ETSI GS ORI 002-2 V1.1.1 (2012-08)

Open Radio equipment Interface (ORI);
ORI Interface Specification;
Part 2: Control and Management
(Release 1)

Disclaimer

This document has been produced and approved by the Open Radio equipment Interface (ORI) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGS/ORI-0002-2

Keywords
control, interface, management, radio

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

Contents

Intellectual Property Rights 7
Foreword 7
1 Scope 8
2 References 8
2.1 Normative references 8
2.2 Informative references 9
3 Definitions, symbols and abbreviations 9
3.1 Definitions 9
3.2 Symbols 10
3.3 Abbreviations 11
4 Introduction 12
5 C&M plane description 12
5.1 Protocol stack 12
5.2 C&M plane protocols 12
5.2.1 Ethernet / Fast C&M 13
5.2.2 IP 13
5.2.3 DHCP 13
5.2.4 TCP / UDP 13
5.2.5 FTP 13
5.2.6 ORI C&M Protocol (OCP) 13
5.2.7 AISG/3GPP Iuant interface support within ORI 14
5.3 OCP message encoding and object modelling 14
5.4 Vendor specific extensions 15
6 RE Resource Model and its related Management Functions 15
6.1 Introduction 15
6.2 Object types 17
6.3 Object lifecycle 17
6.4 Object naming / addressing 17
6.5 Object relations 19
6.6 Object parameters 20
6.6.1 Roles/Purposes of parameters 20
6.6.2 Parameter characterization 20
6.6.2.1 Object containment 20
6.6.2.2 Parameter Name 20
6.6.2.3 Parameter Type 21
6.6.2.4 Parameter Access 21
6.6.2.5 Default value 21
6.6.3 Parameter data types 21
6.6.4 Parameter access functions 22
6.7 Object states / State handling 22
6.7.1 State types / states 22
6.7.1.1 Administrative State (AST) 22
6.7.1.2 Functional state (FST) 24
6.7.1.3 Relationship between Administrative and Functional states 24
6.7.2 State management functions 25
6.7.3 Linkage between object state changes and fault reporting 25
6.8 Fault management 25
6.8.1 Fault States 26
6.8.2 Fault Severity 26
6.8.3 Fault Reporting 26
6.8.4 Fault History 27
6.8.5 ORI Fault Types 27
6.8.6 Handling of Fault Reports by the REC 27

6.9 Performance Management 27
6.10 Logging 27
6.10.1 Log concept 27
6.10.2 Log types & content 28
6.10.3 Log categories 29
6.10.4 Log configuration and control 29
6.10.5 Log file naming 30
6.11 Vendor specific extensions to the resource model 30
6.11.1 Vendor specific parameters 31
6.11.2 Vendor specific object types 31
6.11.3 Vendor specific fault types 31

7 OCP format and encoding structure 32
7.1 Roles of REC/RE 32
7.2 Message format and encoding 32
7.2.1 Message types/handling 32
7.2.2 Encoding basis 33
7.2.3 Data types 33
7.2.4 Name spaces 36
7.2.5 Message structure 37
7.2.5.1 Message Header 37
7.2.5.2 Message body 38
7.2.5.3 OCP message 38
7.3 Transport protocol embedding 38
7.3.1 Message framing 38
7.4 Common default result codes 39
7.5 Default failure response message 40
7.5.1 Message parameter details 41
7.5.2 Message encoding 42

8 OCP Elementary Functions and Messages 43
8.1 Device management 43
8.1.1 Health Check 43
8.1.1.1 Success 43
8.1.1.2 Failure 43
8.1.1.3 Message Parameter Details 43
8.1.2 Set Time 45
8.1.2.1 Success 45
8.1.2.2 Failure 45
8.1.2.3 Message parameter details 45
8.1.2.4 Message encoding 46
8.1.3 RE Reset 46
8.1.3.1 Success 47
8.1.3.2 Failure 47
8.2 Software management 48
8.2.1 Version Query 48
8.2.1.1 Success 48
8.2.1.2 Failure 49
8.2.1.3 Message parameter details 49
8.2.1.4 Message encoding 49
8.2.2 Software Update Preparation 50
8.2.2.1 Success 51
8.2.2.2 Failure 51
8.2.2.3 Message parameter details 51
8.2.2.4 Message encoding 52
8.2.3 Software Download 52
8.2.3.1 Success 53
8.2.3.2 Failure 53
8.2.3.3 Message parameter details 53
8.2.3.4 Message encoding 54
8.2.4 Software Activation 54
8.2.4.1 Success 55

8.2.4.2 Failure 55
8.2.4.3 Message parameter details 55
8.2.4.4 Message encoding 55
8.3 Configuration management 56
8.3.1 Object Parameter Reporting 56
8.3.1.1 Success 57
8.3.1.2 Failure 57
8.3.1.3 Message Parameter Details 58
8.3.1.4 Message encoding 58
8.3.2 Object Parameter Modification 60
8.3.2.1 Success 61
8.3.2.2 Failure 61
8.3.2.3 Message parameter details 62
8.3.2.4 Message encoding 63
8.4 Object lifecycle 65
8.4.1 Object Creation 65
8.4.1.1 Success 66
8.4.1.2 Failure 66
8.4.1.3 Message parameter details 67
8.4.1.4 Message encoding 68
8.4.2 Object Deletion 71
8.4.2.1 Success 71
8.4.2.2 Failure 71
8.4.2.3 Message parameter details 72
8.4.2.4 Message encoding 72
8.5 Object State management 73
8.5.1 Object State Reporting 73
8.5.1.1 Success 73
8.5.1.2 Failure 74
8.5.1.3 Event-triggered reporting of state change 74
8.5.1.4 Message parameters 75
8.5.1.5 Message encoding 76
8.5.2 Object State Modification 78
8.5.2.1 Success 78
8.5.2.2 Failure 79
8.5.2.3 Abnormal operation 79
8.5.2.4 Message parameter details 79
8.5.2.5 Message encoding 80
8.6 Fault management 82
8.6.1 Fault Reporting 82
8.6.1.1 Success 82
8.6.1.2 Failure 83
8.6.1.3 Event-triggered reporting 83
8.6.1.4 Message parameter details 83
8.6.1.5 Message encoding 84
8.7 Performance management 87
8.8 Logging 87
8.8.1 File Available Indication 87
8.8.1.1 Message Parameter details 87
8.8.1.2 Message encoding 87
8.8.2 File Upload 88
8.8.2.1 Success 89
8.8.2.2 Failure 89
8.8.2.3 File Upload Completion reporting 90
8.8.2.4 Message parameter details 90
8.8.2.5 Message encoding 91
8.8.3 Activate Configuration 93
8.8.3.1 Success 94
8.8.3.2 Failure 94
8.8.3.3 Activate Configuration completion reporting 94
8.8.3.4 Message Parameter details 95
8.8.3.5 Message encoding 95

8.9 AISG specific procedures 98
8.9.1 Device scan 98
8.9.1.1 Success 98
8.9.1.2 Failure 99
8.9.1.3 Device Scan completion 99
8.9.1.4 Message parameter details 99
8.9.1.5 Message encoding 100
8.9.2 AISG Layer 7 message and Alarm transfer 101
8.9.2.1 AISG message transmission (between REC and RE) 102
8.9.2.1.1 Success 102
8.9.2.1.2 Failure 102
8.9.2.2 AISG ALD receive indication 103
8.9.2.3 Message parameter details 103
8.9.2.4 Message encoding 103

9 Specified object types/parameters and fault types 105
9.1 Specified object types and their associated parameters 105
9.1.1 RE Object 105
9.1.2 Physical Antenna Port Object 107
9.1.3 TxSigPath Object 108
9.1.4 RxSigPath Object 111
9.1.5 ORI Link Object 113
9.1.6 External Event Port Object 116
9.1.7 AISGPort Object 116
9.1.8 AISGALD Object 118
9.1.9 Log Object 121
9.2 Specified fault types 123

10 RE management procedures 125
10.1 RE Device management 125
10.1.1 OCP layer establishment and supervision between REC and RE 125
10.1.1.1 DHCP options 128
10.1.1.2 TCP options 129
10.1.2 RE startup procedure / Alignment 129
10.1.3 Software management 130
10.1.3.1 RE restart triggers 130
10.1.3.2 Restart actions 130
10.1.3.3 REC<->RE Software Alignment 131
10.2 RE operation 132
10.2.1 Dynamic object initial status alignment 132
10.2.2 ORI link configuration 133
10.2.2.1 General 133
10.2.2.2 Transceiver module inserted during RE operation 133
10.2.3 ORI link maintenance 134
10.2.4 Delay calibration 134
10.2.5 Signal Path control (Cell configuration) 138
10.2.5.1 Signal Path setup 138
10.2.5.2 Modify a Signal Path parameter when locking required 140
10.2.5.3 Modify a Signal Path parameter when locking not required 143
10.2.5.4 Delete a Signal Path 144
10.2.5.5 Switch a Signal Path off and on 144
10.2.6 AISG/3GPP Iuant management 145
10.2.6.1 AISG/3GPP Iuant Layer 2 establishment 145
10.2.6.1.1 Layer 2 connection establishment using Device Scan 146
10.2.6.1.2 Layer 2 connection establishment using AISGALD object creation requested by the REC 147

Annex A (informative): Example for a vendor specific parameter in the RE resource model 149

Annex B (normative): ORI vendor codes 150

History 151

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification (ISG) Open Radio equipment Interface (ORI).

The present document is part 2 of a multi-part deliverable covering the ORI Interface Specification, as identified below:

Part 1: "Low Layers (Release 1)";

Part 2: "Control and Management (Release 1)".

1 Scope

The present document aims to define Control and Management plane functions and protocols, addressing, message format and coding of the Open Radio equipment Interface (ORI) for Release 1.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

[1] Common Public Radio Interface (CPRI): "Interface Specification" V 4.1.

NOTE: Available at http://www.cpri.info/spec.html.

[2] ETSI GS ORI 001: "Open Radio equipment Interface (ORI); Requirements for Open Radio equipment Interface (ORI) (Release 1)".

[3] ETSI GS ORI 002-1: "Open Radio equipment Interface (ORI); ORI Interface Specification; Part 1: Low Layers (Release 1)".

[4] IEEE Std 802-2001: "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture".

[5] W3C Recommendation 28 October 2004: "XML Schema" (part 0 to 2).

NOTE: Part 0: Primer Second Edition http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/
Part 1: Structures Second Edition http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/
Part 2: Datatypes Second Edition http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[6] The Unicode Consortium: " The Unicode Standard V6.0.0"; edited by Julie D. Allen ... [et al.]. Version 6.0, February 2011, ISBN 978-1-936213-01-6.

NOTE: http://www.unicode.org/versions/Unicode6.0.0/.

[7] W3C Recommendation 26 November 2008: "The Extensible Markup Language (XML) 1.0" Fifth Edition.

NOTE: http://www.w3.org/TR/2008/REC-xml-20081126/.

[8] IETF RFC 2045 (1996): Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.

NOTE: http://www.ietf.org/rfc/rfc2045.txt.

[9] ETSI TS 125 461 (V10.2.0): "Universal Mobile Telecommunications System (UMTS); UTRAN Iuant interface: Layer 1 (3GPP TS 25.461 version 10.2.0 Release 10)".

[10] ETSI TS 125 462 (V10.1.0): "Universal Mobile Telecommunications System (UMTS); UTRAN Iuant interface: Signalling transport (3GPP TS 25.462 version 10.1.0 Release 10)".

[11] ETSI TS 125 466 (V10.3.0): "Universal Mobile Telecommunications System (UMTS); UTRAN Iuant interface: Application part (3GPP TS 25.466 version 10.3.0 Release 10)".

[12] Antenna Interface Standards Group, Standard No. AISG v2.0, 13th June 2006 "Control interfaces for antenna line devices".

NOTE: Available at www.aisg.org.uk.

[13] ETSI TS 125 215: "Universal Mobile Telecommunications System (UMTS); Physical layer; Measurements (FDD) (3GPP TS 25.215)".

[14] IETF RFC 959: "File Transfer Protocol".

[15] ISO/IEC 13239 (2002): "Information technology - Telecommunications and information exchange between systems - High-level data link control (HDLC) procedures".

[16] IETF RFC 2131: Dynamic Host Configuration Protocol.

[17] IETF RFC 2132: DHCP Options and BOOTP Vendor Extensions.

[18] IETF RFC 793: TCP: Transmission Control Protocol.

[19] ETSI TS 125 104: "Universal Mobile Telecommunications System (UMTS); Base Station (BS) radio transmission and reception (FDD) (3GPP TS 25.104)".

[20] ETSI TS 136 104: "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception (3GPP TS 36.104)".

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] The Apache Software Foundation: Apache log4cxx™ for C++.

NOTE: http://logging.apache.org/log4cxx/index.html

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

active link: See definition in [2].

active software image: software image that is currently being executed on the RE

Antenna Line Device (ALD): See definition in [12].

Antenna-Carrier (AxC): See clause 2.1 of [1].

DHCP client: See definition in [16].

DHCP server: See definition in [16].

FTP password: See definition in [14].

FTP server IP address: See definition in [14].

FTP username: See definition in [14].

universally unique MAC address: 48 bit universally administered, globally unique identifier in compliance with MAC-48/EUI-48 format assigned to a network interface, as defined in IEEE Std 802-2001 [4]

HDLC control octet: See definition in [15].

I-frame: See definition in [15].

ORI line bit rate: See definition in [3].

ORI link: See definition in [2].

ORI port: See definition in [2].

passive link: See definition in [2].

passive software image: software image that is not currently being executed on the RE

radio standard: standardised radio access technology (e.g. UMTS, LTE, GSM)

Received Total Wideband Power (RTWP): See [13].

software image: software stored on the RE as a result of a successful software upgrade that is compatible with the RE hardware version

software upgrade package version identifier: information exchanged between the REC and the RE to identify the Software Upgrade Package

software upgrade package: unique set of (one or several) files provided by the RE vendor for the purpose of performing a software upgrade of one or more RE hardware versions provided by the same RE vendor, and associated with a maximum of one Software Image on the RE

NOTE 1: The RE vendor should provide flexibility for supporting Software Upgrade Packages containing software for multiple hardware versions and Software Upgrade Packages containing software for a single Hardware version.

NOTE 2: Even if one of the files of a set changes, a new software upgrade package is created.

TCP socket: See definition in [18].

transceiver module: module that may be used to connect the internal circuits of the RE to the Layer 1 (see GS ORI 002-1 [3]) of the ORI link

Uplink Automatic Gain Control (UL AGC): See definition in [3].

3.2 Symbols

For the purposes of the present document, the following symbols apply:

AxC	Antenna-Carrier
Rx	Receiver
Tc	UTRA FDD chip period (1/3,84 MHz)
T_{DSR}	DHCP Server Response Timer
T_{HCI}	Health Check Idle timer
T_{TSY}	TCP Synchronization Timer
T_{TCE}	TCP Connection Establishment Timer
T_{TLM}	TCP Link Monitoring Timer
Tx	Transmitter

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

3GPP	3rd Generation Partnership Project
AGC	Automatic Gain Control
AISG	Antenna Interface Standards Group
ALD	Antenna Line Device
AST	Administrative State
AxC	Antenna-Carrier
C&M	Control & Management
CET	Central European Time
CPRI	Common Public Radio Interface
DHCP	Dynamic Host Control Protocol
DL	DownLink
EARFCN	E-UTRA Absolute Radio Frequency Channel Number
EBNF	Extended Backus-Naur Form
E-UTRA	Evolved-UMTS Terrestrial Radio Access
FCS	Frame Check Sequence
FDD	Frequency Division Duplex
FST	Functional STate
FTP	File Transfer Protocol
HDLC	High Level Data Link Control
IAB	Individual Address Block
ID	Identifier
IE	Information Element
IETF	Internet Engineering Task Force
IP	Internet Protocol
IQ	In-phase data and Quadrature data
LAN	Local Area Network
LED	Light Emitting Diode
LOF	Loss Of Frame
LOS	Loss Of Signal
MAC	Media Access Control
MIMO	Multiple-Input Multiple-Output
MTU	Maximum Transmission Unit
O&M	Operations & Maintenance
OCP	ORI C&M Protocol
ORI	Open Radio Interface
OUI	Organizationally Unique Identifier
PA	Power Amplifier
RE	Radio Equipment
REC	Radio Equipment Controller
RF	Radio Frequency
RMS	Root Mean Square
RO	Read-Only
RTWP	Received Total Wideband Power
RW	Read Write
SSH	Secure SHell
TCP	Transmission Connection Protocol
TMA	Tower-Mounted Amplifier
UARFCN	UTRA Absolute Radio Frequency Channel Number
UDP	User Datagram Protocol
UID	Unique IDentifier
UL AGC	Uplink Automatic Gain Control
UL	UpLink
UTC	Coordinated Universal Time
UTF	Unicode Transformation Format
UTRA	UMTS Terrestrial Radio Access
VSWR	Voltage Standing Wave Ratio

XID eXchange Identification

NOTE: As defined in [15].

