64

The user trust level of a public key can be adjusted during runtime whenever a public key is re-verified in a more secure manner. If for example an “auto accepted” public key is later re-verified successfully as a “commissioned” public key, the trust level SHALL be adjusted to the “commissioned” trust level.

A SHIP node MUST implement at least one of the verification modes. Two or more verification modes MAY be active in parallel during runtime. To allow the user to choose between an easy verification mode and a more secure option, it is RECOMMENDED to implement two or more verification modes.

In general, the “commissioning” mode is always a good option for most devices, as it can be used in combination with a smart phone or a web service to provide a very user friendly method of establishing a high level of trust between SHIP nodes.

12.3.1.1 Auto Accept

This mode should only be implemented by very simple SHIP nodes without any user interface (UI), as it is the least secure verification mode.

If “auto accept” is triggered in a SHIP node (e.g. by push button) by the user, the SHIP node SHALL open a time window in which it MUST automatically accept an unknown public key that is received during registration.

Note: Only one public key SHALL be accepted during a single “auto accept” time window. Hence, after accepting one unknown public key during “auto accept”, the “auto accept” mode SHALL instantly be deactivated.

As “auto accept” skips the public key verification of the received unknown public key (thus introducing the potential risk of man-in-the-middle attacks), the duration for the auto accept time window SHALL be kept as low as possible from a usability perspective and MUST be lower than or equal to 2 minutes. However, please keep in mind that man-in-the-middle attacks are still possible even with a shorter time window.

12.3.1.2 User Verification

This mode SHOULD be implemented by any SHIP node with an appropriate display or communication interface for the user.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

If “user verification” is used, the SHIP node MUST inform the user when unknown SKI values of public keys are presented during local service discovery of other nodes via mDNS or during a TLS handshake. The information MUST include the SKI value of the public key. “User verification” can also be applied to already stored public keys to increase the user trust level or discard public keys.

12.3.1.3 Commissioning

This mode SHOULD be implemented by any SHIP node that can be connected with an appropriate commissioning tool.

Note: The trust level of a public key that was verified via “commissioning” mode depends on the trustworthiness of the used commissioning tool.

If SKI values of public keys are received from a commissioning tool, those SKI values SHALL be stored persistently together with the user trust level of the commissioning tool and used during verification to identify matching public keys. A SHIP node SHALL also check if there are already known SHIP nodes with a matching public key and adjust the trust level accordingly.

A commissioning tool MAY also be used to provide access to a revocation list or to update key material of a SHIP node.

Manufacturers are free to use their own solution for commissioning. However, to provide interoperability for B2B and from a user perspective, an interoperable commissioning functionality that can be reached via a trustworthy communication channel such as SHIP should be used. The commissioning functionality may be reachable via SHIP by using “auto accept” and an additional PIN or “user verify” or “user input” or “commissioning”.

Note: As “commissioning” provides user trust, the user SHALL be part of the commissioning procedure and user consent is required.

12.3.1.4 User Input

This mode SHOULD be implemented by any SHIP node that has an appropriate interface for out of band data input, e.g. an appropriate user interface.

If SKI values of public keys are entered into the SHIP node, those SKI values SHALL be stored persistently together with the user trust level of user input and used during verification to identify matching public keys. A SHIP node SHALL also check if there are already known SHIP nodes with a matching public key and adjust the trust level accordingly.

12.3.2 Trust Level

The trust level expresses the trust in a certain communication partner using generic values. The higher the values, the stronger the trust in the corresponding communication partner.

The trust level consists of different categories, which include different mechanisms and permit a differentiated view of the trustworthiness of a communication partner. Currently, there are the following trust level categories in SHIP:

1. User trust

verification mode	user trust level value
none	0
auto accept	8
user verified	32
commissioned	32-96 (depending on trustworthiness of commissioning tool)
user input	64

Table 10: User Trust

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

2. PKI trust

PKI verification	PKI trust level value
self-signed	0
signed by PKI	0-65535 depending on SHIP node policy / trust in certain PKI

Table 11: PKI Trust

3. Second factor trust

second factor	Second factor trust level value
none	0
PIN	16 or 32 (see section 12.5: "32" reserved for first PIN transmitter)

Table 12: Second Factor Trust

If multiple mechanisms are used from the same category, only the mechanism which offers the highest trust level in this category SHALL be accounted for. E.g. if a SHIP node has verified a public key with “auto accept” and “user verify”, only “user verify” is accounted for and therefore the “user trust” value is “32”.

A “user trust” of “8” is the minimal “user trust” that is required for general SHIP communication. This means if the “user trust” is less than “8”, the SME “hello” handshake SHALL be aborted, like described in chapter 13.4.4.1.

For commissioning via SHIP, a trust level of “32” or higher MUST be achieved in the “user trust level” or “second factor trust level” category. E.g. a “second factor trust level” of “32” would allow commissioning over SHIP, but also a “user trust level” of “32” would allow commissioning.

The PKI trust depends on the manufacturer's policy. PKI certificates are not mandatory, hence general communication SHALL also be possible without the use of a trusted PKI and a “PKI trust level” of “0”.

The PIN is currently the only second factor authentication mechanism and MAY provide an additional trust level value of “16-32” in the “second factor trust” category, as described in chapter 12.5.

A layer above SHIP can use the trust level to control access to certain functionality. The trust level requirements MAY differ depending on the feature, the kind of application/use case, and legal or device related security requirements. Some privacy relevant use cases might require a high “user trust” while manufacturer specific use cases might have a high requirement regarding the “PKI trust level”.

12.4 Symmetric Key

Depending on the chosen security method, a SHIP node SHOULD store the necessary key material in order to reconnect in an efficient manner.

If TLS was used, the session state of this connection SHOULD be stored and reused during a reconnection, as described in chapter 9.6.

The session state SHALL be stored in alignment with the public key and the trust level of the corresponding communication partner.

12.5 SHIP Node PIN

The SHIP node PIN provides a very user friendly way to reach mutual authentication, e.g. via QR-code scan between a smart phone and a SHIP node that uses auto accept and additionally waits for a valid PIN. The SHIP node PIN transmission is described in chapter 13.4.4.3.

The SHIP node PIN is bound to the public key/SKI of the SHIP node. Therefore, the SHIP node PIN SHALL only be transmitted to a SHIP node that is the cryptographically proven owner of

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

the corresponding public key. After the SHIP node PIN was transmitted, the sender SHOULD discard the PIN.

The PIN is an authentication secret that must be kept confidential and SHALL only be shared with authenticated and authorized communication partners. Therefore, the SHIP node PIN SHALL NOT be transmitted if the public key of the corresponding communication partner has a user trust level that is less than “32”.

The first communication partner after factory default that sends the SHIP node PIN MAY gain a higher second factor trust level of “32” and therefore MAY gain access to special functionality. E.g. a communication partner that has a second factor trust of “32” MAY act as a “commissioning tool” towards the SHIP node. In SHIP, it is not possible that two SHIP nodes may gain a second factor trust of “32” with the SHIP node PIN. Any SHIP node that sends the PIN afterwards SHALL only get a second factor trust of “16”. If another communication partner should be the one with a second factor trust of “32”, a factory reset must be performed. After factory reset, the first communication partner that sends the SHIP node PIN MAY gain a higher second factor trust level of “32” again.

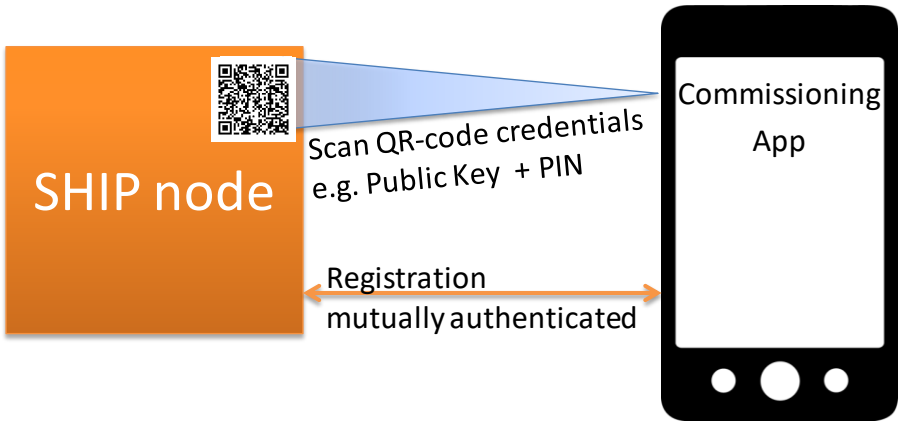


Figure 5: Easy Mutual Authentication with QR-codes and Smart Phone

The SHIP node PIN SHALL be provided to the user as 8-16 hexadecimal digits in the following form (equivalent to 32-64 Bit PIN). To increase the readability of the PIN and provide interoperability from a user perspective, spaces should be inserted between every 4 hexadecimal digits in a graphical presentation as shown below:

XXXX XXXX (8 digits)

XXXX XXXX X (9 digits)

...

XXXX XXXX XXXX XXXX (16 digits)

Example of a 40-Bit PIN user representation on display or device label:

- Graphical representation: 5555 AAAA FF
- Corresponding 40-Bit PIN value in hexadecimal format: 0x5555AAAAFF

Remark: The SHIP node PIN MAY also be integrated into a QR-code together with the SKI, as described in chapter 12.6.

This subsection only addresses the PIN in the scope of key management. The actual transmission format is described in subsection 13.4.4.3 of the SHIP data exchange chapter.

The PIN of a SHIP node MAY be changed by the user, e.g. via a user interface or commissioning tool. However, via a factory reset the original PIN MUST be restored again. This is especially important if the original PIN is printed on a device label.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

If a SHIP node PIN is changed during runtime, this SHALL NOT affect any trust level of already trusted SHIP nodes. If a user wants other SHIP nodes to reenter the PIN, the user MUST force a re-registration of the corresponding SHIP node by deleting the trust relationship to the corresponding SHIP node.

12.6 QR Code

QR Code Model 2 with at least “low” ECC level SHOULD be used.

The size of each module of the QR-code SHALL be at least $330*10^{-6}$ metres. This equals a module size of 4 pixels when printed with 300dpi.

The quiet zone SHALL have a width of at least 4 modules.

Certain SHIP specific data MAY be integrated into a QR-code. The main advantage of the QR-code is that it allows a very user friendly and mutually authenticated connection establishment with smart phones via “scan”, ideally also enabling the smart phone as “commissioning tool”.

If SHIP data is integrated, the following encoding rules MUST be fulfilled:

1) **SHIP prefix**: The SHIP specific data in the QR-code SHALL start with a SHIP prefix, the string “SHIP;” in UTF-8 encoding.

2) **SKI (mandatory)**: After the SHIP prefix, the SKI MUST follow. The SKI MUST be encoded as follows:

   a) The first 4 bytes MUST include the string “SKI:” in UTF-8 encoding.

   b) The next bytes MUST include the SKI value as a non-prefixed hexadecimal string in UTF-8 encoding with an additional space every 4 hexadecimal digits, as also described in chapter 12.2. Upper or lower case letters MAY be used (0-9, A-F, a-f).

   c) The last byte MUST include the character ';' (semicolon) in UTF-8 encoding. This marks the end of the hexadecimal SKI string.

   d) Example: `“SKI:5555 AAAA FfFf 1111 CCCC 3333 EEeE ddDD 9999 2222;”`

3) **PIN (mandatory if SHIP node has a PIN)**: If the SHIP node also has a PIN, the encoded PIN SHALL follow after the encoded SKI. The PIN MUST be encoded as follows:

   a) The first 4 bytes MUST include the string “PIN:” in UTF-8 encoding.

   b) The next bytes MUST include the PIN as non-prefixed hexadecimal string in UTF-8 encoding with an additional space every 4 characters, as also described in chapter 12.5. Upper or lower case characters MAY be used (0-9,A-F,a-f).

   c) The last byte MUST include the character ';' (semicolon) in UTF-8 encoding. This marks the end of the hexadecimal PIN string.

   d) Example:`“PIN:5555 AaAa FF;”`

4) **ID (recommended)**: The SHIP ID. After the PIN, if present, or otherwise after the SKI, if the PIN is not present, the ID SHOULD follow. The ID MUST be encoded as follows:

   a) The first 3 bytes MUST include the string “ID:” in UTF-8 encoding.

   b) The next bytes MUST include the ID value as string in UTF-8 encoding. The ID value itself MUST NOT contain a semicolon character!

   c) The last byte MUST include the character ';' (semicolon) in UTF-8 encoding. This marks the end of the ID string.

   d) Example: `“ID:EXAMPLEBRAND-EEB01M3EU-001122334455;”`

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

5) **BRAND (optional)**: MAY be used after ID to provide brand information about the device in the QR-code.

   a) The first 6 bytes MUST include the string “BRAND:” in UTF-8 encoding.

   b) The next bytes MUST include the brand information as string in UTF-8 encoding. The brand information itself MUST NOT contain a semicolon character!

   c) The last byte MUST include the character ';' (semicolon) in UTF-8 encoding. This marks the end of the BRAND string.

   d) Example: `“BRAND:EXAMPLEBRAND;”`

6) **TYPE (optional)**: MAY be used after BRAND to provide type information about the device in the QR-code.

   a) The first 5 bytes MUST include the string “TYPE:” in UTF-8 encoding.

   b) The next bytes MUST include the type information as string in UTF-8 encoding. The type information itself MUST NOT contain a semicolon character!

   c) The last byte MUST include the character ';' (semicolon) in UTF-8 encoding. This marks the end of the TYPE string.

   d) Example: `“TYPE:DISHWASHER;”`

7) **MODEL (optional)**: MAY be used behind TYPE to provide model information about the device in the QR-code.

   a) The first 6 bytes MUST include the string “MODEL:” in UTF-8 encoding.

   b) The next bytes MUST include the model information as string in UTF-8 encoding. The model information itself MUST NOT contain a semicolon character!

   c) The last byte MUST include the character ';' (semicolon) in UTF-8 encoding. This marks the end of the MODEL string.

   d) Example: `“MODEL:EEB01M3EU;”`

Example QR-code encoding with only SKI and PIN:

```
“SHIP;SKI:5555 AAAA FFFF 1111 CCCC 3333 EEEE DDDD 9999 2222;PIN:5555
AAAA FF;”
```

Figure 6: QR Code Model 2, “low” ECC level, 0.33mm/Module, with SKI and PIN

The QR code with SKI and PIN has a size of 33x0.33mm = 10.89mm (without quiet zone).