XML eXtensible Markup Language
XSD XML Schema Definition

4 Introduction

The present document is a subset of the ORI Release 1 specifications.

5 C&M plane description

5.1 Protocol stack

The C&M plane protocol stack that shall be supported by REC and RE within the overall ORI protocol stack, is shown in figure 5.1-1 and complies with [1], clause 4.1.

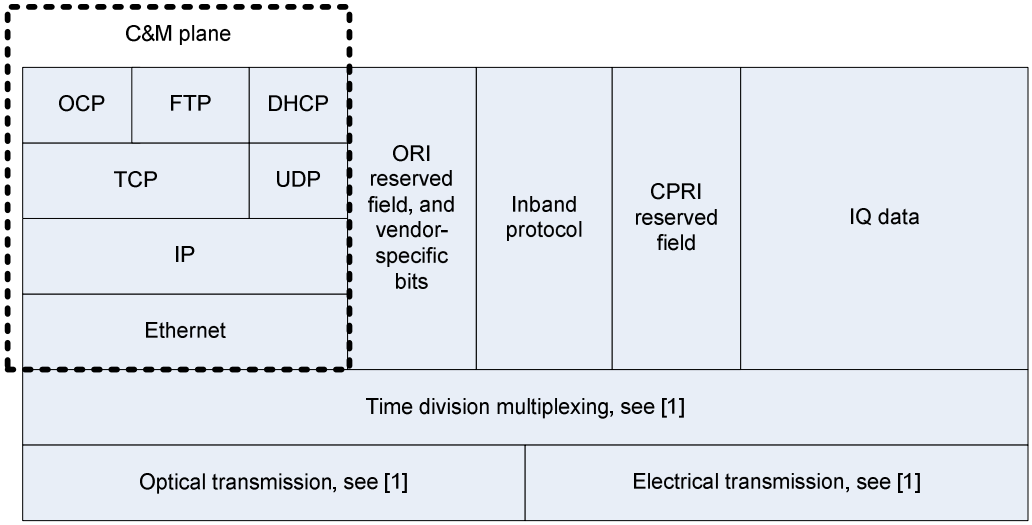

Figure 5.1-1: ORI protocol stack highlighting C&M plane

The IQ data, CPRI reserved field, Vendor-specific bits (a subset of which are used for the ORI reserved field), and Inband protocol, are as defined in [1]. The ORI reserved field itself is defined in [3].

The use of Ethernet, IP, TCP, DHCP, FTP and OCP (ORI C&M Protocol) within the C&M plane is described in clause 5.2.

Further details of OCP are defined in clauses 7, 8 and 10.

5.2 C&M plane protocols

The C&M plane consists of the following:

- OCP.
- Protocols that provide the underlying transport for the OCP layer.
- Protocols used by OCP to perform its functions. OCP functions are described in clause 5.2.6.

The OCP layer shall be transported over a TCP/IP connection established between the RE and the REC (see clause 10.1.1). This TCP/IP connection shall use the IP address provided to the RE via DHCP. The IP datagrams are transported in Ethernet frames. Ethernet is the ORI Data link layer, which is carried over the Fast C&M channel, as specified in [3].

FTP services shall be used by OCP for file transfer.

Additional IP-based protocols (such as Telnet, SSH) may be supported to allow access from an external node to the RE via the REC.

5.2.1 Ethernet / Fast C&M

The RE shall use universally unique MAC addresses to ensure that there is no conflict of devices on the Ethernet LAN.

The MTU (the largest number of bytes that can be carried by an Ethernet frame, excluding the header and trailer) is the Ethernet default value (1 500 bytes).

NOTE: The way Ethernet frames are managed in case of daisy chained configuration is out of the scope of the present document.

5.2.2 IP

The REC communicates with the REs using IP V4 protocols. There shall be one RE IP address for the C&M communication context.

5.2.3 DHCP

DHCP shall be used by the RE to acquire an IP address for OCP layer establishment. DHCP shall be supported by the REC and RE. DHCP shall be used as specified in RFC 2131 [16].

DHCP vendor specific codes and their usage within ORI are detailed in clause 10.1.1.1.

5.2.4 TCP / UDP

TCP shall provide the transport for OCP.

The REC shall be responsible for establishing the TCP connection to the RE.

The TCP port for transporting OCP shall be indicated by the RE within a Vendor Specific code in DHCP.

The establishment of the TCP/IP connection for OCP layer transport is further described in clause 10.1.1, and the TCP options that shall be used are defined in clause 10.1.1.2.

5.2.5 FTP

FTP shall be used as specified in [14]. Passive mode shall be used for all file transfers.

The RE shall provide FTP client services to allow the REC to use OCP services related to file management services.

The RE may provide a FTP server for service purposes.

5.2.6 ORI C&M Protocol (OCP)

OCP provides the following functions:

- Control & Management of the RE by the REC.
- Transport of AISG/3GPP Iuant Layer 7 messages and alarms between REC and RE.

The Control & Management function consists of Software Management, Configuration Management, State Management, Fault Management, Log and File handling, ORI link management, and AISG ALD Layer 2 connection management (see clause 5.2.7).

To perform the OCP functions, the REC and RE shall interact using the "RE resource model" (described in clause 6), and via the exchange OCP elementary messages (described in clause 8). Some of the more complex interactions between REC and RE are further described in clause 10.

OCP layer establishment and supervision between REC and RE shall be as described in clause 10.1.1.

5.2.7 AISG/3GPP Iuant interface support within ORI

ORI supports operation of the AISG/3GPP Iuant interface for the architecture where an RE has one or more connected AISG Antenna Line Device (ALD), and where the Layer 1&2 of the AISG/3GPP Iuant interface terminate in the RE and ALD, with the Layer 7 of the AISG/3GPP Iuant interface terminating in the REC and ALD. The protocol stack is shown in figure 5.2.7-1.

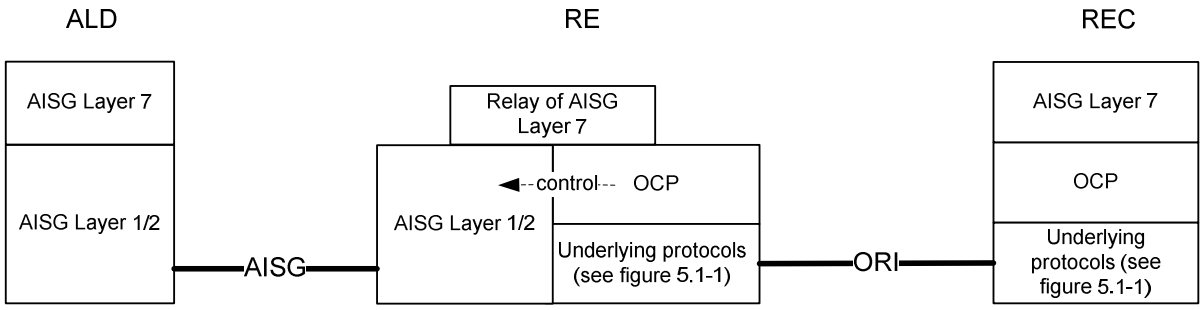

Figure 5.2.7-1: Protocol stack for OCP as a transport for AISG/3GPP Iuant Layer 7

ORI specifies the 2 main functions:

- AISG/Iuant interface message tunnelling: the use of OCP to transport the AISG/ 3GPP Iuant layer 7 messages and alarms between the REC and RE using ORI.
- ALD connection management: the use of OCP to manage an AISG Layer 2 connection between RE and ALD that transports the AISG/ 3GPP Iuant layer 7 messages and alarms between RE and ALD.

In order to support this functionality, the RE shall support the primary functions for layer 1 and 2 according to TS 125 461 [9], TS 125 462 [10] and AISG [12] specifications, as well as the following functions:

- The AISG/3GPP Iuant layer 2 control, through which the management of the AISG/3GPP Iuant Layer 2 connection between RE and ALD is performed. AISG/3GPP Iuant Layer 2 connection establishment is further described in clause 10.2.6.1.
- The AISG/3GPP Iuant layer 7 message and alarm forwarding, which transparently relays AISG/3GPP layer 7 messages and alarms between the OCP and the AISG/3GPP Iuant Layer 2 connection. The RE does not interpret these messages/alarms, it packs them into a HDLC Frame according to the AISG/3GPP specification.

These functions are performed via one of the AISG ports in the RE. The control and management of AISG ports and their connected ALDs is performed via the RE resource model (see clause 6), using the AISGPort object (see clause 9.1.7) and the associated AISGALD child objects (see clause 9.1.8).

Also a further description of the AISG/3GPP Iuant management procedures is described in clauses 8 and 10.2.6.

5.3 OCP message encoding and object modelling

A significant part of the OCP messages conveys or manipulates data which is described in the RE Resource Model. Since the OCP messages are XML encoded, the objects of the Resource Model shall also be specified in a XML compliant manner. This specifically includes (but is not necessarily limited to) the data types, which shall be based on XML defined base types, or XML-codable derivatives hereof.

Details are defined in clause 6.

5.4 Vendor specific extensions

Vendor specific extensions may be provided as additions to existing ORI defined means; they shall not remove, override or re-define ORI specified parts of the interface.

Vendor-specific extensions shall not impair the performance of standard compliant devices that do not have these extensions. Neither REC nor RE shall rely on the presence of such extensions to provide the ORI-defined functionality.

Vendor specific extensions, taking into account the rules above, are possible on all layers of the ORI.

When receiving ORI compliant vendor specific elements, RECs and REs which are not aware of those shall tolerate them without failure (usually by ignoring them).

A method for realization of extensions to the RE Resource Model (additional parameters, or additional object types) is defined in clause 6.11.

6 RE Resource Model and its related Management Functions

6.1 Introduction

The RE has a defined set of logical or physical resources that are to be managed by the REC. These resources are represented towards the REC by data objects called "objects", which are accessible by the REC through a set of common management functions. The set of defined objects and their relationship define the RE resource model.

The role of the resource model is to provide an external management view of the RE's capabilities to the REC, without relying on or revealing implementation details of the RE. In addition, the concept of using a dedicated resource model allows for modification and extension of the RE management interface while keeping the related management functions untouched. This simplifies specification maintenance and reduces implementation and test effort.

The purpose / scope of the resource model is to provide the following services to the REC:

- Configuration Management (CM):
  This service provides the RE configuration parameters, and read and modification access functions to configuration parameters. It also provides the functions to manage the life cycle of dynamic Resource Model objects.
  In addition to RE configuration, this service is also used to provide information about capabilities of the RE (e.g. presence or number of certain physical or logical resources).

- State Management (SM):
  This service provides the state of resources (objects) in the RE, and read and modification access functions to states. It also provides a configurable notification service to the REC about state changes

- Fault Management (FM):
  This service provides information about the RE fault status, and access functions to get information about the status of fault reports. It also provides a configurable notification service to the REC about fault report state changes.

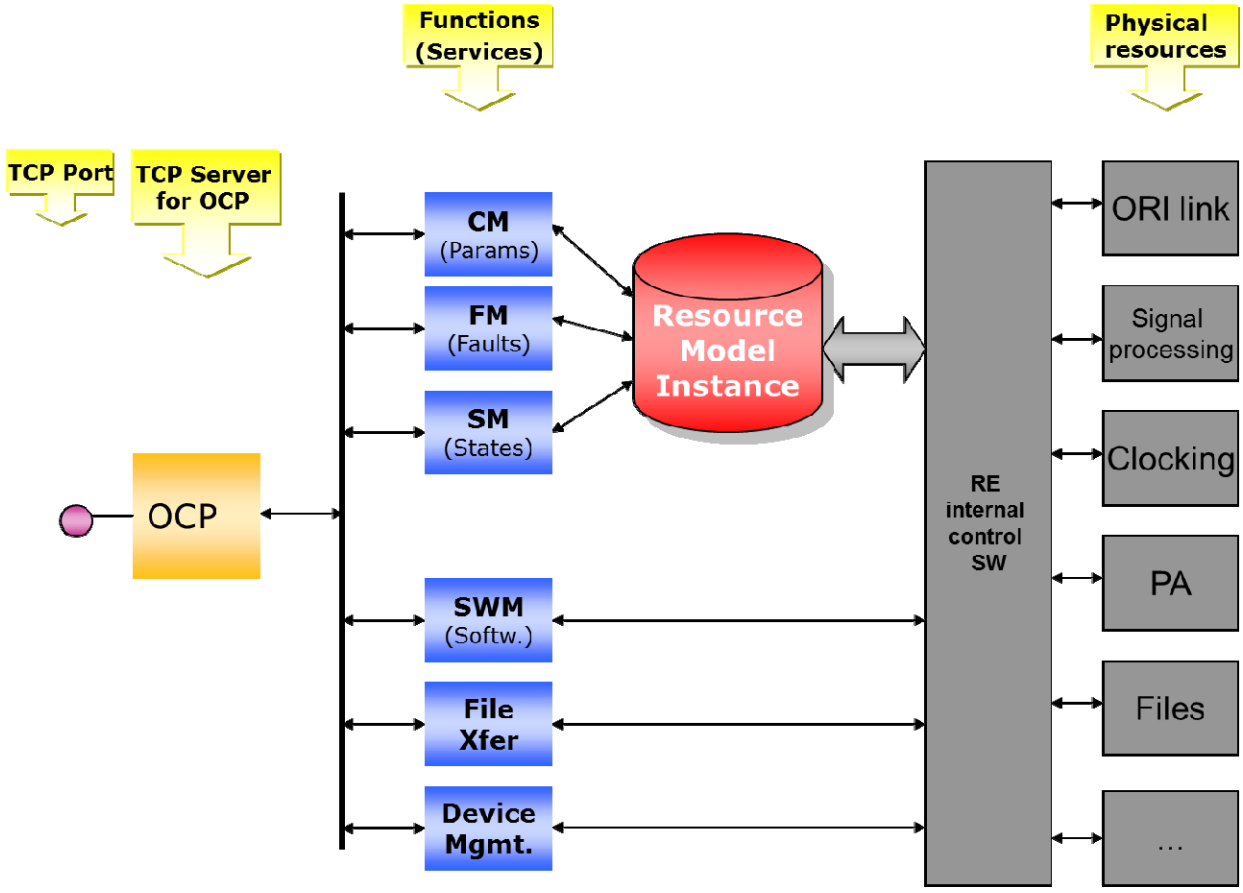

Figure 6.1-1: System view of the Resource Model

The individual objects of the Resource Model represent either physical or logical entities within the RE. Examples for physical entities are ports, like the ORI link ports or the antenna ports, but also the RE itself. Examples for logical entities are the Signal Paths, which bear configuration parameters for the processing of individual antenna carriers; they finally may be allocated and use physical resources, but do not represent these.

An object consists of parameters, states and faults; not all of them need to be defined for a given object.

Objects may contain relations to other objects, which are realized as object parameters of a dedicated "object reference" type. The semantics of individual relations are described as part of the detailed object descriptions. Objects may also contain other objects (parent - child relationship), which is realized by using hierarchical naming.

The object descriptions in the present document describe the types of objects which the Resource Model consists of. Zero, one or more instances of a given object type may actually be instantiated in the RE. Instances of "static" objects are created by the RE at start-up time (during system initialization), and remain instantiated until the RE shuts down. The number of static instances is defined as part of the object description, and may be either a fixed number, or be related to the actual number of underlying physical resources represented by the given object type being present on the RE. Instances of "dynamic" objects can be created and deleted by the REC during operation of the RE.

The REC may retrieve information from the RE about the existing object instances (objects) at any time during RE operation by using so-called "wildcard" queries. The REC may also retrieve information from the RE about the parameter values, states and faults of existing objects.