Example QR-code encoding with only mandatory values:

```
“SHIP;SKI:5555 AAAA FFFF 1111 CCCC 3333 EEEE DDDD 9999 2222;PIN:5555 AAAA
FF;ID:EXAMPLEBRAND-EEB01M3EU-
001122334455;BRAND:EXAMPLEBRAND;TYPE:DISHWASHER;MODEL:EEB01M3EU;“
```

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Figure 7: QR Code Model 2, “low” ECC level, 0.33mm/module, with all values

The QR code with all values has a size of 47x0.33mm = 15.51mm (without quiet zone). In a different example, the size might vary because of the variable length of ID, BRAND, TYPE and MODEL.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

13 SHIP Data Exchange

13.1 Introduction

This section (13.1 only) is informative only for the underlying revision of the specification. It is normative for the development of a successor of this specification (though the successor may well adjust this section accordingly).

The concept makes use of the following assumptions:

1. The exchange of data between two SHIP nodes is considered (i.e. no routing to other SHIP nodes and no routing/branching within SHIP nodes are considered).
2. The communication between two SHIP nodes is connection oriented.
3. The connection is bidirectional.
4. Only so-called SHIP messages are exchanged (i.e. no streams).
5. SHIP Transport assumes a communication channel that is chosen or defined and used in a manner to permit safe message separation for every supported message format (this is relevant for binary formats).
6. The communication channel is reliable (verification of successful data transmission).
7. SHIP messages are delivered in the same order as they were submitted.

The concept is designed to permit modifications on the assumptions and mechanisms in future versions of the specification as long as extensibility and compatibility mechanisms are properly considered.

13.2 Terms and Definitions

“Server”, “Client”, connection role

In this chapter, the terms “server” and “client” are primarily used with regards to an underlying connection technology, i.e. they are usually NOT used as functional roles such as “light switch” or “time information server”. See 13.4.1 for details.

Message Definition

This specification provides definitions for so-called SHIP messages. The definitions make use of miscellaneous description concepts (XSD, binary structure, table, ...). For each message, there can also be a process definition on the use of the message.

Message Parts and Composition

A message is composed of message parts. The parts can have different requirements regarding representation.

Example: The "message type" part can be a byte whereas the "message value" can be represented with JSON-UTF8 or another agreed format.

Representation

A representation is an instance of a message part in an "official" format (which implies rules on its use). Each format is a "wire format".

Depending on the scope or message variant (see below) different representations are permitted (JSON-UTF8, JSON-UTF16, binary, "mixed"/message dependent).

Note: For a given message variant, all permitted representations must be equivalent (i.e. there must be a lossless conversion available between the representations)!

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Note: Subsequent versions may use further representations.

Message Wire Format

The message wire format determines the wire format of the whole message, i.e. the composition of all message parts. This requires that the message format allows to determine all representations.

Unless stated otherwise, the term "SHIP message" refers to a wire format. In the wire format, a SHIP message is of limited size and has a start and an end.

Format Descriptor

For each message wire format, a format descriptor is defined. The format descriptor can be used during protocol handshake to agree on the wire format for subsequent communication.

SHIP Message Facets

This specification defines different variants and types of SHIP messages.

The variant of a SHIP message is uniquely determined by the SHIP transmission context or by the message itself (using a message variant identifier).

Examples for SHIP message variants: CMI message (see 13.4.3), connectionHello (SME “Hello” message, see 13.4.4.1), etc.

Each variant belongs to a SHIP message type. These types (“init”, “control”, etc.) are defined in Table 13.

Extensibility

Definitions of messages and message parts can prescribe if or how they can be extended with content not explicitly specified in this specification.

Unless stated otherwise, extension rules serve to achieve and preserve forward and backward compatibility between devices belonging to different SHIP releases. Rules for manufacturer-specific extensions are given separately and are marked for this purpose.

"RFU" - "Reserved for Future Use"

Definitions that are marked with "RFU" denote potential future extensions of the specification. Such extensions can be defined by the specification authority ONLY. Under no circumstances shall such extensions be used for manufacturer-specific extensions.

Remark: This rule is crucial in order to prevent ambiguities and to keep interoperability. Therefore, for subsequent releases of the specification, the specification authority can give concrete specifications for regions formerly marked with "RFU", regardless of any manufacturer-specific use.

Default Structure Extensibility

A structure consists of a "parent" element with zero or more optional or mandatory "child" elements. The child elements have no meaning without the parent element.

A well-defined structure is a structure definition of this specification. Among other things, a definition imposes rules for the unique identification of parent and child elements, on the order and presence of child elements, and on their types. The child elements of a well-defined parent are called "known children".

By default, no other child than a "known child" is permitted in a well-defined parent.

The "default structure extensibility" is a property that can be associated with a given structure. This property is defined as follows: It marks the structure as being extensible by the

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

specification authority in a specific way. A future version of the specification may define further child elements beyond the last "known child" of the current revision's parent element. This property applies to immediate children of the parent only (i.e. not to second-degree children, e.g.). The "default structure extensibility" applies only where explicitly mentioned.

The property "default structure extensibility (recursive)" extends the "default structure extensibility" recursively, i.e. down to all children that are themselves parents for their children.

Remark: This simply means that a future specification may append new children to a parent that permits default structure extensibility. It does **not** mean that new children can appear before or between known children. This is most relevant for a serialized form of a structure instance. The default structure extensibility also does **not** mean that manufacturer-specific children are permitted.

13.3 Protocol Architecture / Hierarchy

13.3.1 Overview

The following protocol architecture is used to define responsibilities for data exchange and interfaces or algorithms:

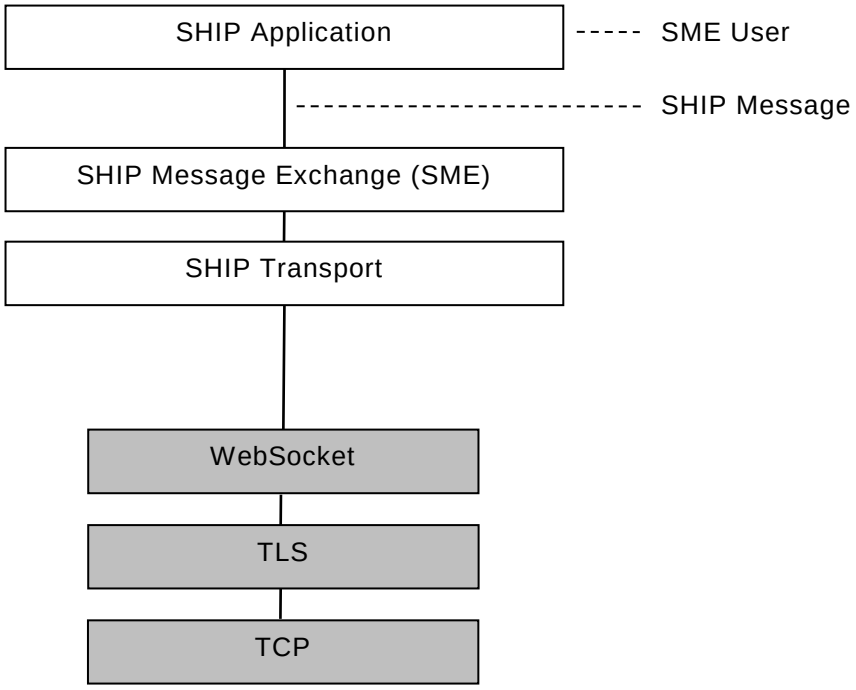


Figure 8: Protocol Architecture and Hierarchy

This chapter focuses on the elements with white boxes. The grey boxes are for reference only.

This specification does not require implementations to provide modules or functionalities to be structured as shown by the architecture. Rather, the architecture provides clear definitions to also ensure extensibility and flexibility in future releases of this specification.

13.3.2 SHIP Message Exchange (SME), SME User

The interface of SME towards a SHIP Application (i.e. the “SME User”) is called “SME-AP”. Using the ISO-OSI model terminology, SME-AP is the service access point of the SHIP Message Exchange Instance (SME-I). The SME-AP indicates received SHIP messages and takes SHIP messages for the transport to another SME-AP. A SHIP Application uses SME-AP to receive or send SHIP messages.

The SME-I protocol is the SHIP message in the wire format. SME-I itself requires a service access point to a SHIP Transport Instance to perform the message exchange.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Technical details on SME are described in 13.4.

Remark: Future versions of the specification may also define the exchange of a "SHIP Stream" as a further service. The service would probably be parallel to SME-I in terms of a layered hierarchy. It would also require a (properly adjusted) SHIP Transport Instance.

13.3.3 SHIP Transport

A SHIP Transport Instance is responsible for extracting SHIP messages from received data and offer it to SME-I. For the opposite direction, it is responsible for taking SHIP messages from SME-I and transmit proper data to the communication partner.

In this revision of the specification, the SHIP Transport Instance is implemented as follows:

SHIP Transport itself uses the message frame of WebSocket for data exchange (dependent on the used security method). Exactly one SHIP message (in the wire format) is exchanged in exactly one WebSocket message. This means exactly the last WebSocket fragment of a SHIP message has the FIN bit set.

Remark (informative): This basically means that SHIP Transport defines how WebSocket shall be used for SHIP Transport specific needs. This applies to at least this revision of the specification. Future versions of the specification may use other technologies as an alternative to WebSocket. The support of a "SHIP Stream" may also require changes. This can, for example, result in the definition of an additional frame around the SHIP messages. As a consequence, this version of the specification must provide at least a possibility for future extensions on the transport layer and initial compatibility rules. One step towards the preparation for future extension is the definition of "CMI".

13.4 SHIP Message Exchange

13.4.1 Basic Definitions and Responsibilities

Underlying Connection States

This specification does not define how a connection between the SME instances of two SHIP nodes is established. It just assumes that a lower layer or internal mechanisms will provide a connection that is either open or closed.

SME Connection and States

An SME connection is a connection between the SME instances of two SHIP nodes. Depending on the underlying connection state, the SME connection state is also open or closed. In addition to these basic states, each SME instance can have further states. These states can change depending on exchanged SHIP messages. In the following sections, the messages and states are explained in more detail.

SME Connection and Roles

It is REQUIRED that each SME instance has a unique role assigned for each connection. This role is either “server” or “client”. If one of the communication partners has the role “server”, the other communication partner MUST have the role “client”. The role is constant as long the connection is not closed. Unless stated otherwise, it is NOT required that a reconnection between the communication partners assigns the same roles as the previous connection.

Note: A SHIP node MAY have multiple (distinct) SME connections to different communication partners. The SHIP node is permitted to have different roles per connection.

Note: The role described in this section relates to the SME connection only. It does NOT impose any “functional role” related with application specific messages (e.g. a role as “energy management server” would be independent from the SME connection role).

Remark: This specification does not describe how this information is gained from lower layers or implementations.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Responsibilities

All messages and processes described within section 13.4 are the responsibility of the user of the SME-AP (i.e. of SME User), unless stated otherwise.

Remark (informative): First of all, the use of “SME User” instead of “SHIP Application” shall help make clear that it is SME that imposes rules on its use. I.e. the rules are not determined by the SHIP Application. Secondly, in general, a layer should avoid mentioning a specific upper layer.

13.4.2 Basic Message Structure

A SHIP message is formally defined using ABNF:

```
Message = MessageType MessageValue

MessageType = OCTET

MessageValue = 1*OCTET
```

The following values are defined for MessageType:

Value	Name
0	init
1	control
2	data
3	end
4-255	RFU

Table 13: MessageType Values

If an SME User receives a message with unknown type, it SHALL silently discard it or close the communication channel.

Remark (informative): The leading type identifier is primarily a preparation for future binary formats.

The subsequent sections define several messages with their type and value. In most cases, the proper definition for MessageValue is given in a text-based format. However, as will be seen later on, the SHIP protocol is prepared for multiple formats.

13.4.3 Connection Mode Initialisation (CMI)

As soon as an SME instance has opened a connection to a communication partner, it enters the SME connection state “CMI_INIT_START”, which immediately refers to a connection role specific state.

Remark (informative): CMI is designed to permit more efficient reconnections or format agreements in subsequent versions of this specification. The concept ensures the definition of a compatibility concept between these versions through fall-back mechanisms. I.e. devices based upon newer versions of the specification will benefit from more efficient procedures while even a firmware/specification downgrade will not do any harm because of a robust fall-back to version 1.0.

A CMI message is defined as follows:

```
MessageType = %x00 ; init

MessageValue = CmiHead CmiRemainder

CmiHead = OCTET

CmiRemainder = *OCTET
```

The permitted content and meaning of MessageValue is defined as follows:

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

1. If the first byte (CmiHead) is 0: State "Connection data preparation".

2. Else: RFU

In this version of the specification, only CmiHead of MessageValue of a received message is analysed. CmiRemainder is reserved for future use. The following process (including state information) is defined and SHALL be performed:

1. SME connection role “client”:

    1.1. CMI_STATE_CLIENT_SEND: Client sends a CMI message with MessageValue = 0 to “server” and enters state CMI_STATE_CLIENT_WAIT immediately afterwards.

2. SME connection role “server”:

    2.1. CMI_STATE_SERVER_WAIT: “Server” waits for the CMI message from “client”.

    2.2. CMI_STATE_SERVER_EVALUATE: “Server” evaluates the received message.

        2.2.1. If the received MessageType is not 0: “Server” sends a CMI message with MessageValue = 0 to “client” and closes the connection afterwards.

        2.2.2. If the received CmiHead has the decimal value 0: “Server” sends a CMI message with MessageValue = 0 to “client” and enters the SME connection state "Connection data preparation".

        2.2.3. If the received CmiHead has a decimal value greater than 0: “Server” sends a CMI message with MessageValue = 0 to “client” and closes the connection afterwards.

3. SME connection role “client”:

    3.1. CMI_STATE_CLIENT_WAIT: “Client” waits for the CMI message from “server”.

    3.2. CMI_STATE_CLIENT_EVALUATE: “Client” evaluates the received message.

        3.2.1. If the received MessageType is not 0: “Client” closes the connection immediately.

        3.2.2. If the received CmiHead has the decimal value 0: “Client” enters the SME connection state "Connection data preparation".

        3.2.3. If the received CmiHead has a decimal value greater than 0: “Client” closes the connection immediately.

Timeout procedure:

The CMI process above begins as soon as a connection has been established (entering state CMI_INIT_START). This corresponds to the process steps CMI_STATE_CLIENT_SEND and CMI_STATE_SERVER_WAIT, respectively. Then, the following rules apply:

1. For process step CMI_STATE_SERVER_WAIT, a timeout of CmiTimeout applies. If the server does not receive a message before this time elapses, it SHALL close the connection immediately.

2. For process step CMI_STATE_CLIENT_WAIT, a timeout of CmiTimeout applies. If the client does not receive a message before this time elapses, it SHALL close the connection immediately.

An SME User SHALL assign any value from 10 seconds to 30 seconds to CmiTimeout.

Remark on CmiHead “RFU” and potential definitions in subsequent specification versions:

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

In a subsequent version of this specification, values other than “0” may be defined. In fact, it is possible to define a multi-byte “extended CmiHead”. Compatibility as well as fault tolerance can be preserved through the rule to reconnect with CmiHead “0” if a previous connection with another value failed.

A brief overview of the CMI procedure is given in the following sequence diagram.

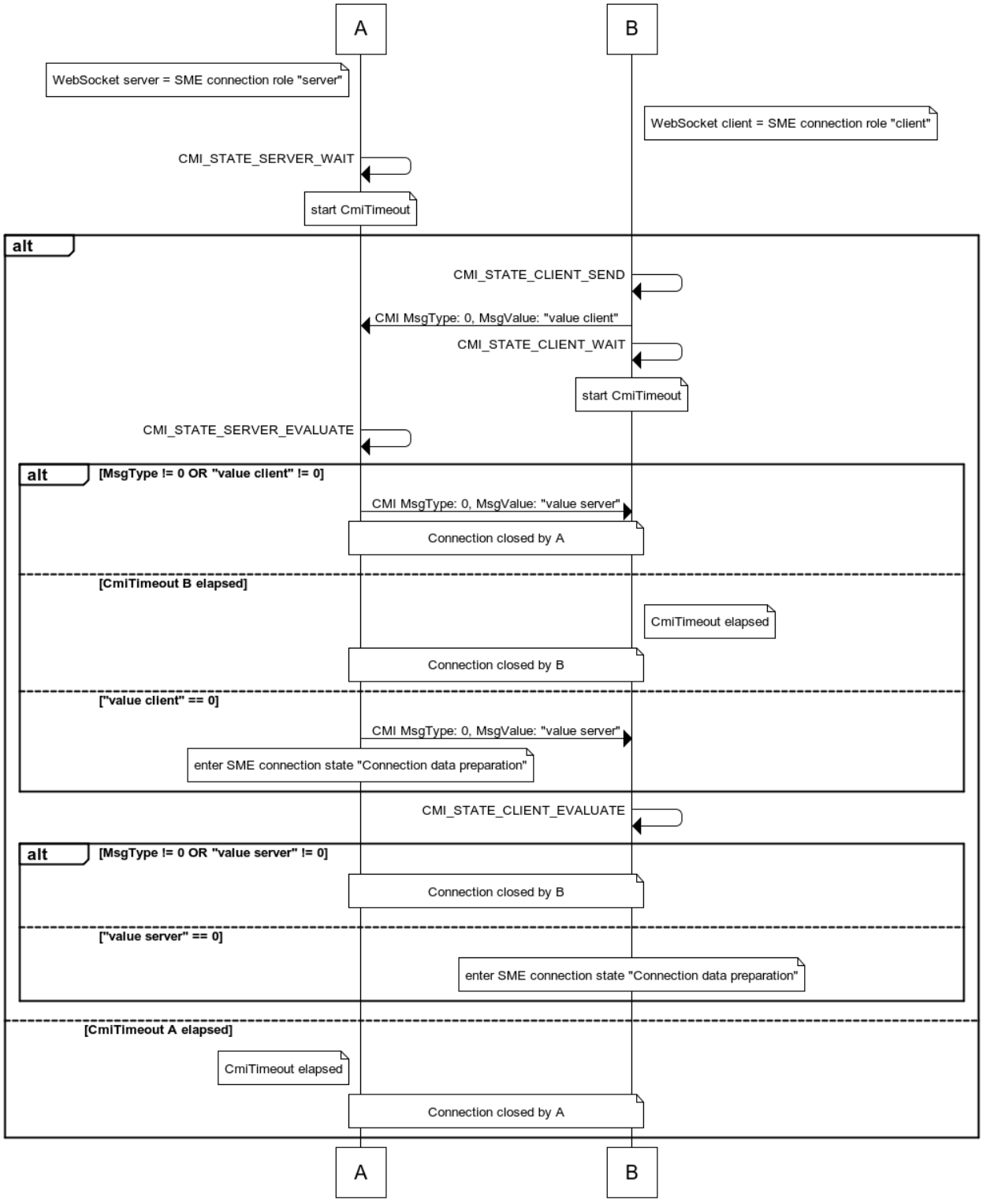


Figure 9: CMI Message Sequence Example

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

13.4.4 Connection Data Preparation

If an SME User enters this state, it SHALL proceed with “Connection state Hello” (see 13.4.4.1).

13.4.4.1 Connection State “Hello”

13.4.4.1.1 Basic Definitions

In this state, the SME Users negotiate whether they allow to continue with the next communication state or not. This concept enables implementations to give a user of a SHIP node some time to decide whether the communication partner can be trusted or not. From a process point of view, a connection that is not (yet) trusted is considered “PENDING”, whereas it is “READY” as soon the connection is trusted (the terms “READY” and “PENDING” are defined in section 13.4.4.1.2).

Remark (informative): Trusting a device or not is typically related to the phase when a device just presented its certificate and public key (see chapter 12) in order to allow a user to verify this information (provided there is a proper user interface). I.e. this procedure can start even before Connection Mode Initialisation begins (and in case of auto_accept, it can also be finished before CMI begins). In case of a real user based (manual) verification, it is likely that such a verification time is required at least for the very first establishment of a connection between two SHIP nodes.

This state uses an SME “hello” message, which is defined as follows:

```
MessageType = %x01 ; control

MessageValue = SmeHelloValue

SmeHelloValue = *OCTET
```

The content of SmeHelloValue is defined as follows: The structure is defined by the SHIP root tag “connectionHello” (including the root tag “connectionHello”) of the XSD “SHIP_TS_TransferProtocol.xsd”. The default structure extensibility applies to this structure. The format of this structure MUST be JSON-UTF8.

Element name	Mandatory/ Optional/ Not Valid (NV)	Brief explanation
connectionHello.phase	M	The sender’s phase during the “hello” process (enumeration: pending, ready, aborted). See 13.4.4.1.2.
connectionHello.waiting	O	Remaining time (in milliseconds) granted by the sender.
connectionHello.prolongationRequest	O	Request to prolong the remaining time.

Table 14: Structure of SmeHelloValue of SME "hello" Message.

The SME “hello” process does not require knowledge of the connection role (server or client). Instead, each of the SME Users SHALL execute the process as described below.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

13.4.4.1.2 Process Overview

When this process is entered, an SME User SHALL be in exactly one of the following basic "Hello" states for a given connection to another node (depending on the trust in the connection, see 13.4.4.1.1):

1. **READY**
   This state MUST ONLY be entered if the communication partner is already trusted. In this state, the SME User is ready to proceed with the next state after "Hello" as soon as it receives from the communication partner that it is ready to proceed as well. The SME User will only wait for this information from the communication partner for a limited time. However, the communication partner can request a prolongation of this time.

2. **PENDING**
   In this state, the SME User is not yet ready to proceed with the next state after "Hello". Nevertheless, it will only wait for a final "READY" information from the communication partner for a limited time, although the communication partner can request a prolongation of this time (this is the same principle as for state "READY"). Furthermore, the SME User must ensure to

   a. EITHER switch into state "READY" and inform the communication partner accordingly in time,

   b. OR request a prolongation of the time the communication partner waits for the "READY" information,

   c. OR report the abortion of this process if the SME User finally decides not to trust the communication partner.

The third basic state "HELLO_OK" can only be accessed if certain conditions are fulfilled as described in the following sections.

Please note: The connection state "Hello" is defined independent from any authentication procedures of any lower layer. It just assumes the knowledge whether a communication partner is trusted or not. This means that lower layers need to specify the procedures and conditions to trust another device and whether to keep this information persistently or not.

13.4.4.1.3 Process Details

Timer Overview:

The following timers are defined:

1. Wait-For-Ready-Timer

   Default value: T_hello_init (see below).

   Purpose: The communication partner must send its "READY" state (or request for prolongation") before the timer expires.

2. Send-Prolongation-Request-Timer

   Purpose: Local timer to request for prolongation at the communication partner in time (i.e. before the communication partner's Wait-For-Ready-Timer expires).

3. Prolongation-Request-Reply-Timer

   Purpose: Detection of response timeout on prolongation request.

Each timer operates in a countdown mode. If a timer is activated, it MUST be initialized with a value greater than 0 in general, and according to the rules of the corresponding process step specifically. Details on the use of the timers are described for the appropriate process steps. See also "Implementation Advice".

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

The following symbols for constant time values are used:

1. T_hello_init: Any value from 60 seconds up to (and including) 240 seconds. An implementation can choose for any value of the specified range. However, this value SHALL be constant during a connection. This value SHOULD also be constant across different connections.
2. T_hello_inc: The same value as T_hello_init.
3. T_hello_prolong_thr_inc: 30 seconds.
4. T_hello_prolong_waiting_gap: 15 seconds.
5. T_hello_prolong_min: 1 second.

States and Sub-states Overview:

Depending on the basic state, the following sub-states are defined:

1. Sub-states of basic state „READY“:
    1.1. SME_HELLO_STATE_READY_INIT
    1.2. SME_HELLO_STATE_READY_LISTEN
    1.3. SME_HELLO_STATE_READY_TIMEOUT
2. Sub-states of basic state „PENDING“:
    2.1. SME_HELLO_STATE_PENDING_INIT
    2.2. SME_HELLO_STATE_PENDING_LISTEN
    2.3. SME_HELLO_STATE_PENDING_TIMEOUT

General information on the basic state „READY“:

In this case, the SME User need not request a prolongation with the communication partner. Thus, Send-Prolongation-Request-Timer is not needed. Of course, this requires submitting information on the own „READY“-state to the communication partner. Consequently, any sub-element „waiting“ for an SME „hello“ message received from the communication partner can be ignored.

General information on the basic state „PENDING“:

In this case, the use of Send-Prolongation-Request-Timer is required as soon as the “waiting” sub-element from the communication partner is available, independent from the communication partner’s phase (except for “aborted”).

As described above, an SME User with state “PENDING” requires the communication partner to also announce its basic state “READY” in time. Thus, the Wait-For-Ready-Timer is required as long as this information has not been received.

Sub-state SME_HELLO_STATE_READY_INIT:

This is the first state an SME User SHALL enter if it is in the basic state „READY“. In this state, it SHALL

1. initialise Wait-For-Ready-Timer to the default value and start the timer,
2. deactivate Send-Prolongation-Request-Timer and Prolongation-Request-Reply-Timer,
3. send an SME “hello” update message (see definition below),

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

4. enter state SME_HELLO_STATE_READY_LISTEN.

Please note that this state does not evaluate any received messages. I.e. SME “hello” messages that are received before or during this state are subject of subsequent states.

Sub-state SME_HELLO_STATE_READY_LISTEN:

In this state, the SME User SHALL evaluate SHIP messages received from the communication partner.