The information provided by the RE Resource Model instances (object instances, parameter values, states, faults) allows the REC to understand the capabilities of the RE, to the degree these are reflected in the Resource Model, as well as their operational status.

More details about the aspects described in this clause can be found in subsequent clauses of this clause. In addition, clause 8 contains the descriptions of the management elementary functions, and clause 10 provides additional descriptions of management procedures for some of the primary use cases of the RE Resource Model.

6.2 Object types

Object types are the "construction plans" for the object instances derived from them. Object types are identified using object type names; they are characterized by:

- the set of parameters they contain;
- the set of states they contain;
- the set of faults they can report;
- their embedding in parent/child relationships;
- their lifecycle types (static/dynamic).

One common set of access methods is defined for all object types for each of these aspects. Where needed, individual additional methods are defined on a per-object type basis.

The detailed object types are defined in clause 9.1.

6.3 Object lifecycle

Objects can be either static or dynamic.

Static objects are instantiated at system startup, before C&M link is established, and remain instantiated until shutdown. In consequence, Object Creation or Object Deletion functions are not applicable to these objects; they shall issue an error when applied to static objects. Static objects typically represent resources where the cardinality (the number of instances) is defined by the physics of the RE. Examples are objects representing ports, like the ORI link ports or the physical antenna ports.

During the RE startup / alignment procedure between REC and RE, the REC can get a report on the static object instances and parameters being present on the RE.

Dynamic objects are object instances which can be created and deleted by the REC while the RE is operating, using the Object Creation / Object Deletion functions. Dynamic objects represent resources where the cardinality is not known in advance.

It is recommended that the Resource Model Reporting step of the RE startup/alignment procedure is finished before dynamic objects are created.

Clause 9.1 specifies whether an object type is static or dynamic.

6.4 Object naming / addressing

Object naming serves the purpose of unambiguously identifying every individual object type and object instance in the resource model. Note that the words "object instance" and "object" are often used synonymously, especially when it is clear from the context that object instances are meant by this.

The elements used to build object type and object instance references are:

- `namespaceIdentifier` (type: `string`, optional)
  The optional `namespaceIdentifier` shall be present for object types which do not belong to the default namespace "`ori`" (see clause 6.11).
  In case of namespace "`ori`", the `namespaceIdentifier` may be present, but is not strictly required.
  See clause 7.2.4 for a further description of namespaces.

- `objectTypeName` (type: `string`, with restrictions below)
  This is the name of the object type from which this object instance is created.

  - For ORI defined object types it shall be the XML encoded object type name as specified in clause 9.1.
  - For vendor specific object types it shall be the vendor defined object type name.

- The name shall consist of the character set [A-Z] [a-z] [0-9] "_".
- The name "ALL" is reserved for use as wildcard keyword.

- `instanceNumber` (type: `unsignedByte`)
  This is the instance number of the individual object instance of the same type. Instance numbers shall be unique across objects of the same type and, if applicable, the same parent object.

Object types are named using textual names; using the EBNF based notation from [7], clause 6, these are built according to the following rule

```
objTypeID = (namespaceIdentifier "::")? objectTypeName
```

EXAMPLE 1: `antPort, RE, xyz::SpecialControls`

Object instances are named using the object type name of the type they implement, and a numeric ID to identify the individual instance within the scope of a given parent object:

```
objID = objTypeID ":" instanceNumber
```

EXAMPLE 2: `antPort:0,xyz::SpecialControls:3`

Objects may contain other objects, see next clause. Contained ("child") objects are addressed by concatenating the object identifier of their parent objects with their objects, separated by a "/" (slash) character.

A complete reference to an object type consists of the parent object instance(s) identifiers(s) concatenated with a "/", and the object type identifier:

```
objTypeRef = (objID "/")* objTypeID
```

A complete object instance ("object") reference is then

```
objRef = objTypeRef ":" instanceNumber

       = (objID "/")* objID
```

Example for object instance reference: `aisgPort:0/aisgALD:4`

Child objects in a containment are fully qualified by the combination of parent object instance name and own name. No two objects shall have the same object ID under the same parent object type. Across different parent object types, child objects of the same name can exist without creating ambiguities (nevertheless the intention should be clear when doing this in practice).

For static objects representing physical resources, unless explicitly specified otherwise, the instance number that the RE allocates to an object instance representing a given physical resource shall remain the same, also between subsequent RE shut-down and start-ups (constant mapping of physical resource to object instance). The allocated instance number may change only when the used software image changes.

It is recommended that, for static object instances of the same object type, the RE should generate consecutive instance numbers starting with 0 and increasing in steps of 1.

Wildcarding:
Some procedures allow/need wildcarding to address a set of object instances.

Wildcards are used by the REC to address a set of objects (static and dynamic), usually within reporting procedures. The wildcard shall only address objects that exist at the time that the REC initiates the procedure. This also means that when using wildcards to address a set of objects on which to configure event-triggered reporting within a procedure, the event-triggered reporting configuration shall only apply to those objects which exist at the time the configuration was requested, i.e. the configuration shall not apply to objects that come into existence only after the configuration was requested.

The wildcard keyword is defined to be "`ALL`". In a wildcarded object reference, the wildcard shall be the terminating expression (no wildcard "in the middle" of a containment hierarchy).

Where wildcards are allowed (denoted in the individual procedure descriptions in clause 8), the syntax of a wildcarded object reference is:

```
objRefWc = (objID "/")* (objID | "ALL")
```

Examples:

- "ALL" addresses all objects
- "someObjwithChildren:0/ALL" addresses all child objects of object "someObjwithChildren:0"
- "objA:0/objChildOfA:1/ALL" addresses all child objects of a child object of object "objA:0"

Naming is a mandatory aspect of objects, i.e. the (type-) name of an object shall be a mandatory part of its specification.

6.5 Object relations

Two kinds of object relations are defined: containment and object references (object links).

Containment describes a relation where an object is part of a parent object. The contained object is called a child object. For example, an AxC object (describing the parameters of a CPRI antenna carrier on the ORI link) can be contained within the ORI link object.

Rules for containment relationships:

1) Parent objects may or may not have child objects.
2) Child objects shall (always) have one, and only one, parent object.

Consequences of these rules are:

- A static parent object can have static or dynamic child objects.
- A dynamic parent object shall (always) have dynamic child objects.
- Child objects shall be created after their parent object.
- Child objects shall be deleted before (or together with) their parent objects.
- If a dynamic parent object containing child objects is deleted, is child objects shall be deleted automatically.

Containment is a mandatory aspect of objects, i.e.:

- If an object is a part (child) of another object, the information about its parent object shall be a mandatory part of its specification.
- If an object has contained child objects, the information which objects are child objects shall be a mandatory part of its specification.

As described in the previous clause, object containment is expressed in the naming (addressing) of an object. A child object bears the identifier of its parent object instance in its object name. There are no explicit object reference parameters inside the bodies of parent or child objects pointing to each other.

The practical use of containments should be driven by data structuring considerations: when a parent object should get lists (arrays) of parameters which are either of non-fixed dimensions, or consist of complex data types, or both, then such parameters are candidates for child objects.

What should not solely drive containment are physical considerations, like "the RE contains an ORI link, therefore the ORI link shall become a child object", as this would lead to unnecessary complex object hierarchies.

Object links are used when object relations, that are not containments, need to be expressed in the resource model. An example for such a link can be a signal routing information (e.g. an AxC mapping).

Object links are realized as `objRef` type parameters in the body of the originating objects. The semantics ("meaning") of a given object link shall be described as part of the parameter specification of the originating object.

6.6 Object parameters

Parameters are the actual data items associated with objects. The general idea is that there are no other means to read or configure data of the RE than by using resource model parameters. Exceptions to this rule should be kept to a minimum, for example in startup scenarios where the resource model cannot be accessed.

6.6.1 Roles/Purposes of parameters

Object parameters serve two main purposes:

- Resource configuration
- Resource capability/constraints reporting

There are no technical means associated with these roles; they are purely determined by the intended use / semantics of every individual parameter.

6.6.2 Parameter characterization

Parameters are characterized by the following properties:

- Object containment
- Parameter name
- Parameter type
- Parameter access
- Default value

These aspects are described in more detail in the following clauses.

6.6.2.1 Object containment

Parameters never exist "standalone"; they are always defined as part of an object type. When the RE is operating, they exist as part of an object instance.

6.6.2.2 Parameter Name

Parameters are identified by textual names. They are fully qualified by the combination of object instance and parameter name. No two parameters shall have the same name in the same object type. Across different object types, parameters of the same name can exist without creating ambiguities (nevertheless the intention should be clear when doing this in practice).

Parameter names shall consist of the following elements:

- `namespaceIdentifier` (type: `string`, optional)
  The optional `namespaceIdentifier` shall be present for object types which do not belong to the default namespace "`ori`" (see clause 6.11).
  In case of namespace "`ori`", the `namespaceIdentifier` may be present, but is not strictly required.
  See clause 7.2.4 for further description of namespaces.

- `parameterName` (type: `string`, with restrictions below)
  This is the name of the parameter, for which the following applies:

  - For ORI defined parameters, it is the XML encoded parameter as specified in clause 9.1.
  - For vendor specific parameters it is the vendor defined XML encoded name
  - The name shall consist of the character set [A-Z] [a-z] [0-9] "_".

- The name "ALL" is reserved for use as wildcard keyword.

Using the EBNF based notation, an object parameter reference is defined as
`paramRef = (namespaceIdentifier "::")? parameterName`

Wildcarding:
Some procedures allow/need wildcarding to address the set of parameters belonging to a given object or set of objects (usually for reporting purposes). The wildcard keyword is defined to be "`ALL`". In a wildcarded parameter reference, the wildcard shall be the only expression (no "regular expressions").

Where wildcards are allowed (denoted in the individual procedure descriptions in clause 8), the syntax of a parameter name allowing wildcards shall be:

```
paramRefWc = paramRef | "ALL"
```

Using the wildcard ("ALL") in the access method addresses all parameters of the given object. If the given object contains child objects, the child objects and their parameters shall not be included in the set of parameters addressed by the wildcard.

6.6.2.3 Parameter Type

This is the data type a parameter is representing / encoding. As the description and encoding of the model is based upon XML/XSD, data types and constraint attributes from XML ([7] Part 2) can be used to specify object parameters precisely and in a way which is unambiguously mapping back to XML.

Object parameters can be primitive data types (like "`string`", "`unsignedInt`"), or complex data types like structures or arrays of primitive or complex types. In case of complex types, the primitive elements of these complex types are not individually addressable/accessible; the complex element can only be read or modified as a whole. Child (contained) objects should be considered instead of complex data types.

The set of data types and qualifiers used in ORI is listed in clause 7.2.3.

6.6.2.4 Parameter Access

It shall be defined whether a parameter shall be modifiable (read-write) or read-only from an REC's perspective. This is enforced by the parameter access functions described later; e.g. they refuse modification attempts to read-only parameters with an appropriate error code.

In addition, it shall be defined whether a modifiable parameter requires its object's administrative state to be "Locked" in order to allow consistent modification. This shall be enforced by the access functions, which shall fail when access restrictions are violated.

Valid access qualifiers are:

- RW: the parameter can be read and modified by the REC.
- RW-LOCKED: the parameter can be read always, but only be modified by the REC when the AST of its containing object is in "locked" state. This qualifier is only applicable to parameters of objects which have an Administrative State, see clause 6.7.1).
- RO: this parameter can only be read but not be modified by the REC.

6.6.2.5 Default value

If a default value is defined, it shall be set by the RE at the time of object instantiation.

Specifications for parameters and the associated object type definition shall contain the above mentioned information for every parameter.

6.6.3 Parameter data types

The XML data types defined in clause 7.2.3 are used as basis for encoding.

6.6.4 Parameter access functions

The REC accesses parameters through the following access functions. Note that the actual encoding will use different names, as shown in the examples (conceptual names are too long, and contain spaces).

- OBJECT PARAMETER REPORTING
  This function reads out the value of a given parameter and reports it to the REC.
  The elementary procedure defined to perform this function is specified in clause 8.3.1.

- OBJECT PARAMETER MODIFICATION
  This function modifies the value of a given parameter. The function checks whether the preconditions for modifications are met (R/W access, administrative lock conditions), and modifies the parameter to the requested value. Upon success the new parameter value is reported back for confirmation.
  Use of wildcards is not applicable for this function.
  The elementary procedure defined to perform this function is specified in clause 8.3.2.

No assumptions are made on how the RE internally accesses/handles parameters.

6.7 Object states / State handling

Objects may have states. The states that apply to each object in the present document of the specification are specified in clause 9.1.

6.7.1 State types / states

Two state types are proposed to describe the status of an object: administrative state (AST), and functional state (FST).

In addition the keyword "ALL" is used as wildcard to address all defined state types of the given object in procedures where this is applicable (further described in clause 8.5). If the given object contains child objects, the child objects and their states shall not be included in the states addressed by the wildcard.

6.7.1.1 Administrative State (AST)

This state is used to block a resource from operational use and make it available for configuration or other administrative activities. This state can be used to reserve resources for (re-)configuration which cannot be (re-)configured during operation, e.g. because they require re-calibration, or configuration could cause unwanted side effects when done while operating.

An object transitioning from "locked" to "unlocked" state has completely finished its configuration sequence, and is considered to be configured. The unlock request is used to trigger consistency checks to the object's configuration where necessary and applicable.

This behaviour is known as "transactional behaviour".

The state "unlocked" also implies that the object is put into use and its underlying physical resources (processing units, data/signal paths) are allocated.

AST states:

- Locked
  The object is blocked for operational use and available for configuration/administration.

- Unlocked
  The object is fully configured and available for operation. Underlying physical resources are allocated for use.

State transitions from "locked" to "unlocked" and vice versa are requested by the REC using the appropriate state modification request. Note that this trigger is literally to be considered as a *request*, since the RE implementation can defer or even deny the actual transition.

The RE implementation adjusts the AST according to the object's status during the transitions.

Transition activities:

- Object State Modification Request to state "Locked" (transition "lock"):
  Where necessary, the RE checks whether specified preconditions to take the object out of service are met (e.g. usage by other objects being still in service). If the check fails, the lock request is denied with an appropriate failure reason.

  If the check succeeds, the RE stops operation of the object and the AST is set to locked. Physical resources are de-allocated, but remain reserved.

- Object State Modification Request to state "Unlocked" (transition: "unlock"):
  The unlock request marks the end of the configuration sequence for the associated object. Where necessary, the RE checks consistency of the object's configuration settings and availability of its physical resources. If a check fails, the unlock request is denied with an appropriate failure reason.

  If checks succeed the request is accepted, underlying physical resources are allocated, operation is started, and the AST is set to "unlocked".