Only SME “hello” messages are considered here. The following rules apply:

1. If the received message has sub-element “phase” set to “ready”: Enter state “HELLO_OK”.
2. If the received message has sub-element “phase” set to “pending” and NO sub-element “prolongationRequest” is set: No specific action is required here (i.e. ignore the message).
3. If the received message has sub-element “phase” set to “pending” and sub-element “prolongationRequest” is set to “true”:
    a. Execute the common procedure to decide an incoming prolongation request.
    b. Execute the common procedure to send an SME “hello” update message.
4. If the received message has sub-element “phase” set to “aborted”: Execute the common “abort” procedure.

If a received message is not an SME “hello” message while in this state, the SME User SHALL execute the common “abort” procedure.

Sub-state SME_HELLO_STATE_READY_TIMEOUT:

This state SHALL be entered if Wait-For-Ready-Timer expired. The SME User SHALL execute the common “abort” procedure.

Sub-state SME_HELLO_STATE_PENDING_INIT:

This is the first state an SME User SHALL enter if it is in the basic state „PENDING“. In this state, it SHALL

1. initialise Wait-For-Ready-Timer to the default value and start the timer,
2. deactivate Send-Prolongation-Request-Timer and Prolongation-Request-Reply-Timer,
3. send an SME “hello” update message (see definition below),
4. enter state SME_HELLO_STATE_PENDING_LISTEN.

Sub-state SME_HELLO_STATE_PENDING_LISTEN:

In this state, the SME User SHALL evaluate SHIP messages received from the communication partner.

Only SME “hello” messages are considered here. The following rules apply:

1. If the received message has sub-element “phase” set to “ready” and NO sub-element “waiting”: Execute the common “abort” procedure.
2. If the received message has sub-element “phase” set to “ready” and sub-element “waiting” is set:
    a. Deactivate Wait-For-Ready-Timer and Prolongation-Request-Reply-Timer.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

b. If the received sub-element “waiting” is greater than or equal to T_hello_prolong_thr_inc: Initialize Send-Prolongation-Request-Timer to a new value as described below and (re-)start the timer. Otherwise (i.e. the received sub-element “waiting” is less than T_hello_prolong_thr_inc): Deactivate Send-Prolongation-Request-Timer.

3. If the received message has sub-element “phase” set to “pending” and sub-element “waiting” is set and NO sub-element “prolongationRequest” is set:

   a. Deactivate Prolongation-Request-Reply-Timer.

   b. If the received sub-element “waiting” is greater than or equal to T_hello_prolong_thr_inc: Initialize Send-Prolongation-Request-Timer to a new value as described below and (re-)start the timer. Otherwise (i.e. the received sub-element “waiting” is less than T_hello_prolong_thr_inc): Deactivate Send-Prolongation-Request-Timer.

4. If the received message has sub-element “phase” set to “pending” and NO sub-element “waiting” and sub-element “prolongationRequest” is set to “true”:

   a. Execute the common procedure to decide an incoming prolongation request.

   b. Execute the common procedure to send an SME “hello” update message.

5. If the received message has sub-element “phase” set to “aborted”: Execute the common “abort” procedure.

6. If the received message does not match any of the aforementioned schemes: Execute the common “abort” procedure.

In addition, the following rules apply:

1. If a received message is not an SME “hello” message while in this state, the SME User SHALL execute the common “abort” procedure.

2. If an SME User finally decides to not trust the communication partner, the SME User SHALL execute the common “abort” procedure.

The following rules SHALL be applied to calculate a new value for Send-Prolongation-Request-Timer:

1. The new value SHALL be by T_hello_prolong_waiting_gap lower than the value from the received sub-element “waiting”.

2. Under normal operation, the value calculated above should be positive. However, in case the result is less than T_hello_prolong_min, the SME User SHALL disable the Send-Prolongation-Request-Timer.

Sub-state SME_HELLO_STATE_PENDING_TIMEOUT:

This state SHALL be entered if any of the timers expired:

1. If Wait-For-Ready-Timer expired: The SME User SHALL execute the common “abort” procedure.

2. If Send-Prolongation-Request-Timer expired:

   a. The SME User SHALL send an SME “hello” message with the following content:

      i. Sub-element “phase” set to “pending”.

      ii. Sub-element “prolongationRequest” set to “true”.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

iii. No further sub-element shall be set.

b. Initialize Prolongation-Request-Reply-Timer to the value of the last received sub-element “waiting” of the communication partner. If no sub-element “waiting” was received so far, the 1.1-fold of the current value of Wait-For-Ready-Timer SHALL be used as the initialization value.

c. Start Prolongation-Request-Reply-Timer.

d. Return to the previous state.

3. If Prolongation-Request-Reply-Timer expired: The SME User SHALL execute the common “abort” procedure.

Switching Between Basic States “READY” and “PENDING”:

1. It is NOT permitted to switch from basic state “READY” and its sub-states to basic state “PENDING” and any of its sub-states.

2. If an SME User switches from basic state “PENDING” to “READY”, it SHALL

a. deactivate Send-Prolongation-Request-Timer and Prolongation-Request-Reply-Timer,

b. send an SME “hello” update message (with its new state “READY”),

c. enter state

i. EITHER “HELLO_OK” (only if one of the previously received SME “hello” messages had sub-element “phase” set to “ready”)

ii. OR SME_HELLO_STATE_READY_LISTEN otherwise.

State HELLO_OK:

All timers of connection state “Hello” can be deactivated. The SME User SHALL continue with “Connection state Protocol handshake” (see 13.4.4.2).

Common “abort” procedure:

This procedure SHALL be executed where referenced. The SME User SHALL

1. deactivate all SME specific timers of connection state “Hello”,

2. send an SME “hello” message with sub-element “phase” set to “aborted” (further sub-elements SHALL NOT be set) if the connection is not already closed,

3. close the connection (if the connection is not already closed).

Common Procedure for Sending an SME “hello” Update Message:

This procedure SHALL be executed where referenced. The SME User SHALL send an SME “hello” message with the following content:

1. Sub-element “phase” set to

a. “ready” in case of the basic state “READY”,

b. or “pending” in case of the basic state “PENDING”.

2. Sub-element “waiting” set to the current value of Wait-For-Ready-Timer if Wait-For-Ready-Timer is active. This sub-element SHALL NOT be set if the timer is not active.

Further sub-elements SHALL NOT be set.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Common Procedure to Decide an Incoming Prolongation Request:

This procedure SHALL be executed where referenced.

1. If an SME User accepts the prolongation request: It SHALL increase its Wait-For-Ready-Timer by T_hello_inc.
2. Otherwise: No specific action required.

Further rules apply:

1. An SME User SHALL accept at least two prolongation requests.

Implementation Advice:

Several rules and procedures described above include instructions on the report of a state or response. The timers of this section SHALL NOT be used to delay such reports more than necessary.

Example Sequence Diagrams:

Example sequence diagrams for connection state “Hello” are shown in the subsequent figures:

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

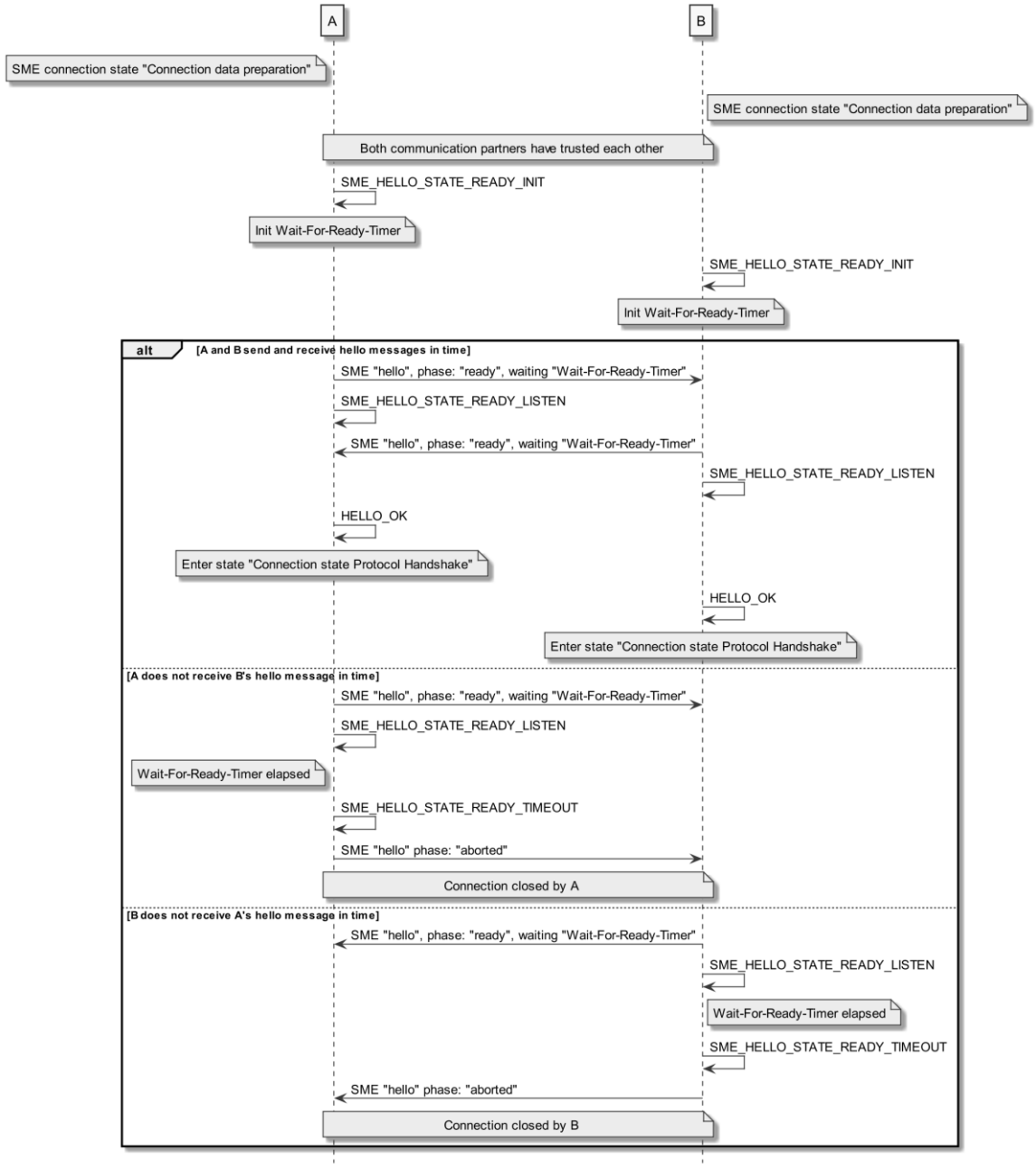


Figure 10: Connection State "Hello" Sequence Example Without Prolongation Request: "A" and "B" already trust each other; "B" is slower/delayed.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

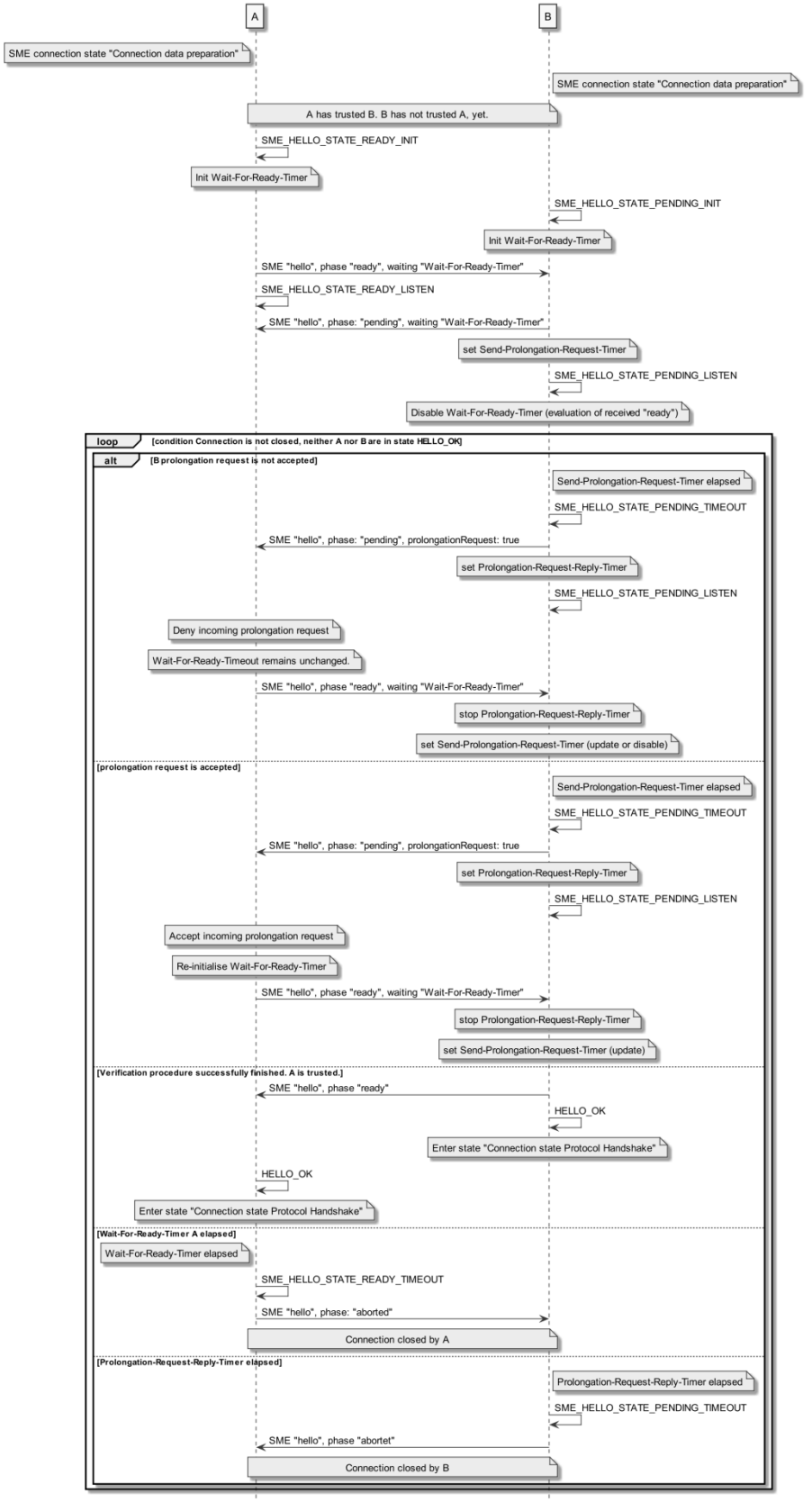


Figure 11: Connection State "Hello" Sequence Example With Prolongation Request.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

13.4.4.2 Connection State “Protocol handshake”

13.4.4.2.1 Basic Definitions

In this state, the communication partners agree on the further SHIP protocol version and on the message format to continue with.

This state uses an SME “protocol handshake” message and a dedicated error message.

SME “Protocol Handshake” Message:

The SME “protocol handshake” message is defined as follows:

```
MessageType = %x01 ; control
MessageValue = SmeProtocolHandshakeValue
SmeProtocolHandshakeValue = *OCTET
```

The content of SmeProtocolHandshakeValue is defined as follows: The structure is defined by the SHIP root tag “messageProtocolHandshake” (including the root tag “messageProtocolHandshake”) of the XSD “SHIP_TS_TransferProtocol.xsd”. The default structure extensibility applies to this structure. The format of this structure MUST be JSON-UTF8.

Element name	**Mandatory/ Optional/ Not Valid (NV)**	**Brief explanation**
messageProtocolHandshake.handshakeType	M	The kind of the handshake information (enumeration: announceMax, select).
messageProtocolHandshake.version	M	Parent element of SHIP specification version information.
messageProtocolHandshake.version.major	M	Version information: Major version part.
messageProtocolHandshake.version.minor	M	Version information: Minor version part.
messageProtocolHandshake.formats	M	Protocol format(s).
List of “format” (1..unbounded)	M	In general, the subsequent child element “format” SHALL be present at least one time and CAN be present more than one time. However, the number of permitted occurrences finally depends on the phase of the protocol handshake. See definitions in the text for details! The “format” instances of the list SHALL have a

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

		unique value, i.e. no two “format” values may be identical.
messageProtocolHandshake.formats.format	M	Protocol format. See text for permitted values.

Table 15: Structure of SmeProtocolHandshakeValue of SME "Protocol Handshake" Message.

Permitted values for the child element “format” are “JSON-UTF8” and “JSON-UTF16” (without the quotation marks), but only “JSON-UTF8” is REQUIRED to be supported (see also 13.4.4.2.2). Other values are reserved for future use. An empty string is NOT a permitted value.

SME “Protocol Handshake Error” Message:

The SME “protocol handshake error” message is defined as follows:

```
MessageType = %x01 ; control

MessageValue = SmeProtocolHandshakeErrorValue

SmeProtocolHandshakeErrorValue = *OCTET
```

The content of SmeProtocolHandshakeErrorValue is defined as follows: The structure is defined by the SHIP root tag “messageProtocolHandshakeError” (including the root tag “messageProtocolHandshakeError”) of the XSD “SHIP_TS_TransferProtocol.xsd”. The default structure extensibility applies to this structure. The format of this structure MUST be JSON-UTF8.

Element name	**Mandatory/ Optional/ Not Valid (NV)**	**Brief explanation**
messageProtocolHandshakeError.error	M	Error number.

Table 16: Structure of SmeProtocolHandshakeErrorValue of SME "Protocol Handshake Error" Message.

13.4.4.2.2 Compatibility Aspects

In this version of the specification, the exchange of SME “protocol handshake” messages is exclusively executed with JSON-UTF8 as described above.

Each communication partner MUST support each SHIP specification version from “1.0” up to and including their own maximum supported SHIP specification version.

Each communication partner MUST support the format JSON-UTF8 (see also 13.4.4.2.1).

Remark (informative): Subsequent versions of the SHIP specification may define a Connection Mode Initialization that permits a different protocol handshake. Even the omission of a handshake might be defined dependent on the circumstances. However, a fall-back mechanism to version 1.0 of the SHIP specification must be defined and preserved.

13.4.4.2.3 Protocol Handshake Process

Connections Roles:

The concept requires the knowledge of the SME connection role (client, server).

Timer overview:

The following timer is defined:

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

1. Wait-Timer

   Default value: 10 seconds.

   Purpose: The communication partner must provide the required protocol handshake information before the timer expires.

State SME_PROT_H_STATE_SERVER_INIT:

This is the first state a server SME User SHALL enter. In this state, it SHALL

1. initialize Wait-Timer to the default value and start the timer,
2. enter state SME_PROT_H_STATE_SERVER_LISTEN_PROPOSAL.

State SME_PROT_H_STATE_CLIENT_INIT:

This is the first state a client SME User SHALL enter. In this state, it SHALL

1. send an SME “protocol handshake” message with the following content:
   a. Sub-element “handshakeType” set to “announceMax”.
   b. Sub-element version (and its children) set to the maximum supported SHIP specification version: In this version of the specification, this means “major” MUST be set to “1” and “minor” MUST be set to “0”.
   c. Sub-element “formats” set to all format values supported by the client.
2. initialise Wait-Timer to the default value and start the timer,
3. enter state SME_PROT_H_STATE_CLIENT_LISTEN_CHOICE.

State SME_PROT_H_STATE_SERVER_LISTEN_PROPOSAL:

In this state, a server SME User evaluates received messages from the client:

1. If the received message is a valid SME “protocol handshake” message: The server SME User SHALL
   a. deactivate Wait-Timer,
   b. select the maximum supported SHIP specification version supported by both communication partners,
   c. select a single format supported by both communication partners,
   d. send an SME “protocol handshake” message with the following content:
      i. Sub-element “handshakeType” set to “select”.
      ii. Sub-element version (and its children) set to the selected SHIP specification version.
      iii. Sub-element “formats” set to the selected format.
   e. re-initialize Wait-Timer to the default value and start the timer,
   f. enter state SME_PROT_H_STATE_SERVER_LISTEN_CONFIRM.
2. Otherwise: Execute the common “abort” procedure with error type “unexpected message”.

State SME_PROT_H_STATE_CLIENT_LISTEN_CHOICE:

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

In this state, a client SME User evaluates received messages from the server to analyse the server's selection:

1. If the received message is a valid SME "protocol handshake" message: The client SME User SHALL

    a. deactivate Wait-Timer,

    b. verify if the received sub-element "handshakeType" is set to "select" and – in case the verification succeeded – continue with the next step,

    c. verify if the received sub-element "version" matches the client's capability and – in case the verification succeeded – continue with the next step,

    d. verify if the received sub-element "formats" contains a single format and matches the client's capability and – in case the verifications succeeded – continue with the next step,

    e. send the received SME "protocol handshake" message back to the server (this denotes the confirmation of the server's choice),

    f. enter state SME_PROT_H_STATE_CLIENT_OK.

    If any of the aforementioned verifications failed, the node SHALL execute the common "abort" procedure with error type "selection mismatch".

2. Otherwise: Execute the common "abort" procedure with error type "unexpected message".

State SME_PROT_H_STATE_SERVER_LISTEN_CONFIRM:

In this state, a server SME User evaluates received messages from the client:

1. If the received message is a valid SME "protocol handshake" message: The server SME User SHALL

    a. deactivate Wait-Timer,

    b. verify if the received message is identical to the message the server previously (in state SME_PROT_H_STATE_SERVER_LISTEN_PROPOSAL) submitted to the client and – in case the verification succeeded – continue with the next step,

    c. enter state SME_PROT_H_STATE_SERVER_OK.

    If any of the aforementioned verifications failed, the SME User SHALL execute the common "abort" procedure with error type "selection mismatch".

2. Otherwise: Execute the common "abort" procedure with error type "unexpected message".

Sub-state SME_PROT_H_STATE_TIMEOUT:

This state SHALL be entered if Wait-Timer expired. The SME User SHALL execute the common "abort" procedure with error type "timeout".

States SME_PROT_H_STATE_CLIENT_OK, State SME_PROT_H_STATE_SERVER_OK:

As soon as an SME User enters this state, it SHALL switch to the selected SHIP specification version and selected format. Then, it SHALL proceed with "Connection state PIN verification".

Example (informative): Before this state is reached, all messages are exchanged with the format JSON-UTF8 (in SHIP specification version 1.0). For this example, we assume there is a new SHIP specification version 2.1 available and a binary format "ASN.1-PER" and both SME Users agreed on this version and format. This means all messages MUST follow the protocol and

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

format requirements as soon as the "…_OK" state is reached. This may include new values for MessageType and non-textual (but compact, i.e. efficient) content of MessageValue.

Common "abort" procedure:

This procedure SHALL be executed where referenced. The SME User SHALL

1. deactivate all SME specific timers of connection state "Protocol handshake",
2. send an SME "protocol handshake error" message with sub-element "error" set to the proper value (see Table 17),
3. close the connection.

Value	Error type
0	RFU
1	Timeout
2	unexpected message
3	selection mismatch
4-255	RFU

Table 17: Values of Sub-element "error" of messageProtocolHandshakeError.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

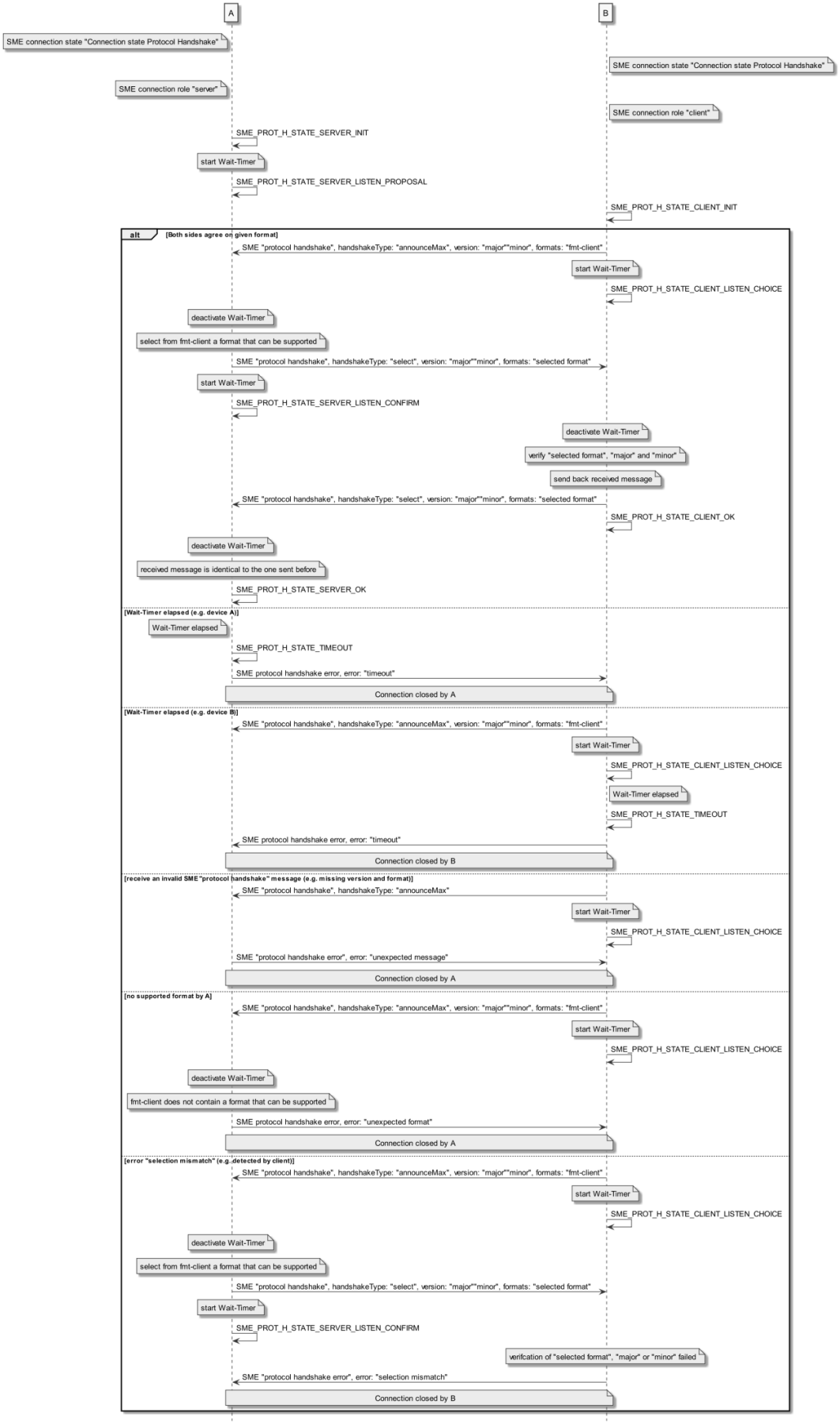


Figure 12: Connection State "Protocol Handshake" Message Sequence Example

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

13.4.4.3 Connection State “PIN Verification”

13.4.4.3.1 Introduction (Informative)

The exchange of a PIN is a common procedure to gain a certain (minimum) trust level required to permit the exchange of sensitive data between authorized communication partners only. In general, a PIN based authorization can be considered a more secure methodology than a simple “push button” based method with “auto_accept”. Consequently, successful PIN exchange might be mandatory for certain data that is “too sensitive” to be exchanged between communication partners that are introduced by “push button/auto_accept” only. See chapter 12 for more details.