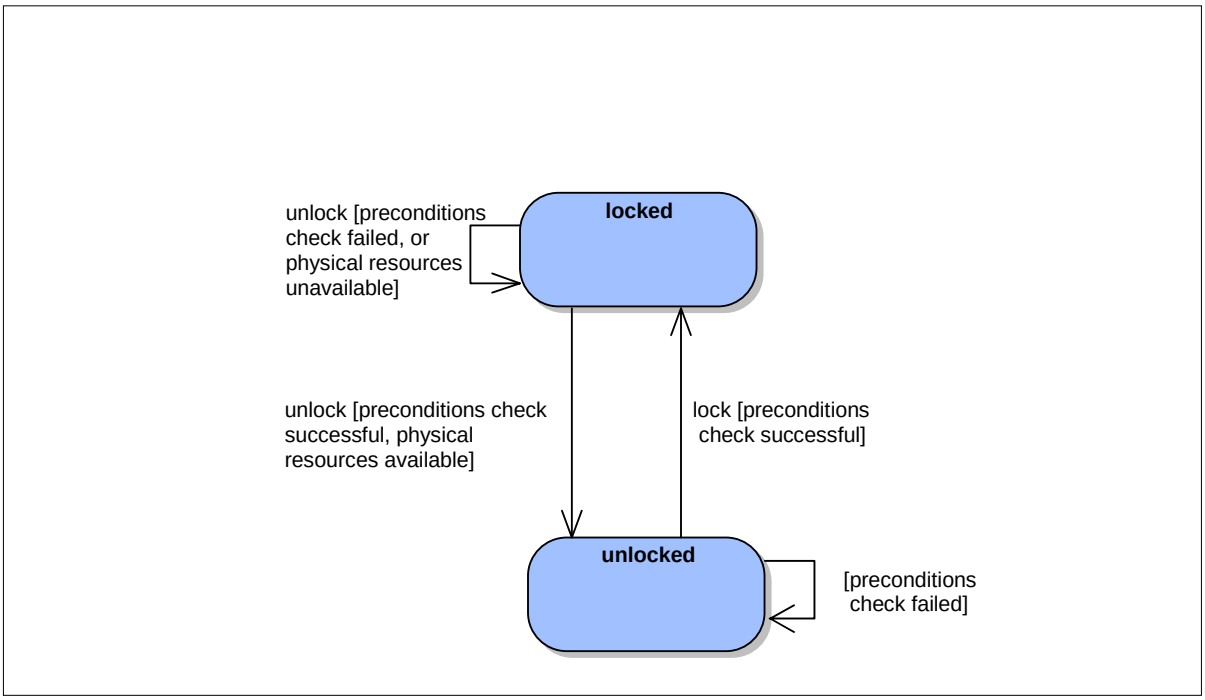

Figure 6.7.1.1-1: AST state transition diagram

Rules for AST in containments

For objects being in a containment (parent/child) relationship as defined in clause 6.5, the following default rules apply in addition to any specific preconditions to be met:

Parent has AST, children have no individual AST:

- Children may contain parameters with access type "RW-Locked"
- Children shall obey the parent's AST (e.g. for RW-Locked parameter access and object deletion)

Parent and children have an AST:

- Child transition to "unlocked" is only possible if the parent is in state "unlocked"
  (i.e. the parent shall be unlocked first before unlocking children)
- Parent transition to "lock" is only possible if none of the children are in state "unlocked"
  (i.e. children shall be locked first before the parent can lock)

Parent has no AST, children have an AST:

- No additional rules.

6.7.1.2 Functional state (FST)

This state type is used by an RE object to report its internal state. The FST is (only) modified by the RE object it belongs to. Controlling entities (e.g. the REC) can use this state type information to determine availability and health state of this object. This is used to take appropriate action, including, but not limited to, further aggregation and O&M reporting.

States:

- Pre-operational
  Object is preparing for operation but not yet able to provide the expected service. (e.g. warming up, calibrating, etc).

- Operational
  Object is fully functional.

- Degraded
  Object operates, but deviates from expected performance. An object in this state shall not trigger an update of its parameter values.
  The transition into this state is usually accompanied by a fault report.

- Failed
  Object is faulty and not operating.
  The transition into this state is usually accompanied by a fault report.

- Not operational
  Object is available, but not in operation.

- Disabled
  Object is unavailable for operation, e.g. because of not equipped hardware resources.

Rules for FST in containments:

No additional rules are defined for objects being in a containment (parent/child) relationship as defined in clause 6.5.

6.7.1.3 Relationship between Administrative and Functional states

When an object is in AST state "locked", it shall be in one of the following FST states:

- "not operational"

- "disabled"

- "failed"

AST and FST have been kept separate for two reasons:

- Different logical customers (different usage, see figure 6.7.1.3-1)
  While AST is used by the configuration management facility to control the availability of resources for configuration, the FST is used by the REC state management to determine the status of the object from a functional point of view.
  Each of the customers is only required to listen to one of these states in order to get the state information it needs, so keeping them separate does not create a multi-dimensional state vector.

- Unambiguous semantics
  If AST and FST would be merged, the "unlocked" state would be implicitly mapped to multiple FST states. This has the potential to create ambiguities, making it difficult to impossible for the REC to determine the AST. If for example the decision would be made that a failure state of a object takes precedence over anything else, no indication on the AST status would be possible any more.

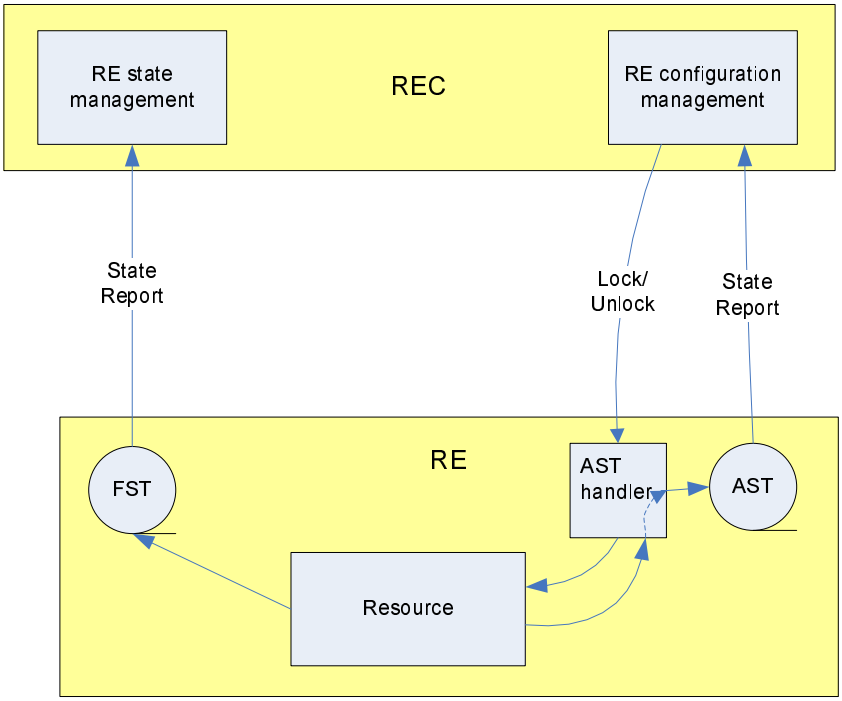

Figure 6.7.1.3-1: AST and FST relationship in RE and REC

6.7.2 State management functions

The REC manages states through the following management functions:

- OBJECT STATE REPORTING
  This function reads out the value of a given object state and reports it to the REC. Using wildcards to address the object, and to address the state type is possible. The elementary procedure defined to perform this function is specified in clause 8.5.1.

- OBJECT STATE CHANGE INDICATION
  If event-driven state reporting is enabled for a object, this function notifies the REC about any state changes for that object. The elementary procedure defined to perform this function is specified in clause 8.5.1.

- OBJECT STATE MODIFICATION
  This function modifies the value of a given object state. The function checks whether the preconditions for modifications are met (specified individually per object and per state, where applicable), and modifies the state to the requested value. Use of wildcards is not applicable for this function. The elementary procedure defined to perform this function is specified in clause 8.5.2.

6.7.3 Linkage between object state changes and fault reporting

Object state change and fault reporting are not independent of each other, but nevertheless not redundant. While object state change gives an indication of the availability of the object for operational purposes, fault reporting gives the cause of a failure or degradation.

Object state changes and faults are reported to the REC.

6.8 Fault management

Fault Management allows the REC to become aware of faults in the RE.

Each fault type is defined by the Fault ID, applicable Fault Severity levels, and applicable impacted object types. The impacted object types consist of the Primary object type, which is the source of the fault, and additional affected object types. The Fault ID, Fault Severity levels, the primary object type, and affected object types for each fault shall be as defined in table 9.2-1.

For each object instance of the primary object type, there shall be one instance of the fault type.

6.8.1 Fault States

A fault instance shall have one of the following states at any one time:

- **Active**
  the fault condition is present at the time of reporting.

- **Cleared**
  the fault condition has been corrected and does not exist anymore at the time of reporting.

A fault instance that is in state "active", that has an associated primary object that is "dynamic", shall go to state "cleared" if the object is deleted. The deletion of any additional affected objects that are "dynamic" shall not impact the fault state.

The state of a fault instance shall be independent of the administrative state of the primary object or affected objects.

The REC shall be informed by the RE for each state transition of a fault i.e. when a fault state is set to "Active" or a fault state is set to "Cleared".

6.8.2 Fault Severity

A fault instance that is in state "active" shall have one of the following severity levels at any one time:

- **Failed**
  a definite failure to provide service is associated with the occurrence of such a fault, e.g. loss of an antenna carrier, loss or significant reduction of output power.

- **Degraded**
  a degradation of performance, or a potential danger for the overall system health without immediate failure is associated with faults of this level, e.g. reduced output power, reduced Rx performance, loss of redundancy.

- **Warning**
  information about a condition which may potentially lead to a fault situation, but does not cause any degradation at time of reporting. The intention of a warning is to allow preventive action.

The REC shall be informed by the RE for a change in the severity of a current active fault by reporting the same fault ID with the new severity and fault description.

6.8.3 Fault Reporting

Faults shall be reported by the following means:

- Event-driven fault reporting
  Event-driven fault reports shall be sent by the RE to the REC only when the status of one or more of the following parameters changes for a fault instance:

  - Fault State

  - Fault Severity

  - Any of the "additional affected objects"

  A change in the underlying cause of the fault or a change in the Fault Description shall not trigger a new fault event for a fault instance if it does not lead to a change in status of one of the above parameters.

- On-demand fault report
  upon request by the REC, the RE reports a list of fault instances which are in state "active" at the time of the reporting request.

The Fault Description may be updated by the RE when it sends a new fault report, due to e.g. updated information about the underlying cause being available.

Details of the Fault Reporting procedure are specified in clause 8.6.

6.8.4 Fault History

The RE may maintain a fault history file of fault events e.g. activations and clearances.

The fault history file shall be uploadable using FTP as per the procedure defined in clause 8.8.1.

The information log into the fault history file should contain the same level of details as the event-driven reporting.

6.8.5 ORI Fault Types

The specified fault types are described in clause 9.2.

6.8.6 Handling of Fault Reports by the REC

Although the REC is not required to take any action towards the RE based on fault reports, this is not precluded. In addition, when multiple faults are triggered for the same reason, a smart REC may filter out some of the faults to focus on the essential ones causing the problem in the RE.

6.9 Performance Management

Performance management describes the measurements and counters used to collect network or network element related performance indicators for the purpose of optimizing the network. It also deals with the configuration and control of such scanners and counters.

In the present document, no common performance indicators have been defined.

Implementation-specific performance data collectors shall use the logging facility defined in clauses 8.8 and 9.1.9 to control data collection and data transfer to RE external storage.

6.10 Logging

The Logging functionality allows the RE to record information about various aspects of the system, and provides a way for the REC to upload this information from the RE in the form of transfer files using the FTP protocol. Different log types can be provided by the RE which are represented by the respective Log objects, as defined in clause 9.1.9.

6.10.1 Log concept

Conceptually (not implying a specific implementation), a log is described as a log buffer on the RE in which RE generated log data is stored until it is moved into one or more transfer files for upload to a FTP server. No assumption is made on whether this FTP server is located in the REC or anywhere else. Every instance of a log is represented in the object model by a corresponding object instance of type "Log". Every log instance is assumed to have its own log buffer and transfer file(s).

The size of an individual transfer file may be limited by the REC by means of a configuration parameter. The total size of all transfer files for a given log instance may be limited by the RE. The RE shall report this size limit via a read-only parameter defined within the Log object. The RE may in addition have a limit of the total size of transfer files across all logs which may be smaller than the sum of the individually reported RE size limits.

If a log is enabled, the RE shall collect log data in its associated log buffer until it gets triggered to generate a transfer file. A transfer file is generated by moving the content of the log buffer into the transfer file.

NOTE: This does not imply a certain implementation; concrete implementations may use a file as buffer, close it and continue writing to a different file.

Triggers for generating a transfer file are:

- The amount of log data in the log buffer reaching the REC defined size limit for transfer files.

- The amount of log data in the log buffer content reaching the RE defined size limit for transfer files.
- By an on-demand FILE UPLOAD REQUEST received from the REC while there is no transfer file existing for that log.
- When the timeout period configured by the REC (described in 6.10.4) has been reached.

When a transfer file becomes available, it shall be queued by the RE for later upload to the REC. Every log instance provides its individual upload queue. Transfer files shall be queued in first-in-first-out order (i.e. the oldest file is transferred first).

For transfer files in the upload queue, the following applies:

- New transfer files shall be added to the foremost empty position of the upload queue.
- Files shall be retrieved or removed from the first position in the queue;
  if more files exist in the queue, all remaining files shall move one position forward in the upload queue.
- If notification is enabled for that log, a FILE AVAILABLE INDICATION shall be provided to the REC every time a transfer file becomes available in the first position of the upload queue.
- Availability of one or more transfer files in the upload queue shall be indicated to the REC by setting the "File Available" parameter of the respective Log object to "TRUE". The RE shall re-set the "File Available parameter" to "FALSE" if there are no more transfer files available in the upload queue.

For retrieving files from the queue, the following applies:

- The REC retrieves the transfer file which is first in the queue using the FILE UPLOAD procedure (see clause 8.8.2).
- When the transfer file is successfully uploaded (successful completion of the FILE UPLOAD procedure):
  - the RE shall remove the uploaded transfer file from the upload queue (details see above);
  - the RE may discard the uploaded transfer file.

A log overflow is a situation which occurs when transfer files of a given log are not uploaded fast enough so that more log data is created over time in the RE than is uploaded by the REC. The overflow situation is detected by the RE when the RE size limit does not allow to create more transfer files for that log, and the log buffer cannot hold additional log entries.

The behaviour in case of a log overflow shall be configurable using the parameter "Overflow Behaviour".

- In case of Overflow Behaviour = "STOP",
  the RE stops recording, and shall set the parameter "recordingEnabled" to "FALSE".
  After the overflow situation is cleared, the REC may re-enable logging by setting the parameter "recordingEnabled" to "TRUE".
- In case of "Overflow Behaviour" = "FIFO",
  oldest log entries shall be discarded by the RE in favour of newer ones (FIFO behaviour). This shall be done by discarding the first transfer file in the upload queue. The storage space freed up is then available to the RE for creating a new transfer file.
  Note: The RE implementation should take action that a transfer file being in the process of being uploaded does not get discarded.

The default behaviour is log type specific; where applicable, it may be re-configured by the REC.

6.10.2 Log types & content

Logs may be ORI-defined or implementation-specific. For ORI defined logs and implementation specific logs the log content and data format is beyond the scope of the present document. It is not assumed that the REC takes action upon the information contained in logs.

NOTE: For ORI-defined logs, it is intended that the log content and data format is further defined in future Releases of ORI.

Examples for ORI specific logs are the fault log, and potentially any data generated by ORI defined scanners or counters. Examples for implementation specific logs are RE OS specific logs, OS crash dumps, vendor application error logs, vendor defined implementation specific KPI measurement logs, etc.

6.10.3 Log categories

Log categories are ORI defined categories that all logs (ORI defined and implementation specific) shall be associated with. The purpose is for the REC to be able to trigger upload of files of logs of a certain category without knowing the specific log type, especially in the case of implementation-specific logs. For this purpose, the REC may use the log category information contained in every log object to identify the objects belonging to a given category. One log category may contain 1 or more log types and log objects; one log type and one log object shall be associated with exactly one log category.

The log categories defined by ORI are:

- "FAULT": logs containing fault information
- "PERF": logs containing performance data records, e.g. key performance indicators (KPIs)
- "SYSTEM": logs containing records generated by the RE operation system
- "CRASH": logs generated upon process crashes ("core dumps")
- "APPLICATION": RE application related logs, e.g. application error logs, or "flight recorders"
- "DEBUG": logs generated for the purpose of RE application debugging
- "OTHER": logs fitting under none of the above categories

6.10.4 Log configuration and control

Control of the log (beyond transfer file size and upload) includes the possibility to start and stop logging for log types supporting this. Even though the concept prevents the RE file system or memory from being spilled by log data, switching on and off logging may be useful for example to minimize the runtime impact of certain loggers, or to control the point in time when data gets collected.

Immediate start/stop control is provided through the configuration parameter "Enable Recording". Setting this parameter to "TRUE" shall enable logging for the given log, setting it to "FALSE" shall disable logging. The default value for "Enable Recording" is log specific.

Apart from immediate start/stop control, it is also possible to have time-based log control. If parameter "Log Period" is configured to a value > "0", the behaviour shall be as follows:

- For Timer Type = "SINGLE_SHOT",
  - Timed logging shall be started by enabling logging (REC sets Recoding Enabled to "TRUE"), or, if logging is already enabled, the time period shall start when the logging period becomes configured.
  - After expiration of the configured logging period, the log shall logging and shall trigger preparation of a transfer file, which is further processed as defined above.
    The Recording Enabled parameter will be re-set to "FALSE" by the RE.
  - Re-enabling logging starts another time period.
- For Timer Type = "PERIODIC",
  - Timed logging shall be started by enabling logging (REC sets Recoding Enabled to "TRUE"), or, if logging is already enabled, the time period shall start when the logging period becomes configured.