13.4.4.3.2 Basic Definitions

In this state, the communication partners exchange information on their PIN requirements.

This state uses SME messages for the PIN state, PIN input, and a dedicated error message. Broadly speaking, “PIN state” is used to inform the communication partner whether a PIN is expected. The message “PIN input” serves for the transport of a PIN to the owner of the PIN. The error message indicates if a PIN is wrong.

SME “PIN state” Message:

The SME “PIN state” message is defined as follows:

```
MessageType = %x01 ; control

MessageValue = SmeConnectionPinStateValue

SmeConnectionPinStateValue = *OCTET
```

The content of SmeConnectionPinStateValue is defined as follows: The structure is defined by the SHIP root tag “connectionPinState” (including the root tag “connectionPinState”) of the XSD “SHIP_TS_TransferProtocol.xsd”. The default structure extensibility applies to this structure. The format of this structure MUST be the format agreed with the protocol handshake (see 13.4.4.2).

Element name	Mandatory/ Optional/ Not Valid (NV)	Brief explanation
connectionPinState.pinState	M	The originator’s PIN state (in relation to the recipient). See 13.4.4.3.5.1.
connectionPinState.inputPermission	O	Information whether PIN receipt is currently permitted or not. See 13.4.4.3.5.1.

Table 18: Structure of SmeConnectionPinStateValue of SME "Pin state" message.

SME “PIN input” Message:

The SME “PIN input” message is defined as follows:

```
MessageType = %x01 ; control

MessageValue = SmeConnectionPinInputValue

SmeConnectionPinInputValue = *OCTET
```

The content of SmeConnectionPinInputValue is defined as follows: The structure is defined by the SHIP root tag “connectionPinInput” (including the root tag “connectionPinInput”) of the XSD

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

"SHIP_TS_TransferProtocol.xsd". The default structure extensibility applies to this structure. The format of this structure MUST be the format agreed with the protocol handshake (see 13.4.4.2).

Element name	Mandatory/ Optional/ Not Valid (NV)	Brief explanation
connectionPinInput.pin	M	The receiver's PIN.

Table 19: Structure of SmeConnectionPinInputValue of SME "Pin input" message.

The sub-element "pin" MUST be set and interpreted as follows: The value range of the PIN is defined by section 12.5. The format of the PIN within sub-element "pin" of "PIN input" message is the same as the written format of section 12.5, but without any separators. Thus, the sub-element "pin" is a string of 8-16 contiguous hexadecimal characters (as specified in section 12.5), i.e. the decimal characters '0' to '9' and the Latin characters 'a' to 'f' and 'A' to 'F'. The verification of a PIN (i.e. the value of element "pin") SHALL NOT be case-sensitive.

Examples (informative): If the written PIN (for the user) is "12AB C34D" (without quotation marks), the following strings show valid content for the element "pin": "12abc34d", "12ABc34d", "12ABC34D", etc. (without quotation marks; examples not exhaustive).

SME "PIN error" Message:

The SME "PIN error" message is defined as follows:

```
MessageType = %x01 ; control

MessageValue = SmeConnectionPinErrorValue

SmeConnectionPinErrorValue = *OCTET
```

The content of SmeConnectionPinErrorValue is defined as follows: The structure is defined by the SHIP root tag "connectionPinError" (including the root tag "connectionPinError") of the XSD "SHIP_TS_TransferProtocol.xsd". The default structure extensibility applies to this structure. The format of this structure MUST be the format agreed with the protocol handshake (see 13.4.4.2).

Element name	Mandatory/ Optional/ Not Valid (NV)	Brief explanation
connectionPinError.error	M	Error number.

Table 20: Structure of SmeConnectionPinErrorValue of SME "Pin error" message.

13.4.4.3.3 Basic Rules

The device that owns (requires) a device PIN MUST NOT communicate this PIN to another device. Only the other direction is allowed. These rules can be expressed briefly as follows, where node "A" owns a "PIN A" and another node "X" is the communication partner:

1. NOT ALLOWED: Send "PIN A" from left to right: Node A -----> Node X
2. ALLOWED: Send "PIN A" from right to left: Node A <----- Node X

Common security related rules on PINs (examples: which kind of device needs to have a PIN and request it from the communication partner; how is a PIN made available to a user; which

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

conditions are defined to prevent “too simple PIN values”; how does a PIN value contribute to a trust level) are subject of chapter 12.

13.4.4.3.4 Protection Against Brute Force Attempts

Every verified and invalid PIN received from the communication partner is counted. If a node verifies a received PIN and declares it as invalid, it SHALL proceed as specified for the state SME_PIN_STATE_CHECK_ERROR AND impose a penalty to the communication partner according to the following rules:

1. If the number of counted invalid PINs is less than three, NO penalty is required.

2. If the number of counted invalid PINs ranges from three to five, the node SHALL apply a penalty as follows: The node SHALL enter the state SME_PIN_STATE_CHECK_BUSY_WAIT for a period of at least 10 seconds. This period SHOULD exceed 15 seconds only in case of increased security requirements.

3. If the number of counted invalid PINs is greater than five, the node SHALL apply a penalty as follows: The node SHALL enter the state SME_PIN_STATE_CHECK_BUSY_WAIT for a period of at least 60 seconds. This period SHOULD exceed 90 seconds only in case of increased security requirements.

In addition, the node SHALL implement countermeasures against attempts to bypass the aforementioned penalties. Among others, disconnecting the node or switching it off (regularly or suddenly, e.g. through power loss) SHALL NOT disable or weaken the penalties towards the communication partner that sent the invalid PIN. This requirement holds regardless of the number of communication partners a node is capable of distinguishing and (potentially) storing.

Remark (informative): The aforementioned countermeasures may lead to modified PIN requirements (see 13.4.4.3.5.1) after re-powering/re-connection. E.g. if a penalty towards a communication partner was not completed before the disconnection, the “inputPermission” SHOULD be “busy” towards the re-connected communication partner until the penalty has been completed. In fact, if a node is not capable of persistently storing all communication partners with unfinished penalty, it may impose a general penalty (i.e. regardless of the communication partner/connection that is currently established) until the general penalty was completed.

13.4.4.3.5 Process Details

13.4.4.3.5.1 PIN Requirement - Communicated PIN States

The sub-element connectionPinState.pinState of the SME “PIN state” message conveys the PIN requirement towards the communication partner, i.e. the communicated PIN state. Permitted values are:

1. **required**

   The node requires to receive its own valid PIN from the communication partner. The next state “Connection data exchange” will not be reached until a received PIN was verified successfully.

   Note: Setting "pinState" to "required" should only be done for certain cases! In general, a SHIP node cannot know whether the other SHIP node has a user interface or equivalent possibility for PIN input. If a manufacturer of a SHIP device decides to set "pinState" to "required", the manufacturer should also provide a SHIP-based commissioning tool with PIN input.

2. **optional**

   The node does not require its own valid PIN from the communication partner, but restricts data exchange. It is possible to proceed with the next state “Connection data exchange” without the correct PIN; however, the node limits the data exchange to only those data that do not require the PIN. As soon as the communication partner submits the valid PIN, the node grants access to all data.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Note: This means that the communication partner is not forced to submit a PIN. I.e. the node would keep its “pinState” value to “optional” and would communicate according to state “Connection data exchange” (with reduced data) – but in parallel it would continue listening for potential PIN messages from the communication partner for a “late release” of the data exchange restrictions.

3. **pinOk**

   The node already received its own valid PIN from the communication partner and grants unrestricted data exchange. It is possible to proceed with the next state “Connection data exchange” immediately.

4. **none**

   The node does not have an own PIN and grants unrestricted data exchange. It is possible to proceed with the next state “Connection data exchange” immediately.

Note: The above mentioned state “Connection data exchange” requires further conditions to be enabled. This is described in detail in the subsequent sections.

The sub-element connectionPinState.inputPermission of the SME “PIN state” message expresses whether the owner of the PIN currently accepts an SME “PIN input” message for verification or not. Permitted values are:

1. **busy**

   The node does currently not accept an SME “PIN input” message for verification.

2. **ok**

   The node currently accepts an SME “PIN input” message for verification.

The following dependencies between “pinState” and “inputPermission” are defined:

1. If the value of “pinState” is “pinOk” or “none”, the sub-element “inputPermission” MUST NOT be present. This case means the node does not accept an SME “PIN input” message.

2. If the value of “pinState” is “required” or “optional”, the sub-element “inputPermission” MUST be present. The value of “inputPermission” MUST be “busy” as long as a penalty towards the communication partner is in place (see 13.4.4.3.4); otherwise, it MUST be “ok”.

13.4.4.3.5.2 Process States

Broadly speaking, a node needs to

1. report its PIN requirement to a communication partner,

2. verify received PINs (provided a PIN is required or at least optional),

3. await the communication partner’s PIN requirement,

4. send a PIN to the communication partner (provided it is required or at least optional and the node wants to obtain unrestricted data exchange).

The first two items belong to the major state SME_PIN_STATE_CHECK whereas the remaining items belong to the major state SME_PIN_STATE_ASK. These major states are independent from each other and SHALL also be executed in parallel. If connection state “PIN verification” is entered, the first sub-state of SME_PIN_STATE_CHECK to execute is SME_PIN_STATE_CHECK_INIT and the first sub-state of SME_PIN_STATE_ASK to execute is SME_PIN_STATE_ASK_INIT.

Sub-state SME_PIN_STATE_CHECK_INIT:

In this state, the node SHALL perform the following steps:

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

1. It SHALL execute the common procedure to send the PIN requirement with the current requirements it has towards the communication partner.

2. If the node's "pinState" is "required":

   a. State "Connection data exchange" MUST be disabled (i.e. this state must not be executed).

   b. State SME_PIN_STATE_CHECK_BUSY_WAIT SHALL be entered if the sub-element "inputPermission" is set to "busy", otherwise state SME_PIN_STATE_CHECK_LISTEN SHALL be entered.

3. If the node's "pinState" is "optional":

   a. The common procedure to enable the state "Connection data exchange" SHALL be executed.

   b. State SME_PIN_STATE_CHECK_BUSY_WAIT SHALL be entered if the sub-element "inputPermission" is set to "busy", otherwise state SME_PIN_STATE_CHECK_LISTEN SHALL be entered.

4. If the node's "pinState" is "pinOk" or "none", the common procedure to enable the state "Connection data exchange" SHALL be executed and state SME_PIN_STATE_CHECK_OK SHALL be entered.

Sub-state SME_PIN_STATE_CHECK_LISTEN:

In this state, the SME User SHALL evaluate SHIP messages received from the communication partner. When this state is entered, the sub-element "inputPermission" MUST already be set to "ok".

Only SME "PIN verification" messages are considered here. The following rules apply:

1. If the received PIN matches the node's PIN:

   a. Set the node's "pinState" to "pinOk" and remove (disable) the sub-element "inputPermission".

   b. Execute the common procedure to send the PIN requirement.

   c. Execute the common procedure to enable the state "Connection data exchange".

   d. Enter the state SME_PIN_STATE_CHECK_OK.

2. If the received PIN DOES NOT match the node's PIN:

   a. Enter state SME_PIN_STATE_CHECK_ERROR.

Sub-state SME_PIN_STATE_CHECK_ERROR:

In this state, the SME User informs the communication partner that a wrong PIN has been received. The following steps SHALL be performed:

1. Increase the number of counted invalid PINs.

2. Execute the common procedure to send an SME "PIN error" message with the error code for "wrong PIN".

3. If the number of counted invalid PINs DOES NOT require imposing a penalty according to the rules of section 13.4.4.3.4: Enter state SME_PIN_STATE_CHECK_LISTEN.

4. If the number of counted invalid PINs DOES require imposing a penalty according to the rules of section 13.4.4.3.4: Enter state SME_PIN_STATE_CHECK_BUSY_INIT.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Sub-state SME_PIN_STATE_CHECK_BUSY_INIT:

In this state, the SME User prepares a penalty (see 13.4.4.3.4). The following steps SHALL be performed:

1. Set the node's sub-element "inputPermission" to "busy".
2. Execute the common procedure to send the PIN requirement.
3. Enter state SME_PIN_STATE_CHECK_BUSY_WAIT.

Sub-state SME_PIN_STATE_CHECK_BUSY_WAIT:

In this state, the SME User shall impose a penalty according to 13.4.4.3.4. This means that the SME User remains in this state until the end of the penalty's duration. When this state is entered, the sub-element "inputPermission" MUST already be set to "busy". The following steps SHALL be performed:

1. As long as the duration of the penalty has not expired:
    a. For every received SME "PIN input" message, execute the common procedure to send the PIN requirement.
2. As soon as the duration of the penalty expires:
    a. Set the node's sub-element "inputPermission" to "ok".
    b. Execute the common procedure to send the PIN requirement.
    c. Enter state SME_PIN_STATE_CHECK_LISTEN.

Note: In this state, received PINs are NOT evaluated and are NOT counted.

Sub-state SME_PIN_STATE_CHECK_OK:

The SME User SHALL silently discard any SME "PIN input" message.

Note: The state branch "SME_PIN_STATE_CHECK" ends here, i.e. no further action is required. Whether a next state is enabled is not determined here. Instead, this is determined where the common procedure to enable the state "Connection data exchange" is referenced.

Sub-state SME_PIN_STATE_ASK_INIT:

In this state, the SME User SHALL wait for the receipt of an SME "PIN state" message before any other action of this state is performed. Until then, no other message of connection state "PIN verification" SHALL be evaluated. Afterwards, the SME User SHALL enter state SME_PIN_STATE_ASK_PROCESS.

However, the SME User SHALL close the connection if this state (SME_PIN_STATE_ASK_INIT) lasts more than 10 seconds.

Sub-state SME_PIN_STATE_ASK_PROCESS:

In this state, the SME User evaluates and processes received messages according to the following rules:

1. The SME User SHALL close the connection if any of the rules of section 13.4.4.3.5.1 on the sub-elements of a received message are not fulfilled.
2. A received SME "PIN error" message with the sub-element "error" set to the value for "wrong PIN" SHALL be interpreted by the SME User as follows: An SME "PIN input" message previously submitted to the communication partner contained the wrong PIN. The SME User SHOULD wait for a new SME "PIN state" message.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

3. For every received SME “PIN state” message with sub-element “pinState” set to a value that is NOT “required”, the SME User SHALL execute the common procedure to enable the state “Connection data exchange”.

4. A received SME “PIN state” message with the sub-element “inputPermission” present and set to “busy” SHALL be interpreted by the SME User as follows: The communication partner is currently not ready to evaluate any SME “PIN input” message. The node SHOULD wait until the communication partner indicates being ready for an SME “PIN input” message. However, the node CAN send an SME “PIN input” message before the communication partner indicates being ready for this message.

   Remark (informative): The latter case (sending a new PIN before the communication partner indicates being ready) only makes sense to get a confirmation of the state after an “unexpectedly long” period of the assumed penalty.

5. A received SME “PIN state” message with the sub-element “inputPermission” present and set to “ok” SHALL be interpreted by the SME User as follows: The communication partner is currently ready to evaluate an SME “PIN state” message. The node SHALL also evaluate sub-element “pinState” and decide whether to send an SME “PIN input” message or not.

6. A received SME “PIN state” message with the sub-element “pinState” set to “required” or “optional” SHALL be interpreted and processed by the SME User as follows: The node SHALL also evaluate sub-element “inputPermission” and decide whether to send an SME “PIN input” message or not.

7. A received SME “PIN state” message with the sub-element “pinState” set to “pinOk” or “none” SHALL be interpreted and processed by the SME User as follows: The node SHALL enter the state SME_PIN_STATE_ASK_OK.

8. If a node needs to decide whether to send an SME “PIN input” message or not, the following rules apply:

   a. If a received “pinState” value is set to “required”, the SME User SHALL

      i. EITHER execute the common procedure to send an SME “Pin input” message (provided that sub-element “inputPermission” is set to “ok” and the SME User has a PIN to send)

      ii. OR close the connection.

   b. If a received “pinState” value is set to “optional”, the SME User SHALL

      i. EITHER enter state SME_PIN_STATE_ASK_RESTRICTED_OK (i.e. the SME User does not require unrestricted data exchange with the communication partner)

      ii. OR execute the common procedure to send an SME “Pin input” message (provided that sub-element “inputPermission” is set to “ok” and the SME User has a PIN to send)

      iii. OR close the connection.

9. If a received SME “PIN state” message had “pinState” value set to “required” or “optional” and the SME User sent an SME “PIN input” message, the SME User SHALL wait for a new SME “PIN state” message for at least 30 seconds and at most 120 seconds before deciding to continue in this state or close the connection.

   Note: This rule is independent from decisions on a retry to send a PIN. It just focuses on the lack of a “PIN state” update (esp. with “inputPermission” either “busy” or “ok”) from the communication partner.

10. This state remains enabled unless stated otherwise. This also means that the SME User will continue listening for incoming messages as described above.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Sub-state SME_PIN_STATE_ASK_RESTRICTED_OK:

The SME User SHALL silently discard any SME “PIN error” message. It SHALL keep a “PIN state” message (only the latest message is required; this message SHALL NOT be kept it in case a connection is closed).

Note: The state branch “SME_PIN_STATE_ASK” ends here, i.e. no further action is required. Whether a next state is enabled is not determined here. Instead, this is determined where the common procedure to enable the state “Connection data exchange” is referenced.

However, if an SME User wants to submit a PIN on a later occasion, it can take the last SME “PIN state” message and enter state SME_PIN_STATE_ASK_PROCESS.

Sub-state SME_PIN_STATE_ASK_OK:

The SME User SHALL silently discard any SME “PIN error” or “PIN state” message.

Note: The state branch “SME_PIN_STATE_ASK” ends here, i.e. no further action is required. Whether a next state is enabled is not determined here. Instead, this is determined where the common procedure to enable the state “Connection data exchange” is referenced.

Common Procedure to Send the PIN Requirement:

This procedure SHALL be executed where referenced.

The SME User SHALL send an SME “PIN state” message with sub-element “pinState” set according to the node’s PIN requirement towards the communication partner (see 13.4.4.3.5.1). The sub-element “inputPermission” SHALL be omitted or set as required and according to the rules expressed in 13.4.4.3.5.1.

Common Procedure to Send an SME “PIN error” Message:

This procedure SHALL be executed where referenced.

The SME User SHALL send an SME “PIN error” message with sub-element “error” set to the appropriate value (see Table 21).

Value	Error type
0	RFU
1	wrong PIN
4-255	RFU

Table 21: Values of Sub-element “error” of connectionPinError.

Common procedure to Send an SME “PIN input” message:

This procedure SHALL be executed where referenced.

The SME User SHALL send an SME “PIN input” message with sub-element “pin” set to the value required by the communication partner.

Common procedure to Enable the State “Connection data exchange”:

This procedure SHALL be executed where referenced.

The SME User SHALL enable the state “Connection data exchange” if and only if all of the following rules are fulfilled:

1. The node’s own PIN requirement element “pinState” towards the communication partner is NOT “required”.

2. The communication partner’s PIN requirement is available AND its element “pinState” is available AND is NOT “required”.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Enabling the state “Connection data exchange” means this state shall be executed regardless of any (parallel) states of “Connection state PIN verification”. Depending on the element “pinState”, the SME User SHALL also adjust the restriction of data exchange as mentioned in 13.4.4.3.5.1 accordingly.

A brief overview of the PIN verification procedure is given in the following sequence diagrams.

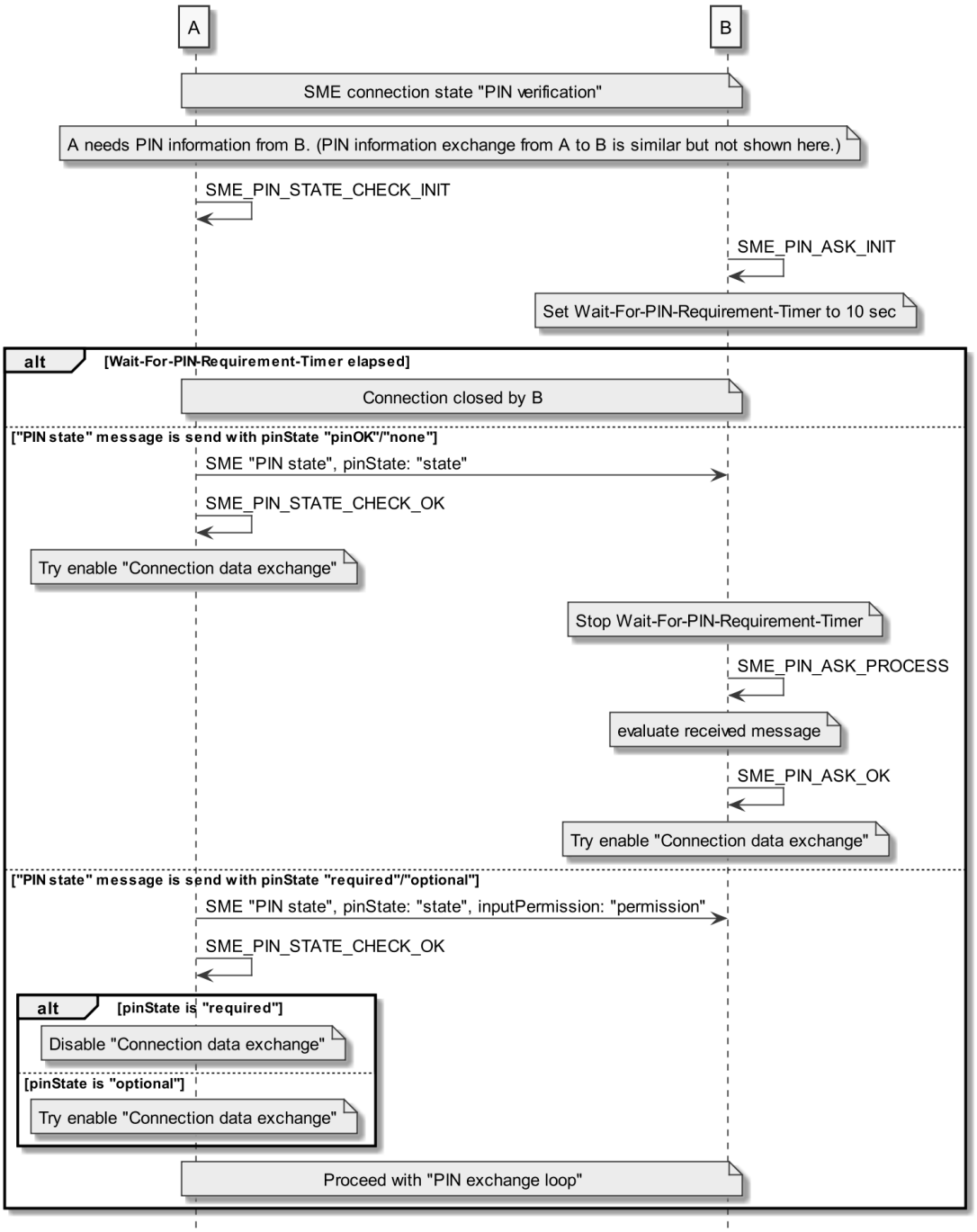


Figure 13: Connection State "PIN verification” Message Sequence Example (Begin)

13.4.5 Connection Data Exchange

13.4.5.1 General Rules

Specification Versions and "Base Specification"

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

A specification may exist in multiple versions. The first version is called "base specification" and denotes the unique origin of the other versions.

General Compatibility Rules for a Given Specification Sequence

Subsequently, the term "content" denotes a data instance that matches a given specification. The compatibility of specifications is immediately related to the question how "content" of different specification versions needs to be processed. Specification versions are compatible with each other if the following rules apply:

1. The base specification must provide rules for forward compatibility. This means it must define how any succeeding version can define additional content without breaking compatibility with existing implementations: An implementation that is based on the base specification must be able to gracefully accept and process "content" that is based on a compatible successor of the base specification. The processing of "content" SHALL include at least the parts that are already defined in the base specification. It MAY well skip the unknown parts.

2. Each compatible successor of the base specification must provide rules for forward compatibility as well. These rules MUST NOT break the compatibility rules of the base specification.

3. Each version of a specification has zero or one immediate successors. No more immediate successors are permitted.

4. Each version of a specification has zero or one immediate predecessors. No more immediate predecessors are permitted.

5. An implementation that is based on a given version of the specification must be able to gracefully accept and process "content" that is based on a compatible predecessor of the version. The processing of "content" SHALL include the whole content.

13.4.5.2 Message “data”

13.4.5.2.1 Purpose and Structure

The element "data" of the XSD “SHIP_TS_TransferProtocol.xsd” is used to exchange higher level protocol data (e.g. SPINE) between two SHIP nodes. The structure is briefly described in Table 22. Details on the elements are given in subsequent sections.

Element name	**Mandatory/ Optional/ Not Valid (NV)**	**Brief explanation**
data. header	M	See 13.4.5.2.3.
data. header. protocolId	M	Identifies how “payload” MUST be evaluated, see 13.4.5.2.4.
data. payload	M	Contains data of the protocol stated in the element protocolId (see 13.4.5.2.5).
data. extension	O	Parent element for manufacturer specific extensions, see 13.4.5.2.6.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

data. extension. extensionId	O	Identifier for content of elements “binary”, “string”.
data. extension. binary	O	Binary data.
data. extension. string	O	Textual data.

Table 22: Structure of MessageValue of "data" Message.

The complete “data” message is defined as follows:

```
MessageType = %x02 ; data

MessageValue = DataValue

DataValue = *OCTET
```

The content of DataValue is defined as follows: The structure is defined by the SHIP root tag “data” (including the root element “data”). The default structure extensibility applies to this structure. The format of this structure MUST be the format agreed with the protocol handshake (see 13.4.4.2).

13.4.5.2.2 Extensibility Rules

The "default structure extensibility" applies for these parts:

1. The first level of "data".
2. The element data.header (recursive).
3. The first level of "data.extension".

13.4.5.2.3 Element “header”

This element serves as header for the remaining content of element "data". The sender of a message MUST set all information as described in the table and as follows.

13.4.5.2.4 Element “protocolId”

Introduction (Informative)

This element announces how the content of "payload" MUST be evaluated. Within a software implementation, it permits to select a specific parser esp. in case of potentially conflicting specifications. This applies even in case of potential future binary formats.