- After expiration of the logging period, the log shall prepare a transfer file and re-start logging. This shall be periodically repeated until logging is stopped (REC sets Recoding Enabled to "FALSE"). Transfer files are further processed as described above.

- If REC or RE size limits are reached, additional transfer files shall be created as described in clause 6.10.1. Overflow behaviour shall be handled as described in clause 6.10.1.

In addition to controls provided by the Log object, the ORI log concept supports configuration of logs using downloaded or locally stored configuration files which can be activated as a "profile" for log types supporting this. Unless specifically defined for ORI log types, the format and content of log configuration files is implementation specific. An example for such a configuration file is the configuration file of the Apache Software Foundation "log4cxx" framework [i.1].

The log configuration activation procedure is further described in clause 8.8 of the present document.

6.10.5 Log file naming

The default file names for log transfer files shall be in the following format:

`<VendorID>_<RE Serial number>_<Log Object ID>_<Log type ID>_<Time of creation>`

where:

- `Vendor ID`
  The 3 letter vendor ID as reported in the DHCP request (clause 10.1.1)

- `RE Serial Number`
  The serial number as reported in the parameter Serial Number of the RE object (clause 9.1.1)

- `Log Object ID`
  The objID of the Log object representing the log for which the data is uploaded (clause 9.1.9)

- `Log type ID`
  The log type ID as reported in parameter logTypeID of the associated Log object

- `Time of creation`
  The timestamp of the RE time when the transfer file has been created, in format `YYYYMMDD-HHmmss`, with
  - `YYYY` being the year (4 digits)
  - `MM` being the month (2 digits)
  - `DD` being the day of the month (2 digits)
  - `HH` being the hour of the day in 24h format (2 digits)
  - `mm` being the minute of the hour (2 digits)
  - `ss` being the second of the minute (2digits)

NOTE: The transfer file is created either because the size of the log entries in the buffer reached the size limit, or because an upload had been explicitly requested by the REC. This is different from the point in time when logging actually started.

The REC may override this by providing a different destination file name in the upload procedure.

6.11 Vendor specific extensions to the resource model

The OCP protocol supports vendor specific extensions to the resource model under the preconditions stated in clause 5.4.

Extensions supported are:

- Additional vendor specific parameters to existing object types.

- Additional vendor specific object types.

6.11.1 Vendor specific parameters

Vendor-specific parameters may be added to existing ORI defined resource model object types.

The names of such parameters shall be namespace qualified by the 3 letter vendor ID as reported in the DHCP request, code 201 (see clause 10.1.1.1).

ORI also allows the possibility for the RE to report vendor specific parameters in addition to the ORI defined parameters as part of the response to Object Parameter Report Requests, where the Object ID matches the containing object type, and the parameter ID matches the parameter name or "ALL".

RECs receiving reports about vendor specific parameters as part of an (wildcarded) Object Parameter Reporting Request shall ignore that part of the response message without ignoring the rest of the message.

REs receiving Object Parameter Reporting message or Modification Requests addressing unsupported vendor specific parameters by means of (non-wildcarded) parameter references (paramRef) shall treat those as unknown parameters and respond accordingly.

6.11.2 Vendor specific object types

Vendor-specific object types may be defined, and coexist with ORI-defined object types.

The names of such objects and their parameters shall be namespace qualified by the 3 letter vendor ID as reported in the DHCP request, code 201 (see clause 10.1.1.1).

ORI allows the possibility for the RE to report vendor specific objects and their parameters in addition to the ORI defined object types as part of the response to Object Parameter Reporting Requests, where the Object ID is wildcarded and matches the vendor specific object type.

RECs receiving reports about vendor specific objects and their parameters as part of an (wildcarded) Object Parameter Request shall ignore that part of the message without ignoring the rest of the message.

REs receiving Object Parameter Reporting or Modification Requests addressing unsupported vendor specific objects by means of non-wildcarded object references (objRef) shall treat those as unknown objects and respond accordingly.

Vendor specific object types may have contained child objects, and may be contained in ORI defined or vendor specific parent objects.

6.11.3 Vendor specific fault types

Vendor specific faults types may be defined, and shall coexist with ORI-defined fault types.

The fault ID of such fault types shall be qualified by the 3-letter vendor ID as reported in the DHCP request, code 201 (see clause 10.1.1.1).

The fault severity of such fault types shall respect the values (failed, degraded, warning) specified by ORI.

The (primary or additional affected) objects shall be objects either specified within ORI, or objects specified by the vendor (see clause 6.11.2).

ORI allows the possibility for the RE to report vendor specific faults in addition to the ORI defined faults as part of the response to Fault Reporting Requests, if the Object ID included in the request matches the primary object of the vendor specific fault type.

REC's receiving vendor specific faults reports shall ignore them if not supported, without ignoring ORI-defined faults.

7 OCP format and encoding structure

7.1 Roles of REC/RE

OCP is designed as a client-server protocol, with the RE being in the server role, and the REC being in the client role. In the current version of the present document, only a single client (REC) is foreseen; multi-client handling is foreseen as an option for later extensions to the present document.

7.2 Message format and encoding

7.2.1 Message types/handling

The following message types are supported:

- Request:
  A Request is sent by the REC to the RE to initiate an activity on the RE.

- Response:
  A Response is the reply of the RE towards the REC for a given request. It bears information about success or failure of the triggered activity, and any result data where applicable.

- Spontaneous Indication:
  A Spontaneous Indication is a message sent by the RE to the REC without being previously solicited by the REC.

The following message exchange patterns are supported:

- Synchronous completion (single request response)
  A request/response pair is used by the REC to initiate an activity on the RE, and to receive confirmation of its completion and the associated result. The implementation shall ensure that all activities triggered by the request are completed when the associated response is sent. The RE shall be ready to execute new requests by then.
  The following rules apply:

  - The Request shall be replied with a Response in less than the timeout defined by the TCP Link Monitoring Timeout T_{TLM} (see clauses 8.1.1 and 10.1.1).

  - Only one Request of one elementary procedure shall be a processed by the RE at a time; the response shall only be sent when the requested operation is completed (synchronous completion).

  - The REC shall only send a new request to a given RE once it has received the Response to the previous Request. This shall apply for new Requests belonging to the same or different elementary procedure as the previous Request.

- Asynchronous completion
  Activities triggered by the REC may take such long time that the blocking duration in the synchronous case may cause unwanted side effects (e.g. a file download). Such activities can be handled through asynchronous completion. Here the REC triggers the activity with a request message; the RE confirms the general execution possibility with an associated response. The actual execution of the activity is then done in the background, while the RE is able to execute other requests. The completion of the activity is confirmed to the REC using a spontaneous indication.
  The following rules apply:

  - The Request shall be replied with a Response in less than the timeout defined by TCP Link Monitoring Timeout T_{TLM} (see clauses 8.1.1 and 10.1.1).

  - The REC shall only send a new request to a given RE once it has received the Response to the previous Request.

- There shall be only one asynchronous completion process for one elementary procedure on-going in the RE at a given time.
- Once a trigger for an asynchronous completion type activity has been confirmed by the RE (Response with result "success" has been sent), the RE shall ensure that a completion indication is sent to the REC, also in case of failure.

- One-way spontaneous indications
  Spontaneous indications are used by the RE to report events back to the REC. Examples are fault reports and state changes.

7.2.2 Encoding basis

XML [7] is used as basis for wire encoding of messages. XML Schema Definition (XSD, [5]) is used to formally describe the structure and content of messages. All characters (per the definition in [7], clause 2.2) used in messages shall be encoded in UTF-8 [6].

To avoid rendering issues in trace tools, only printable characters from the Basic Latin character set (Unicode range U+0020 … U+007E, formerly known as ASCII 7-bit) shall be used.

7.2.3 Data types

The data types used to specify parameter types are defined in the XML schema data type specification [5], Part 2. XML data type definitions are used for specification of parameter types to avoid mapping ambiguities when translating into XML encoded messages.

The following tables describe the used data types, and the ways in which they may be restricted in their usage if necessary. This set of restrictions is used in the detailed message content descriptions for range, length or other limitations. Note that for XML native data types the description is just a short summary of definitions in [5], Part 2.

Table 7.2.3-1: XML native data types

Data type	Description	Qualifiers (Restrictions) / Comments
`String`	A set of characters	`length`: Restricts to an exact fixed length `minLength`: Restricts to a minimum length `maxLength`: Restricts to a maximum length `enumeration`: Restricts the set of possible strings which are valid for a given element. A length range can be defined by using `minLength` and `maxLength` (with `minLength < maxLength`). The `enumeration` qualifier can be used to define enumerated value sets.
`dateTime`	Date, time and optionally time zone in the form `yyyy '-' mm '-' dd 'T' hh ':' mm ':' ss ('.' s+)? (zzzzzz)?`	Example: `2012-01-18T19:45:00+01:00` represents 18-Jan-2012, 19:45:00, time zone UTC+1 (= CET) The time zone indicator is optional. The (legal) time value 24:00:00 shall not be used.
`base64Binary`	Represents Base64-encoded arbitrary binary data according to encoding rules defined in RFC 2045 [8]	The canonical form (no blanks between trailing "=" characters, no line breaks in encoded character stream) shall be used.
`hexBinary`	Represents arbitrary hex-encoded binary data. Valid values are formed by pairs of hexadecimal numbers representing binary octets.	length: Restricts to an exact fixed length minLength: Restricts to a minimum length maxLength: Restricts to a maximum length Length restrictions are interpreted in octets of binary data; for example, the hexBinary sequence “F0” has the length 1.
`Decimal`	Decimal value, not limited but of finite length	`totalDigits`: Restricts the number of digits `fractionDigits`: allows to specify a fixed number of (decimal) digits to be interpreted as fractional digits `min`: minimum value this element can have (translates to XML `minInclusive`) `max`: maximum value this element can have (translates to XML `maxInclusive`) The canonical form (mandatory decimal point, no leading or trailing zeros other than necessary to have at least one digit before and after the decimal point, no "+" sign for positive numbers including 0.0) shall be used.
`Integer`	Integer value, not limited but of finite length	`totalDigits`: Restricts the number of digits `min`: minimum value this element can have (translates to XML `minInclusive`) `max`: maximum value this element can have (translates to XML `maxInclusive`) The canonical form (no leading zeros, no "+" sign for positive numbers including 0) shall be used.
`nonNegativeInteger`	Integer value ≥ 0, not limited but of finite length	See integer, but further restrictions shall be within the bounds given by the value range. The canonical form (no leading zeros, no "+" sign) shall be used.
`Int`	32 bit encode-able integer, value range -2^{31} … $(+2^{31} - 1)$ (-2147483648 … +2147483647)	See integer, but further restrictions shall be within the bounds given by the value range. The canonical form (no leading zeros, no "+" sign for positive numbers including 0) shall be used.

Data type	Description	Qualifiers (Restrictions) / Comments
`Short`	16 bit encode-able integer, value range -2^{15} … $(+2^{15} - 1)$ (-32728 … +32727)	See integer, but further restrictions shall be within the bounds given by the value range. The canonical form (no leading zeros, no "+" sign for positive numbers including 0) shall be used.
`Byte`	8 bit encode-able integer, value range -2^{7} … $(+2^{7} - 1)$ (-128 … +127)	See integer, but further restrictions shall be within the bounds given by the value range. The canonical form (no leading zeros, no "+" sign for positive numbers including 0) shall be used.
`unsignedInt`	32 bit encode-able positive integer, value range 0 … $(+2^{32} - 1)$ (0 … 4294967295)	See integer, but further restrictions shall be within the bounds given by the value range. The canonical form (no leading zeros, no "+" sign) shall be used.
`unsignedShort`	16 bit encode-able positive integer, value range 0 … $(+2^{16} - 1)$ (0 … 65535)	See integer, but further restrictions shall be within the bounds given by the value range. The canonical form (no leading zeros, no "+" sign) shall be used.
`unsignedByte`	8 bit encode-able positive integer, value range 0 … $(+2^{8} - 1)$ (0 … 256)	See integer, but further restrictions shall be within the bounds given by the value range. The canonical form (no leading zeros, no "+" sign) shall be used.
`Float`	32 bit encode-able float value. For details regarding value range refer to [5], Part 2.	`min`: minimum value this element can have (translates to XML `minInclusive`) `max`: maximum value this element can have (translates to XML `maxInclusive`) Examples for lexical representations of float numbers: -1E4, 1.964E3, 18, 0E0 The canonical form shall be used: • the exponent shall be indicated by "E" • Leading zeroes and the preceding optional "+" sign are prohibited in the exponent • If the exponent is zero, it shall be indicated by "E0" • For the mantissa, the preceding optional "+" sign is prohibited and the decimal point is required • Leading and trailing zeroes are prohibited subject to the following: number representations shall be normalized such that there is a single digit which is non-zero to the left of the decimal point and at least a single digit to the right of the decimal point unless the value being represented is zero • The canonical representation for zero is 0.0E0
`Boolean`	Boolean value, value range {`true, false`}	The canonical form (no "0" or "1", just "true" and "false") shall be used.

Table 7.2.3-2: Data types for RE resource model object/parameter addressing

Data type	Description	Qualifiers (Restrictions) / Comments
`objRefT`	A reference to an object instance within the ORI resource model with no wildcard (see definition in clause 6.4).	Reference to an object instance within the ORI resource model as defined in clause 6.4, no wildcards allowed Example: `"ori::TxSigPath_UTRAFDD:0"`
`objRefWcT`	A reference to an object instance within the ORI resource model; wildcard allowed (see definition in clause 6.4).	Reference to an object instance within the ORI resource model as defined in clause 6.4, with wildcards allowed Examples: `"ALL", "aisgPort:0/ALL"`
`objTypeRefT`	A reference to an object type (see definition in clause 6.4).	Reference to an object type within the ORI resource model as defined in clause 6.4 Example: `"TxSigPath_UTRAFDD"`
`paramRef`	A reference to a parameter within an object (see definition in clause 6.6).	Reference to a parameter within an object, as defined in clause 6.6 Example: `"maxTxPwr"` (from Tx Signal Path object)
`paramRefWc`	A reference to a parameter within an object; wildcard allowed (see definition in clause 6.6).	Reference to a parameter within an object, as defined in clause 6.6 Example: `"ALL"`

Simple data types can be aggregated into more complex data types, like arrays (XML: "lists") or structured data types. As described in the previous clause, elements of a parameter consisting of a complex type cannot be addressed individually. If such is required, child objects as described in clause 6.5 should be considered.

7.2.4 Name spaces

The name space for native XML Schema defined data elements is http://www.w3.org/2001/XMLSchema. This namespace will be referred to by the tag "`xsd`".

The namespace structure for ORI namespaces shall be: http://uri.etsi.org/ori/<specification>/<version>

Table 7.2.4-1 defines the details of the above structure.

Table 7.2.4-1: Description of ORI namespace details

Namespace element	Description
http://uri.etsi.org/ori	Assigned to ISG ORI by ETSI Secretariat for use with all ORI specifications.
<specification>	Number of the ISG ORI specification to which the name-space applies.
<version>	Version of the ISG ORI specification where the name-space has been initially defined. The name-space shall apply to this version and all subsequent versions of the specification that precede the version where the namespace is next modified.

The namespace assigned to this version of the present document is http://uri.etsi.org/ori/002-2/v1.1.1. This namespace will be referred to by the tag "`ori`". This is the default name space; addressing elements of this namespace within messages does not require a namespace qualifier (for the XML schema definition it is still needed though).

Elements not belonging to name space "`ori`" shall be name space qualified.

The associated schema fragment is:

```
<xsd:schema
    xmlns:ori="http://uri.etsi.org/ori/002-2/v1.1.1"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://uri.etsi.org/ori/002-2/v1.1.1" elementFormDefault="qualified">
  <!—schema content goes here -->
<xsd:schema/>
```

7.2.5 Message structure

Every message consists of a header and a body. The header information shall be the same for all messages, while the information contained in the message body depends on the individual message.