Rules on protocolId

1. Permitted values for protocolId are defined by the SHIP specification authority only.
2. The value always denotes the "base specification" for the content of "payload", even if the content is based on a newer compatible version of the specification.
3. A recipient that encounters an unknown value of protocolId SHALL silently skip the received message.

Note: Only those specifications can be assigned a value for protocolId that fulfil further requirements demanded by the SHIP specification.

13.4.5.2.5 Element "payload"

This element takes content that MUST match a compatible version of the base specification as determined by the value of protocolId. The content of payload MUST itself permit to evaluate it properly with regards to the specification sequence determined by the value of protocolId.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

The extensibility rule of payload is determined by the authority of the base specification that is given by the value of protocolId. For content defined by the SHIP specification, the rule "default structure extensibility (recursive)" applies.

Remarks (Informative)

1. Typically, this means the content should start with a "unique" root tag (or "unique" type or data identification in case of a binary format). Together with the value of protocolId, it is then possible to identify which compatible definitions of the proper specification sequence apply. This also means that the content of payload and protocolId must be sufficient to identify the definition of the content. I.e. no further information (as from device discovery, e.g.) is required to know which definition applies.

2. Please also note these rules only address the identification of the definition. They do NOT address questions on the purpose of the content (i.e. questions on specific processes or contexts that are related to the specific content).

3. The development of a specification sometimes also includes the definition of new types or data (i.e. content) for a new version of the specification. Nevertheless, only the base specification needs to be referenced in protocolId as new definitions (rather than modifications of existing definitions) just extend a specification. However, such extensions must also be done according to all compatibility rules. This requires special care especially in case of binary formats.

4. The SME Protocol Handshake includes an agreement of the SME message format to be used between two communication partners. This format applies to the SHIP “payload” element and its content as well. However, it does NOT determine which specific types or conversion rules have to be applied for the protocolId specific content of “payload”. To give an example: The handshake may end in JSON-UTF16 as format, for example. In this case, the complete native SHIP MessageValue for “Data” exchange (see section 13.4.5.2.1) must be formatted in JSON-UTF16. This includes the first “data” key and also the “payload” key, among others. The value of “payload” must as well be formatted in JSON-UTF16. However, the SHIP specification DOES NOT rule which JSON type has to be used as value type of “payload”. For an assumed procotolId “abc”, there may be a specific transformation rule to use a JSON string in any case. For an assumed protocolId “def”, it may be a JSON object, e.g. Finally, this means that it is recommended to consider “payload” as an opaque – but well-formatted – container.

13.4.5.2.6 Element "extension"

This element can be used to extend content from payload with manufacturer specific data. The sub-element “extensionId” MAY be used by a manufacturer to identify the kind of content in the (optional) sub-elements “binary” and “string”. However, the use of these elements is manufacturer-specific and not detailed further in this specification.

13.4.6 Access Methods Identification

13.4.6.1 Introduction

This section discusses a possibility for the device that is currently the connection server to fetch information from the current connection client for a potential “reverse re-connection”. The following example shall explain this briefly.

Subsequently, we assume SHIP device “A” got information on how to connect another SHIP device “B” (e.g. from service discovery), but has not had any connection with device “B” before. This means device “A” finally has the IP address and port number of device “B” and establishes a connection for the first time. Device “A” is the connection client and device “B” the connection server. Any intended connection termination processes or sudden interrupts are no problem for a reconnection as long as device “A” initiates the reconnection again. However, there may be cases where it is favoured or even required that device “B” initiates a connection to device “A” under certain circumstances. In such cases, there is a need for device “B” to have proper information on how to find and connect to device “A”. In general, this cannot be derived from device “A”’s socket of the first connection (i.e. where device “A” is a connection client), as server

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

and client sockets typically differ. Furthermore, IP addresses of devices change in many situations.

To overcome this situation, the section describes a method to query device “A” for its access methods.

13.4.6.2 Basic Definitions

In this section, an SME User can request the “access methods” of the communication partner. However, the support of this methodology is not mandatory in all cases. Details on the support are explained in subsequent sections.

The state “Access Methods Identification” can run in parallel to connection data exchange. In fact, this state MUST NOT be entered before connection data exchange is entered (i.e. the “access methods” exchange does NOT apply for earlier states like “hello” or “CMI”, e.g.).

SME “Access methods request” Message:

The SME “Access methods request” message is defined as follows:

```
MessageType = %x01 ; control

MessageValue = SmeConnectionAccessMethodsRequestValue

SmeConnectionAccessMethodsRequestValue = *OCTET
```

The content of SmeConnectionAccessMethodsRequestValue is defined as follows: The structure is defined by the SHIP root tag “accessMethodsRequest” (including the root tag “accessMethodsRequest”) of the XSD “SHIP_TS_TransferProtocol.xsd”. The default structure extensibility applies to this structure. The format of this structure MUST be the format agreed with the protocol handshake (see 13.4.4.2).

Element name	**Mandatory/ Optional/ Not Valid (NV)**	**Brief explanation**
accessMethodsRequest	M	The request for the recipient’s access methods.
		Note: Subsequent versions of this specification may define (optional) sub-elements of accessMethodsRequest.

Table 23: Structure of SmeConnectionAccessMethodsRequestValue of SME "Access methods request" message.

SME “Access methods” Message:

The SME “Access methods” message is defined as follows:

```
MessageType = %x01 ; control

MessageValue = SmeConnectionAccessMethodsValue

SmeConnectionAccessMethodsValue = *OCTET
```

The content of SmeConnectionAccessMethodsValue is defined as follows: The structure is defined by the SHIP root tag “accessMethods” (including the root tag “accessMethods”) of the XSD “SHIP_TS_TransferProtocol.xsd”. The default structure extensibility applies to this

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

structure. The format of this structure MUST be the format agreed with the protocol handshake (see 13.4.4.2).

Element name	Mandatory/ Optional/ Not Valid (NV)	Brief explanation
accessMethods	M	The originator's access methods.
accessMethods.id	M	The originator's unique ID or an empty string if the originator does not have such an ID: This element SHALL be set to the unique ID if the originator of the SME "Access methods" message has such a unique ID. Otherwise, the element SHALL be set to an empty string.
accessMethods.dnsSd_mDns	O	SHALL be present if the originator provides its SHIP service via service discovery as specified in chapter 7. Please note that this REQUIRES that the originator has a unique ID and consequently the element "accessMethods.id" MUST contain this value. Note: Subsequent versions of this specification may define (optional) sub-elements of accessMethodsRequest.dnsSd_mDns.
accessMethods.dns	O	SHALL be present if the originator provides its SHIP service with unicast DNS.
accessMethods.dns.uri	M	The URI where the originator provides its SHIP service. Please see also constraints defined in the text.

Table 24: Structure of SmeConnectionAccessMethodsValue of SME "Access methods" message.

The element "accessMethods.dns.uri" can take a URI as specified by IETF RFC 7320. However, this version of the SHIP specification considers only the URI scheme "wss" as used by WebSockets and defined by IETF RFC 6455.

Roles and Symbols:

Either side can request for the communication partner's "access methods" information. However, obligations on support and information differ depending on the role.

Subsequently, the following symbols are used:

1. **DEV-SERVER**

   This symbol is used for a device with the connection role "server".

2. **DEV-CLIENT**

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

This symbol is used for a device with the connection role “client”.

Please note the roles “server” and “client” denote only an aspect of the connection. They DO NOT denote an aspect of a specific functionality.

Further symbols for two devices “A” and “B” will be defined in section 13.4.7.1.1.

13.4.6.2.1 Process Details

There is no specific requirement in which case an SME “Access methods request” message needs to be sent. However, please consider the recommendations in section 13.4.6.2.2.

The recipient of an SME “Access methods request” message SHALL respond with an SME “Access methods” message. The sender of the aforementioned SME “Access methods request” message CAN close the connection according to section 13.4.7 if it does not receive a proper SME “Access methods” message within 60 seconds.

Unsolicited SME “Access methods” message SHALL NOT be sent. I.e. it SHALL ONLY be sent upon a received SME “Access methods request”.

The recipient of an SME “Access methods” message SHALL store the received information persistently for cases where it needs to initiate a connection to the originator of the message.

13.4.6.2.2 Recommendations

The device DEV-SERVER SHOULD request for DEV-CLIENT’s access methods if DEV-SERVER must be able to initiate a connection to the current DEV-CLIENT under certain circumstances but has no proper information so far.

Note: This specification does not describe binding or subscription processes. However, such use cases are typical for higher layers. Imagine device “A” is DEV-CLIENT and connects to device “B” (DEV-SERVER) and subscribes to some data provided by device “B”. I.e. it asks device “B” to submit data changes to device “A”. Such cases typically intend that device “B” is also able to establish a connection to device “A”. Thus, device “B” SHOULD request for device “A”’s “Access methods” information as long as device “B” is DEV-SERVER.

13.4.7 Connection Termination

13.4.7.1 Basic Definitions

In this state, the SME Users announce or negotiate the termination of a connection. This denotes the regular end of a connection in contrast to a sudden connection interrupt or failure. However, the methods described here do NOT apply in general, i.e. they apply only for the states and situations described below.

This state can run in parallel to connection data exchange (in order to finish a pending “data” message before the connection is finally closed, e.g.). In fact, this state MUST NOT be entered before connection data exchange is entered (i.e. the termination process does NOT apply for earlier states like “hello” or “CMI”, e.g.).

This state uses an SME “close” message which is defined as follows:

```
MessageType = %x03 ; end

MessageValue = SmeCloseValue

SmeCloseValue = *OCTET
```

The content of SmeCloseValue is defined as follows: The structure is defined by the SHIP root tag “connectionClose” (including the root tag “connectionClose”) of the XSD “SHIP_TS_TransferProtocol.xsd”. The default structure extensibility applies to this structure. The format of this structure MUST be JSON-UTF8.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

Element name	Mandatory/ Optional/ Not Valid (NV)	Brief explanation
connectionClose.phase	M	The sender's phase during the "close" process (enumeration: announce, confirm).
connectionClose.maxTime	O	Remaining time (in milliseconds) granted by the sender.
connectionClose.reason	O	Reason for the termination. See 13.4.7.1.1.

Table 25: Structure of SmeCloseValue of SME "close" Message.

The SME "close" process does not require knowledge of the connection role (server or client). Instead, each of the SME Users SHALL execute the process as described subsequently.

13.4.7.1.1 Process Overview

Either side can initiate a connection termination. The respective other side SHALL confirm the termination request accordingly. If – for any reason – a confirmation is not sent or not received in time, the requesting side SHALL close the connection and the respective other side SHALL expect the connection to be closed.

Subsequently the following symbols are used:

1. **DEV-A**

   This symbol is used for a device that initiates a connection termination.

2. **DEV-B**

   This symbol is used for the communication partner of DEV-A.

The reason to close a connection SHALL also be part of the message (sub-element "reason"). The following reasons are defined:

1. **unspecific**

   This value SHALL be used if no other value fits better.

   Remark (informative): This value typically denotes a rather temporary disconnection (e.g. because a device has limited connection capabilities – it may just support one active connection but needs to exchange data with multiple devices; e.g. the device needs to reboot for a firmware update). This means it is likely that it is possible to re-establish a connection later on in order to continue with next/remaining data exchange.

2. **removedConnection**

   This value denotes the removal of the respective node from the list of "known" or "accepted" devices.

   Remark (informative): This value does not mean a reconnection will not be possible at a later time. However, assuming a reconnection is executed, the device will be treated like a new or unknown device.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

13.4.7.1.2 Process Details

Rules for DEV-A:

1. If an SME User wants to initiate a connection termination, it SHALL send an SME “close” message to the communication partner with the following content:
    a. Sub-element “phase” set to “announce”.
    b. Sub-element “maxTime” set to a value when DEV-A will close the connection at the latest (i.e. even if no confirmation from DEV-B was received). The value denotes the duration in ms (milliseconds), starting from the time the message is sent.
    c. Sub-element “reason” set to a proper value (see 13.4.7.1.1).
2. If an SME User initiated a connection termination, it SHALL close the connection latest after a duration of its announced “maxTime”. I.e. the connection SHALL then be closed even if no confirmation from DEV-B was received.
3. If an SME User initiated a connection termination and receives a confirmation from DEV-B in time (i.e. before the announced duration “maxTime” elapsed) it SHALL close the connection immediately.

Rules for DEV-B:

1. If an SME User receives a connection termination, it SHALL prepare stopping its state “connection data exchange” before the received duration of “maxTime” expires. If this was achieved in time and the connection is still not closed, it SHALL send an SME “close” message to the communication partner, with sub-element “phase” set to “confirm” and no other sub-element set. Afterwards, it SHALL close the connection.
2. If an SME User receives a connection termination but does not manage to submit the confirmation in time, it SHALL consider the connection as closed after the received duration of “maxTime” expired.

General rules:

1. It can happen that both sides initiate a connection termination at almost the same time. In this case, each side is both a DEV-A as well as a DEV-B (with different parameters, esp. different “maxTime”). In this case, the confirmation that is sent first closes the connection (i.e. there is no need for a confirmation from both sides). However, the respectively received “reason” values need to be considered with regards to the importance. I.e. a received “removedConnection” is more important than the value “unspecific”.

The execution of reconnection attempts is application specific in general. However, in case of a regular termination process, it SHOULD be avoided to attempt a reconnection immediately.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.

14 Well-known protocolId

protocolId	Definition
ee1.0	EEBus specifications that are compatible to the SPINE data model specification base version 1.0.

Copyright © 2019 EEBus Initiative e.V. All Rights Reserved.