7.2.5.1 Message Header

The message header is defined as follows:

Element	Type	Description
msgType	`ori:msgTypeT`	msgType indicates whether a message is a request, a response or a spontaneous indication. It shall help message handlers to route messages prior to body inspection.
msgUID	`xsd:unsignedShort`	msgUID is a unique ID per message. It shall be set by the REC and be different per message sent (usually incremented). The RE shall copy the value received in the Request message into any related Response or Completion Indication message, but not interpret this value otherwise. This is intended to help the REC to associate responses and indications to their originating request. The UID shall be in the range 1 … 65535; the value 0 shall be used for one-way spontaneous indication messages (ie. which are not related to a specific request/response pair), unless explicitly specified. Receiving the value 0 means that the message cannot be associated to any preceding Request message.

With `ori:msgTypeT` defined as:

Element	Type	Description
ori:msgTypeT	enumeration	Valid values are: "REQ", "RESP", "IND"

The associated XML Schema fragment is:

```
<!-- Define message type enumeration -->
    <xsd:simpleType name="msgTypeT">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="REQ"/>
            <xsd:enumeration value="RESP"/>
            <xsd:enumeration value="IND"/>
        </xsd:restriction>
    </xsd:simpleType>

    <!-- ORI message header -->
    <xsd:complexType name="msgHeaderT">
        <xsd:sequence>
            <xsd:element name="msgType" type="ori:msgTypeT"/>
            <xsd:element name="msgUID" type="xsd:unsignedShort"/>
        </xsd:sequence>
    </xsd:complexType>
```

Example for a message header:

```
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>4711</msgUID>
    </header>
    <body>
        <!—- body content goes here -->
    </body>
</msg>
```

7.2.5.2 Message body

Message bodies are individual per message, and are specified in clause 8 (OCP Elementary Functions and Messages). The body of a message contains exactly one message at a time.

All "Response" type message bodies shall contain a "result" information element which indicates the result of the previous "Request" message. Result codes common to all Response messages are defined in clause 7.4.

The associated XML Schema fragment of a message body is:

```
 <!--  ORI message body -->
<xsd:complexType name="msgBodyT">
   <xsd:choice>
      <xsd:element name="healthCheckReq" type="ori:healthCheckReqT"/>
      <xsd:element name="healthCheckResp" type="ori:healthCheckRespT"/>
      <xsd:element name="getVersionReq" type="ori:getVersionReqT"/>
      <xsd:element name="getVersionResp" type="ori:getVersionRespT"/>
      <!-- additional messages go here -->
   </xsd:choice>
</xsd:complexType>
```

7.2.5.3 OCP message

Using header and body definition, the XML Schema for a complete message is

```
 <!-- ORI message -->
<xsd:element name="msg">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="header" type="ori:msgHeaderT"/>
         <xsd:element name="body" type="ori:msgBodyT"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

7.3 Transport protocol embedding

As mentioned in clause 5.2.4, TCP is used as transport protocol for OCP. TCP is a reliable, stream-oriented protocol, no provisions for packet splitting/re-assembly or packet re-ordering need to be done. The TCP transport stream is bi-directional, one connection is sufficient to handle all OCP messaging between REC and RE, independent of message exchange pattern and direction.

7.3.1 Message framing

Message framing is necessary to recognize the beginning and the end of an individual OCP message in the transport stream.

Messages are enclosed in matching pairs of `<msg [doc parameters]> … </msg>` delimiters, with `<msg [doc parameters]>` being the "message start" delimiter and `</msg>` being the "message end" delimiter. The design of XML ensures that these are unambiguous independent of the message content. These sequences shall be used as message delimiters.

Framing error handling on the RE side:

- Second "message start" delimiter received without first "message end" delimiter:
  The content of the message fragment between the first "message start" delimiter and the second "message start" delimiter shall be discarded. A DEFAULT FAILURE RESPONSE message (see clause 7.5) with the *Result* IE set to "FAIL_FRAME_ERROR" shall be submitted. Message processing shall resume with the second "message start" delimiter.

- Second "message end" delimiter received without second "message start" delimiter:
  The content of the message fragment between the first "message end" and the second "message end" delimiter shall be discarded. A DEFAULT FAILURE RESPONSE message (see clause 7.5) with the *Result* IE set to "FAIL_FRAME_ERROR" shall be submitted. Message processing shall resume when receiving a "message start" delimiter.

- Any spurious data received between "message end" and subsequent "message start" delimiter:
  The data received shall be discarded. Message processing shall resume when receiving a "message start" delimiter.

- No "message end" delimiter received after a "message start" delimiter has been received:
  This causes a link timeout; covered by the procedure described in clause 10.1.1.

7.4 Common default result codes

Table 7.4-1 defines result codes which are common to all synchronous elementary procedures. Additional elementary procedure-specific failure codes are defined in the elementary message descriptions in clause 8.

Table 7.4-1: Common result codes for synchronous elementary procedures

Code	Description
SUCCESS	Indicates that a message was successfully decoded and the associated activity has been successfully executed.
FAIL_SYNTAX_ERROR	Indicates that decoding of a message failed because it was generally malformed / corrupted. The DEFAULT FAILURE RESPONSE message can be used to communicate this failure if the initial request type could not be decoded; otherwise the normal response shall be sent with this failure code in the *Result* IE.
FAIL_UNRECOGNIZED_MESSAGE	Indicates that a message could be successfully decoded, but the message body contains an unknown command. The DEFAULT FAILURE RESPONSE message shall be used to communicate this failure.
FAIL_RE_BUSY	Indicates that the RE cannot handle the incoming message 1) due to another elementary procedure previously triggered by REC still being processed (request message received from REC, but response message not yet sent) 2) due to the new incoming message being for an elementary procedure requiring asynchronous completion, and where another instance of the same elementary procedure is already ongoing; 3) due to other congestion situations (RE not ready to process new elementary procedures/messages triggered by REC for internal reasons); the DEFAULT FAILURE RESPONSE shall be used to communicate this failure in that case. For 1) and 2), the normal response message of the elementary procedure causing the issue shall be sent with the failure code in the *Result* IE set to FAIL_RE_BUSY. For 1) and 3), the originating request shall be re-submitted not earlier than $T_{HCI}/2$ to avoid overloading the RE. For 2), the originating request may be re-submitted immediately after the completion indication of the previously on-going procedure has been received; otherwise the originating request shall be re-submitted not earlier than $T_{HCI}/2$ to avoid overloading the RE.
FAIL_MISSING_PARAMETER	Indicates that a message could be successfully decoded, but a mandatory parameter was missing.
FAIL_PARAMETER_ERROR	Indicates that a message could be successfully decoded, but a parameter value is in wrong format (e.g. an alphanumeric string where a number was expected), or the value is out of range.

These default result codes are referred to as `ori:resT`

Associated XML schema:

```
<!--
        Common result codes for all messages.
        Per-message specific result codes are added within message
```

```
        definition blocks
-->
<xsd:simpleType name="resT">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="SUCCESS"/>
        <xsd:enumeration value="FAIL_SYNTAX_ERROR"/>
        <xsd:enumeration value="FAIL_UNRECOGNIZED_MESSAGE"/>
        <xsd:enumeration value="FAIL_RE_BUSY"/>
        <xsd:enumeration value="FAIL_MISSING_PARAMETER"/>
        <xsd:enumeration value="FAIL_PARAMETER_ERROR"/>
  </xsd:restriction>
</xsd:simpleType>
```

7.5 Default failure response message

For certain types of messaging fault conditions, it may not be possible for the RE to generate a response message matching the initial request. In such cases, the RE shall inform the REC informed about the failure even when the original request could not be processed by the RE using the message DEFAULT FAILURE RESPONSE.

This message shall contain the IE *Result* set to an appropriate code from the following subset:

- FAIL_SYNTAX_ERROR (usage defined in clause 7.4)
- FAIL_UNRECOGNIZED_MESSAGE (usage defined in clause 7.4)
- FAIL_RE_BUSY (usage defined in clause 7.4)
- FAIL_FRAME_ERROR
  This code shall be used when a message framing error has been encountered. See clause 7.3.1 for a more detailed definition.

The header IE `msgUID` shall be set to the value received in the incoming malformed message if the RE has been able to decode it, or to 0 otherwise.

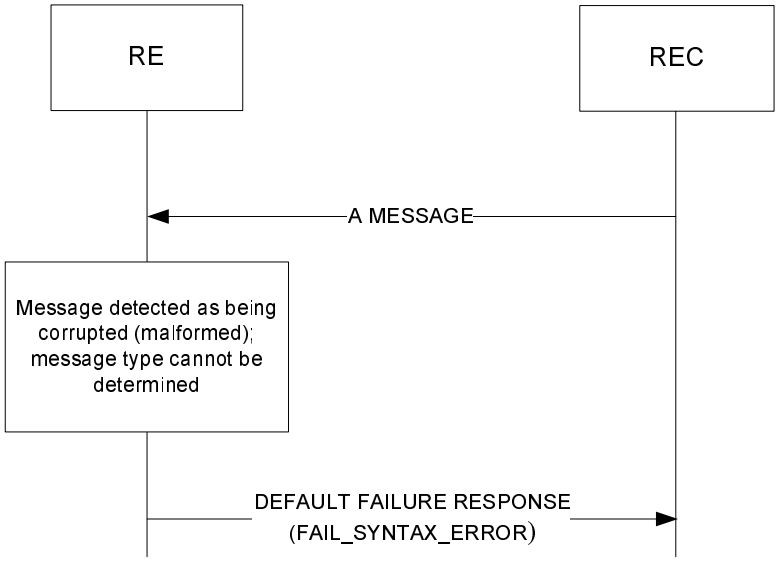

Figure 7.5-1: Default failure response upon malformed message

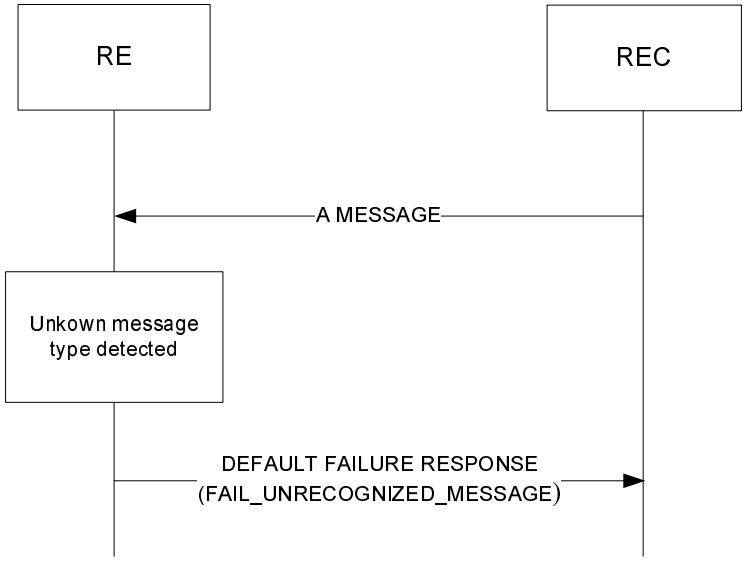

Figure 7.5-2: Default failure response upon unknown message type

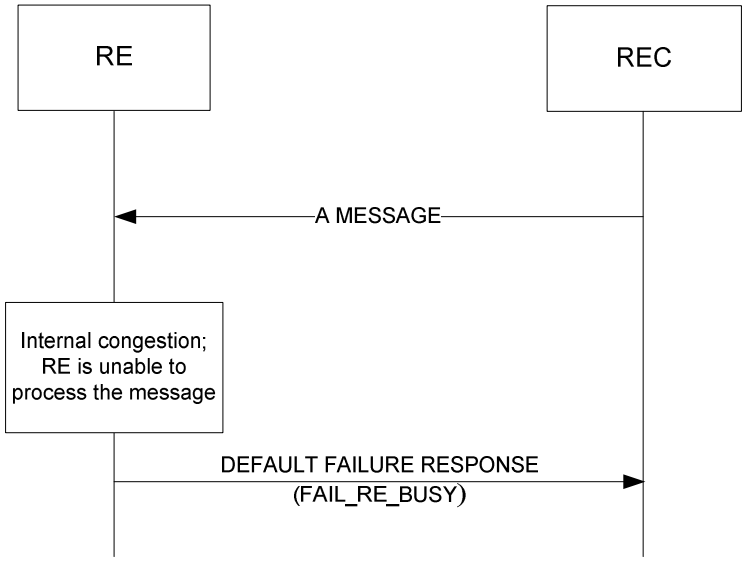

Figure 7.5-3: Default failure response upon internal congestion

Message	Encoded name
DEFAULT FAILURE RESPONSE	`defaultFailResp`

7.5.1 Message parameter details

Message parameter details for DEFAULT FAILURE RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Valid values: "FAIL_SYNTAX_ERROR", "FAIL_UNRECOGNIZED_MESSAGE", "FAIL_RE_BUSY" "FAIL_FRAME_ERROR"

7.5.2 Message encoding

XML Schema of message body:

```
   <!-- Message specific result codes -->
   <xsd:simpleType name="defaultFailRespResT">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="FAIL_SYNTAX_ERROR"/>
         <xsd:enumeration value="FAIL_UNRECOGNIZED_MESSAGE"/>
         <xsd:enumeration value="FAIL_RE_BUSY"/>
         <xsd:enumeration value="FAIL_FRAME_ERROR"/>
</xsd:restriction>
   </xsd:simpleType>

   <!-- Message Body: Default Failure Response -->
   <xsd:complexType name="defaultFailRespT">
      <xsd:sequence>
         <xsd:element name="result" type="ori:defaultFailRespResT"/>
      </xsd:sequence>
   </xsd:complexType>
```

Message examples:

```
    <!-- Example: Default Failure Response upon malformed message
         (msgUID recognized)
    -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>RESP</msgType>
            <msgUID>12345</msgUID>
        </header>
        <body>
            <defaultFailResp>
                <result>FAIL_SYNTAX_ERROR</result>
            </defaultFailResp>
        </body>
    </msg>

    <!-- Example: Default Failure Response upon unknown message type -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>RESP</msgType>
            <msgUID>12345</msgUID>
        </header>
        <body>
            <defaultFailResp>
                <result>FAIL_UNRECOGNIZED_MESSAGE</result>
            </defaultFailResp>
        </body>
    </msg>

    <!-- Example: Default Failure Response upon internal congestion -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>RESP</msgType>
            <msgUID>0</msgUID>
        </header>
        <body>
            <defaultFailResp>
                <result>FAIL_RE_BUSY</result>
            </defaultFailResp>
        </body>
    </msg>
```

8 OCP Elementary Functions and Messages

8.1 Device management

8.1.1 Health Check

The Health Check procedure allows the REC to verify that the OCP layer is functioning correctly at the RE, and is intended to be performed both at OCP layer establishment and throughout the lifetime of the established OCP layer between REC and RE. See figure 8.1.1-1.

The HEALTH CHECK REQUEST message shall include the following IEs: *TCP Link Monitoring Timeout*.

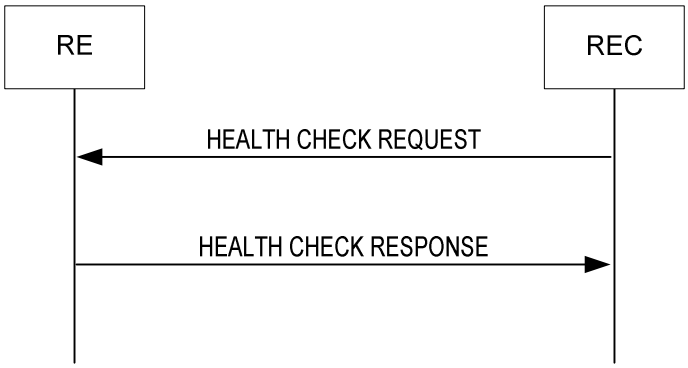

Figure 8.1.1-1: Health Check procedure

8.1.1.1 Success

When the RE receives the HEALTH CHECK REQUEST message from the REC, it shall store the value of the *TCP Link Monitoring Timeout* IE, and re-start the TCP Link Monitoring timer using the new timeout value.

NOTE: Also see clause 10.1.1 for usage of the TCP Link Monitoring timer by the RE for OCP layer supervision, and required actions upon timer expiry.

The RE shall consider setting of the link monitoring timeout as failed if the format or value of *TCP Link Monitoring Timeout* IE is out of range. In this case the timeout value of the TCP Link Monitoring Timer shall be left unmodified. and the TCP Link Monitoring Timer shall be re-started.

Then the RE shall respond with a HEALTH CHECK RESPONSE message and shall include the *Result* IE, with the value set to "success".

8.1.1.2 Failure

In case the HEALTH CHECK REQUEST message was received by the RE, with the exception of common failure causes, as specified in clause 7.4, this procedure shall never fail.

In case the HEALTH CHECK REQUEST message was not received by the RE, the required RE actions are specified in clause 10.1.1.

8.1.1.3 Message Parameter Details

Message	Encoded name
HEALTH CHECK REQUEST	`healthCheckReq`
HEALTH CHECK RESPONSE	`healthCheckResp`

Message parameter details for HEALTH CHECK REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
TCP Link Monitoring Timeout	tcpLinkMonTimeout	1	unsignedShort	Unit: seconds See 10.1.1 for detailed description

Message parameter details for HEALTH CHECK RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. (see note)
NOTE: FAIL_PARAMETER_ERROR occurs if tcpLinkMonTimeout is out of range.				

8.1.1.4 Message encoding

XML Schema of message bodies:

```
<!-- Message Body: Health Check Request -->
<xsd:complexType name="healthCheckReqT">
   <xsd:sequence>
      <xsd:element name="tcpLinkMonTimeout">
         <xsd:simpleType>
            <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction>
         </xsd:simpleType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!-- Message Body: Health Check Response -->
<xsd:complexType name="healthCheckRespT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:resT"/>
   </xsd:sequence>
</xsd:complexType>
```

Message examples (line breaks and formatting for readability only):

```
<!-- Example: Health Check Request -->
 <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
     <header>
         <msgType>REQ</msgType>
         <msgUID>1234</msgUID>
     </header>
     <body>
         <healthCheckReq>
             <tcpLinkMonTimeout>255</tcpLinkMonTimeout>
         </healthCheckReq>
     </body>
 </msg>

 <!-- Example: Health Check Response -->
 <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
     <header>
         <msgType>RESP</msgType>
         <msgUID>1234</msgUID>
     </header>
     <body>
         <healthCheckResp>
             <result>SUCCESS</result>
         </healthCheckResp>
     </body>
 </msg>
```

8.1.2 Set Time

The Set Time procedure allows the REC to set/update the absolute time reference that shall be used by the RE. The procedure is executed as part of the initial alignment procedure (see clause 10.1.2), and shall be repeated at least once every 24 hours to keep the time difference between REC and RE within reasonable bounds.

The SET TIME REQUEST message shall include the following IEs: *New Time*.

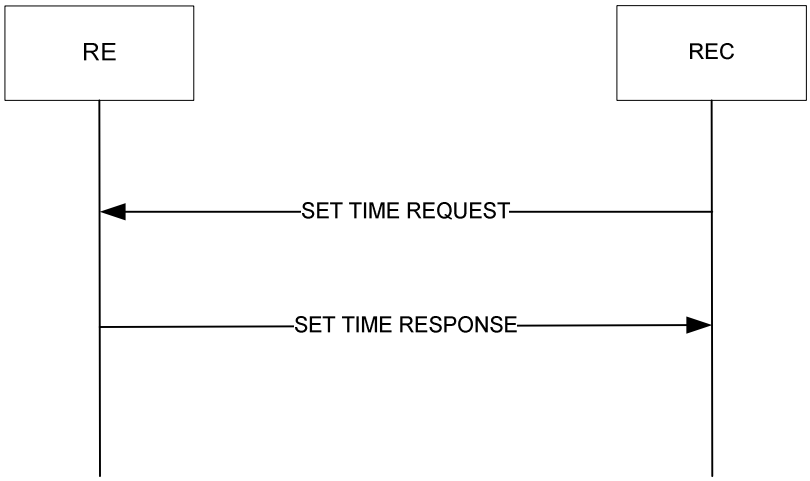

Figure 8.1.2-1: Set Time procedure

8.1.2.1 Success

When the RE receives the SET TIME REQUEST message, it shall use the value of the *New Time* IE to set the new time reference (in Year, Month, Day, Hour, Minute, Second, and Time Zone if this was included in the *New Time* IE), and delete any existing time reference that is being used.

The RE shall then respond with a SET TIME RESPONSE message, and shall include the *Result* IE, with the value set to "success".

8.1.2.2 Failure

The RE shall consider the procedure as failed and send a SET TIME RESPONSE message including the *Result* IE with the value set to to one of the failure cause values listed below in the following cases:

- `FAIL_INVALID_TIMEDATA` if the new time reference could not be applied by the RE.

8.1.2.3 Message parameter details

Message	Encoded name
SET TIME REQUEST	`setTimeReq`
SET TIME RESPONSE	`setTimeResp`

Message parameter details for SET TIME REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
New Time	newTime	1	dateTime	

Message parameter details for SET TIME RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. + FAIL_INVALID_TIMEDATA

8.1.2.4 Message encoding

XML Schema of message bodies:

```
<!-- Set Time Resp specific failure codes -->
   <xsd:simpleType name="setTimeRespResT">
      <xsd:union memberTypes="ori:resT">
         <xsd:simpleType>
            <xsd:restriction base="xsd:string">
               <xsd:enumeration value="FAIL_INVALID_TIMEDATA"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:union>
   </xsd:simpleType>

   <!-- Message Body: Set Time Request -->
   <xsd:complexType name="setTimeReqT">
      <xsd:sequence>
         <xsd:element name="newTime" type="xsd:dateTime"/>
      </xsd:sequence>
   </xsd:complexType>

   <!-- Message Body: Set Time Response -->
   <xsd:complexType name="setTimeRespT">
      <xsd:sequence>
         <xsd:element name="result" type="ori:setTimeRespResT"/>
      </xsd:sequence>
   </xsd:complexType>
```

Message examples:

```
    <!-- Example: Set Time Request -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>REQ</msgType>
            <msgUID>1235</msgUID>
        </header>
        <body>
            <setTimeReq>
                <newTime>2012-04-26T10:23:00</newTime>
            </setTimeReq>
        </body>
    </msg>

    <!-- Example: Set Time Response -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>RESP</msgType>
            <msgUID>1235</msgUID>
        </header>
        <body>
            <setTimeResp>
                <result>SUCCESS</result>
            </setTimeResp>
        </body>
    </msg>
```

8.1.3 RE Reset

The RE Reset procedure allows the REC to reset a specific RE. This type of reset differs from the ORI Low Layer Reset in [3] by allowing the RE to orderly shut down prior to the reset.

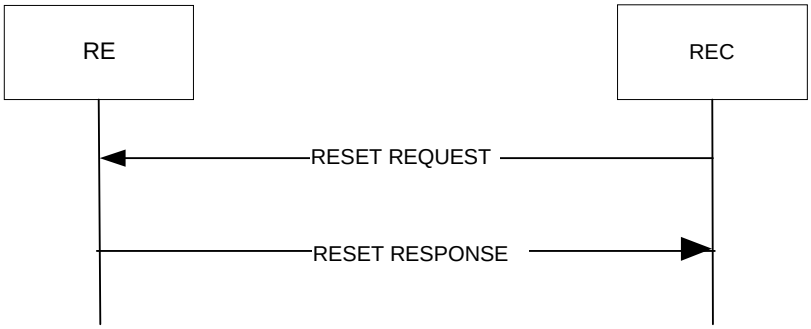

Figure 8.1.3-1: RE Reset procedure

8.1.3.1 Success

When the RE receives the RESET REQUEST, it shall transmit a RESET RESPONSE message to the REC, and perform a reset of itself.

Once the RE has reset itself, it shall trigger the OCP layer establishment and RE start-up procedure defined in clauses 10.1.1 and 10.1.2 respectively.

8.1.3.2 Failure

With the exception of common failure causes, as specified in clause 7.4, this procedure shall never fail.

8.1.3.3 Message parameters

Message	Encoded name
RESET REQUEST	`resetReq`
RESET RESPONSE	`resetResp`

Message parameter details for RESET RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4.

8.1.3.4 Message encoding

XML Schema of message bodies:

```
<!-- Message body: RE Reset Request -->
<xsd:complexType name="resetReqT">
   <!-- no arguments -->
</xsd:complexType>

<!-- Message body: RE Reset Response -->
<xsd:complexType name="resetRespT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:resT"/>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
<!-- Example: Reset Request -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>2345</msgUID>
    </header>
    <body>
        <resetReq/>
    </body>
</msg>
```

```
<!-- Example: Reset Response -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>2345</msgUID>
    </header>
    <body>
        <resetResp>
            <result>SUCCESS</result>
        </resetResp>
    </body>
</msg>
```

8.2 Software management

8.2.1 Version Query

The Version Query procedure allows the REC to request and acquire from the RE a report of the product information, hardware version, and software images that are available in the RE.

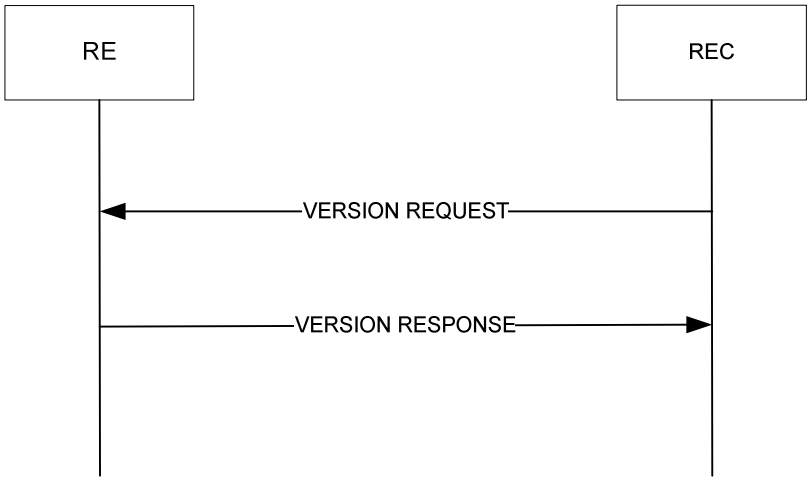

Figure 8.2.1-1: Version Query procedure

8.2.1.1 Success

When the RE receives the VERSION REQUEST message, it shall respond with a VERSION RESPONSE message, containing the following information:

- The *Vendor ID* IE containing the same value as signalled in DHCP code 201, as defined in clause 10.1.1.1, the *Product ID* IE, the *Product Revision* IE, the *Serial Number* IE and the *Hardware version* IE. This is vendor-specific information of the RE hardware.
- The *Software Upgrade Package Version Identifier for Active Software Image* IE.
- The *Active Software Image Version* IE, as a text string.
- The *Software Upgrade Package Version Identifier for Passive Software Image* IE, if available, as a text string.
- The *Passive Software Image Version* IE, if available, as a text string.

NOTE: The details of the *Active Software Image Version* IE and *Passive Software Image Version* IE are RE vendor-specific and the REC is not expected to interpret the text strings.

- The *Result* IE with the value set to "success".

8.2.1.2 Failure

With the exception of common failure causes, as specified in clause 7.4, this procedure shall never fail.

8.2.1.3 Message parameter details

Message	Encoded name
VERSION REQUEST	getVersionReq
VERSION RESPONSE	getVersionResp

Message parameter details for VERSION RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4.
Vendor ID	vendorID	1	String, Length = 3	See 10.1.1.1
Product Revision	productID	1	String	
Serial Number	productRev	1	String	
Hardware version	hardwareVer	1	String	
Software Upgrade Package Version Identifier for Active Software Image	activeSwUpgradePkgVer	1	String	
Active Software Image Version	activeSwImgVer	1	String	
Software Upgrade Package Version Identifier for Passive Software Image	passiveSwUpgradePkgVer	0..1	String	
Passive Software Image Version	passiveSwImgVer	0..1	String	

8.2.1.4 Message encoding

XML Schema of message bodies:

```
-       <!-- Vendor ID data type -->
      <xsd:simpleType name="vendorIdT">
        <xsd:restriction base="xsd:string">
          <xsd:length value="3"/>
        </xsd:restriction>
      </xsd:simpleType>

      <!-- Message body: Version Report Request -->
      <xsd:complexType name="getVersionReqT">
        <!-- no arguments -->
      </xsd:complexType>

      <!-- Message body: Version Report Response -->
      <xsd:complexType name="getVersionRespT">
        <xsd:sequence>
          <xsd:element name="result" type="ori:resT"/>
          <xsd:element name="vendorID" type="ori:vendorIdT"/>
          <xsd:element name="productID" type="xsd:string"/>
          <xsd:element name="productRev" type="xsd:string"/>
          <xsd:element name="serialNumber" type="xsd:string"/>
          <xsd:element name="hardwareVer" type="xsd:string"/>
          <xsd:element name="activeSwUpgradePkgVer" type="xsd:string"/>
          <xsd:element name="activeSwImgVer" type="xsd:string"/>
          <xsd:element name="passiveSwUpgradePkgVer" type="xsd:string"

-               minOccurs="0"/>
          <xsd:element name="passiveSwImgVer" type="xsd:string"
```

```
-               minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
```

Message examples:

```
<!-- Example: Get Version Request -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>3456</msgUID>
    </header>
    <body>
        <getVersionReq/>
    </body>
</msg>

<!-- Example: Get Version Response -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>3456</msgUID>
    </header>
    <body>
        <getVersionResp>
            <result>SUCCESS</result>
            <vendorID>xyz</vendorID>
            <productID>acme0815</productID>
            <productRev>A.01</productRev>
            <serialNumber>201200001</serialNumber>
            <hardwareVer>025-alpha</hardwareVer>
            <activeSwUpgradePkgVer>RAN-201204-007</activeSwUpgradePkgVer>
            <activeSwImgVer>acme0815-Rel1-build180664</activeSwImgVer>
            <passiveSwUpgradePkgVer>RAN-201204-006</passiveSwUpgradePkgVer>
            <passiveSwImgVer>acme0815-Rel1-build150362</passiveSwImgVer>
        </getVersionResp>
    </body>
</msg>
```

8.2.2 Software Update Preparation

The Software Update Preparation procedure allows the REC to request the RE to prepare for a new Software Upgrade Package.

The REC shall include in the SOFTWARE UPDATE PREPARATION REQUEST message the following IEs: *FTP Server IP Address*, *FTP Username*, *FTP Password*, *FTP Software Upgrade Package Directory Path*, and *Software Upgrade Package Version Identifier*.

The *FTP Software Upgrade Package Directory Path* IE points to the directory on the given FTP server in which a vendor-specific Software Upgrade Package for the given Software Upgrade Package Version Identifier is stored. One, and only one, Software Upgrade Package shall be stored in the indicated directory at any instance in time. The details of the Software Upgrade Package (file formats, single or multiple files, archive formats) are vendor specific and beyond the scope of the present document.

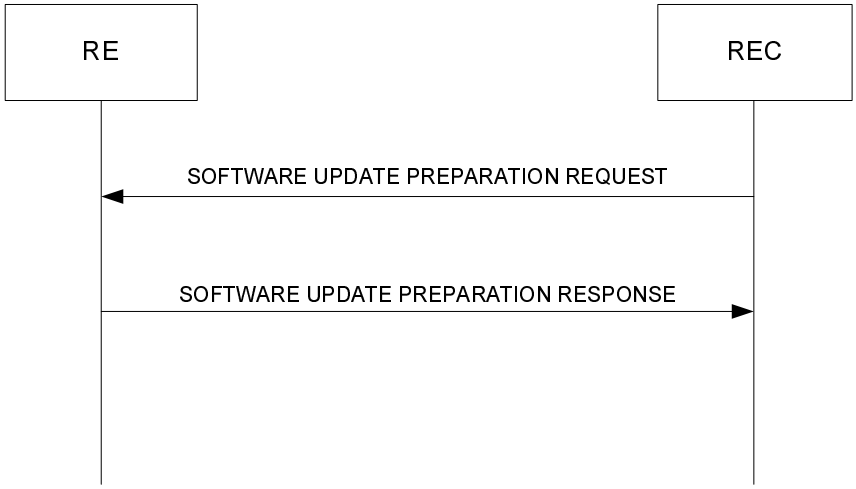

Figure 8.2.2-1: Software Update Preparation procedure

8.2.2.1 Success

When the RE receives the SOFTWARE UPDATE PREPARATION REQUEST message, it shall respond with a SOFTWARE UPDATE PREPARATION RESPONSE message, including the *Result* IE, with the value set to "success". The RE shall then perform the Software Download Procedure.

8.2.2.2 Failure

With the exception of common failure causes, as specified in clause 7.4, this procedure shall never fail.

8.2.2.3 Message parameter details

Message	Encoded name
SOFTWARE UPDATE PREPARATION REQUEST	`updateSwPrepReq`
SOFTWARE UPDATE PREPARATION RESPONSE	`updateSwPrepResp`

Message parameter details for SOFTWARE UPDATE PREPARATION REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
FTP Server IP address	ftpSrvIpAddress	1	String	The IP address shall be in dotted decimal format, e.g. "192.168.0.1"
FTP Username	ftpSrvUserName	1	String	
FTP Password	ftpSrvPassword	1	String	
FTP Software Upgrade Package Directory Path	ftpSrvSwPkgDirPath	1	String	
Software Upgrade package version identifier	SwUpgradePkgVer	1	String	

Message parameter details for SOFTWARE UPDATE PREPARATION RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4

8.2.2.4 Message encoding

XML Schema of message bodies:

```
<!-- Message body: SW Update Preparation Request  -->
 <xsd:complexType name="updateSwPrepReqT">
    <xsd:sequence>
       <xsd:element name="ftpSrvIpAddress" type="xsd:string"/>
       <xsd:element name="ftpSrvUserName" type="xsd:string"/>
       <xsd:element name="ftpSrvPassword" type="xsd:string"/>
       <xsd:element name="ftpSrvSwPkgDirPath" type="xsd:string"/>
       <xsd:element name="SwUpgradePkgVer" type="xsd:string"/>
    </xsd:sequence>
 </xsd:complexType>

 <!-- Message body: SW Update Preparation Response -->
 <xsd:complexType name="updateSwPrepRespT">
    <xsd:sequence>
       <xsd:element name="result" type="ori:resT"/>
    </xsd:sequence>
 </xsd:complexType>
```

Message examples:

```
 <!-- Example: Update SW Preparation Request -->
 <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
     <header>
         <msgType>REQ</msgType>
         <msgUID>4567</msgUID>
     </header>
     <body>
         <updateSwPrepReq>
             <ftpSrvIpAddress>192.168.1.1</ftpSrvIpAddress>
             <ftpSrvUserName>SW_User</ftpSrvUserName>
             <ftpSrvPassword></ftpSrvPassword>
             <ftpSrvSwPkgDirPath>/swImages/acme</ftpSrvSwPkgDirPath>
             <SwUpgradePkgVer>RAN-201204-008</SwUpgradePkgVer>
         </updateSwPrepReq>
     </body>
 </msg>

 <!-- Example: Update SW Preparation Response -->
 <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
     <header>
         <msgType>RESP</msgType>
         <msgUID>4567</msgUID>
     </header>
     <body>
         <updateSwPrepResp>
             <result>SUCCESS</result>
         </updateSwPrepResp>
     </body>
 </msg>
```

8.2.3 Software Download

Following a successful Software Update Preparation procedure, the software download shall be triggered by the RE opening a FTP session using the FTP parameters received during the Software Update Preparation procedure. File transfer is initiated by the RE using the FTP GET (defined in RFC 959 [14] : "RETRIEVE") procedure. Depending on the Software Upgrade Package design the software download may consist of one or several file transfers.

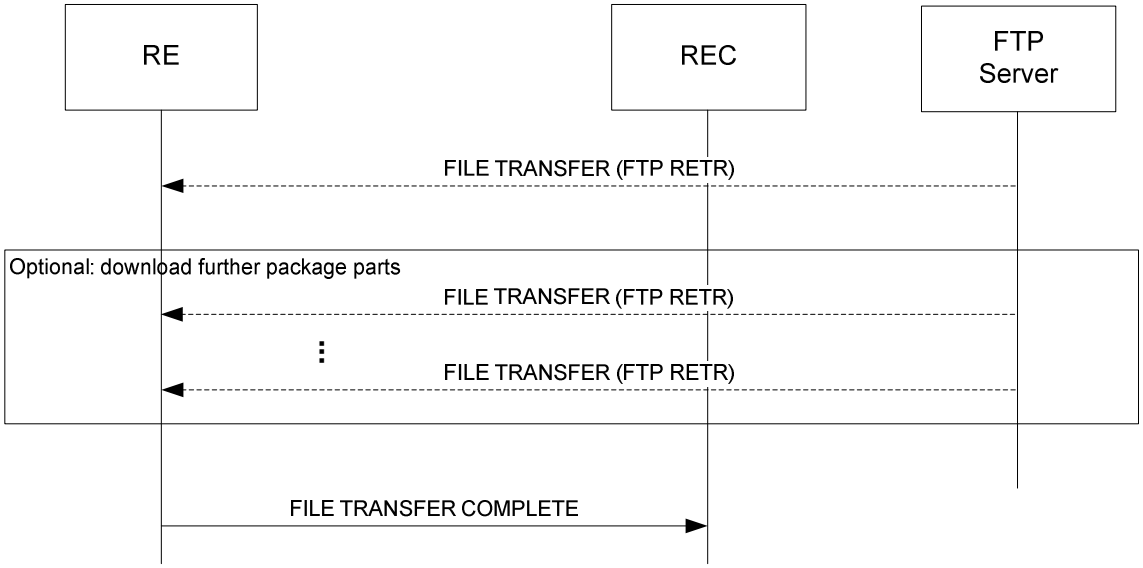

Figure 8.2.3-1: Software Download procedure

8.2.3.1 Success

Following correct reception of the new software via FTP, and once the received software has been stored as the Passive Software Image in non-volatile storage overwriting any existing stored Passive Software image, the RE shall send a FILE TRANSFER COMPLETE message to the REC including the *Result* IE with the value set to "success".

If the Software Upgrade Package contained software compatible with the RE hardware version, but the RE already has the software stored as a software image from a previous software upgrade procedure, the RE is not required to download the software again, and shall send a FILE TRANSFER COMPLETE message to the REC including the *Result* IE with the value set to "success - software already existing".

8.2.3.2 Failure

The RE shall consider the software download as failed, and send a FILE TRANSFER COMPLETE message including the *Result* IE with the value set to one of the failure cause values listed below in the following cases:

- `FAIL_FTP_ERROR` if any file download operation fails.
- `FAIL_BROKEN_IMAGE` if the RE detects that any part of the Software Upgrade Package failed to pass the integrity check. (The integrity check shall verify that the Software Upgrade Package content has not been altered after it was delivered by the RE vendor).
- `FAIL_NO_COMPATIBLE_IMAGE` if the Software Upgrade Package did not contain software compatible with the RE hardware version.
- `FAIL_CANNOT_STORE` if the RE was not able to store the received software image in the non-volatile storage.

If the cause value `FAIL_FTP_ERROR` is reported, the RE shall provide the *Failure Information* IE, containing the "FTP Reply" (as defined in [14]), indicating the reason for the FTP error.

8.2.3.3 Message parameter details

Message	Encoded name
FILE TRANSFER COMPLETE	`transferFileCmpltInd`

Message parameter details for FILE TRANSFER COMPLETE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. + SUCCESS_SOFTWARE_ ALREADY_EXISTING FAIL_FTP_ERROR FAIL_BROKEN_IMAGE FAIL_NO_COMPATIBLE_IMAGE FAIL_CANNOT_STORE
Failure Information	failInfo	0..1	String	

8.2.3.4 Message encoding

XML Schema of message body:

```
<!-- SW download completion specific failure codes -->
<xsd:simpleType name="transferFileCmpltResT">
   <xsd:union memberTypes="ori:resT">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="SUCCESS_SOFTWARE_ALREADY_EXISTING"/>
            <xsd:enumeration value="FAIL_FTP_ERROR"/>
            <xsd:enumeration value="FAIL_BROKEN_IMAGE"/>
            <xsd:enumeration value="FAIL_NO_COMPATIBLE_IMAGE"/>
            <xsd:enumeration value="FAIL_CANNOT_STORE"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>

<!-- Message body: (SW) File Transfer Complete Indication -->
<xsd:complexType name="transferFileCmpltIndT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:transferFileCmpltResT"/>
      <xsd:element name="failInfo" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>
```

Message example:

```
<!-- Example: Transfer File Complete Ind -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>IND</msgType>
        <msgUID>4567</msgUID>
    </header>
    <body>
        <transferFileCmpltInd>
            <result>SUCCESS</result>
        </transferFileCmpltInd>
    </body>
</msg>
```

8.2.4 Software Activation

The Software Activation procedure allows the REC to activate the passive software image in the RE.

8.2.4.1 Success

When the RE receives the SOFTWARE ACTIVATION REQUEST message, it shall set the existing Passive Software Image to be the new Active Software Image and the existing Active Software Image to be the new Passive Software Image. Once this has been achieved, the RE shall respond with a SOFTWARE ACTIVATION RESPONSE message, including the *Result* IE, with the value set to "success". Then the RE shall perform a reset of itself and start-up using the new Active Software Image.

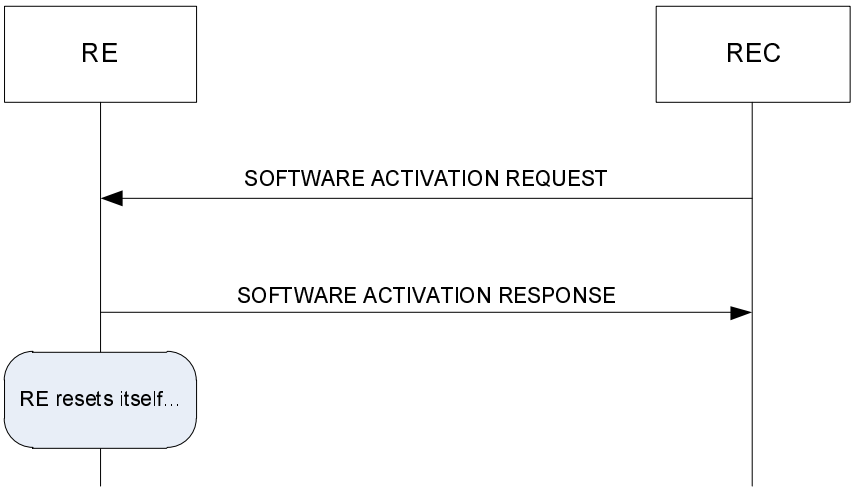

Figure 8.2.4.1-1: Software Activation procedure

8.2.4.2 Failure

The RE shall consider the procedure as failed, and send a SOFTWARE ACTIVATION RESPONSE message including the *Result* IE with the value set to one of the failure cause values listed below in the following cases:

- `FAIL_NOSUCH_IMAGE` if a Passive Software Image does not exist. (This will e.g. be the case for a RE that has not successfully completed the Software Download procedure).

8.2.4.3 Message parameter details

Message	Encoded name
SOFTWARE ACTIVATION REQUEST	`activateSwReq`
SOFTWARE ACTIVATION RESPONSE	`activateSwResp`

Message parameter details for SOFTWARE ACTIVATION RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. + FAIL_NOSUCH_IMAGE

8.2.4.4 Message encoding

XML Schema of message bodies:

```
<!-- Message body: SW Activation Request  -->
<xsd:complexType name="activateSwReqT">
   <!-- no arguments -->
</xsd:complexType>


<!-- SW activation response specific failure codes -->
<xsd:simpleType name="activateSwRespResT">
   <xsd:union memberTypes="ori:resT">
```

```
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FAIL_NOSUCH_IMAGE"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>

<!-- Message body: SW Activation Response -->
<xsd:complexType name="activateSwRespT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:activateSwRespResT"/>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
<!-- Example: Activate Software Request -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>5678</msgUID>
    </header>
    <body>
        <activateSwReq/>
    </body>
</msg>

<!-- Example: Activate Software Response -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>5678</msgUID>
    </header>
    <body>
        <activateSwResp>
            <result>SUCCESS</result>
        </activateSwResp>
    </body>
</msg>
```

8.3 Configuration management

8.3.1 Object Parameter Reporting

The Object Parameter Reporting procedure allows the REC to retrieve the following information:

- the defined object types and instances within the Resource Model of the RE.
- the values of the parameters of the objects.

The OBJECT PARAMETER REPORTING REQUEST shall contain the following IEs: *Object ID*, *Parameter ID*.

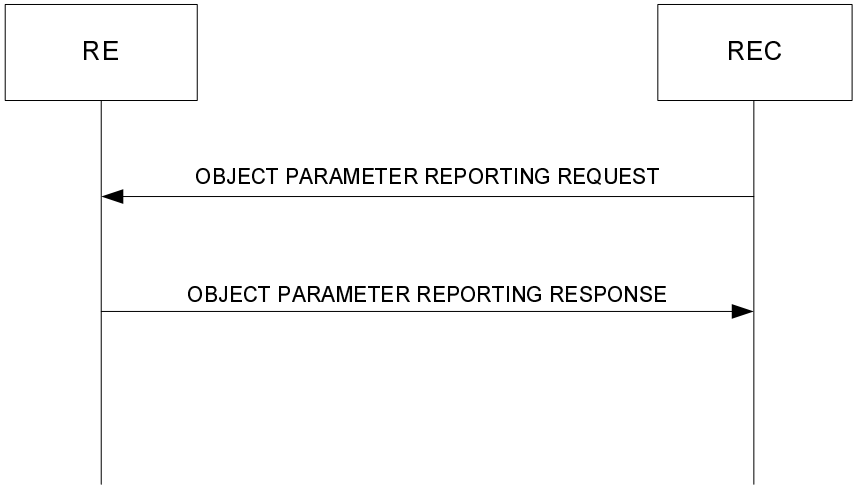

Figure 8.3.1-1: Object Parameter Reporting procedure

8.3.1.1 Success

When the RE receives the OBJECT PARAMETER REPORTING REQUEST message, it shall respond with a OBJECT PARAMETER REPORTING RESPONSE message, containing the following information:

- The *Object ID* IE corresponding to the object for which the report was requested. If the value of the *Object ID* IE in the OBJECT PARAMETER REPORTING REQUEST message contains a wildcard (as defined in clause 6.4), then the RE shall include the *Object ID* IE of every object covered by the wildcard.

  If the RE identifies that the wildcard is valid but does not match any objects, the RE shall not include any *Object ID* IE in the OBJECT PARAMETER REPORTING RESPONSE message.

- The *Parameter ID* IE corresponding to the parameter for which the report was requested. If the value of the *Parameter ID* IE in the OBJECT PARAMETER REPORTING REQUEST message contains a wildcard (as defined in clause 6.6.2.2), then the RE shall include the *Parameter ID* IE of every parameter covered by the wildcard, and which is part of the reported object(s).

  If the RE identifies that the wildcard does not result in any matching parameters, the OBJECT PARAMETER REPORTING RESPONSE message shall not include any *Parameter ID* IE for that object (an "empty" object is reported).

- The *Parameter Value* IE containing the value of each reported parameter.

- The *Result* IE containing the value "success".

8.3.1.2 Failure

The RE shall consider the procedure as failed and send a OBJECT PARAMETER REPORTING RESPONSE message including the *Result* IE with the value set to one of the failure cause values listed below in the following cases:

- FAIL_UNKNOWN_OBJECT if:

  - the value of the *Object ID* IE does not contain a wildcard and does not correspond to any object instance that exists within the RE.

  - the value of the *Object ID* IE contains a wildcard for child objects, but the included "objID" refers to a non-existing parent object.

- FAIL_UNKNOWN_PARAM if the value of the *Parameter ID* IE does not contain a wildcard, and does not correspond to any parameter that is defined within the object or the set of objects given by the *Object ID* IE.

8.3.1.3 Message Parameter Details

Message	Encoded name
OBJECT PARAMETER REPORTING REQUEST	`getParamReq`
OBJECT PARAMETER REPORTING RESPONSE	`getParamResp`

Message parameter details for OBJECT PARAMETER REPORTING REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	objID	1	objRefWcT	
Object parameter	param/name (attribute "name" of element "param")	1	paramRefWcT	

Message parameter details for OBJECT PARAMETER REPORTING RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_UNKNOWN_PARAM"
Object ID	objID	N = 0..*	objRefT	
Object parameter	param/name (attribute "name" of element "param")	M = N * (0..*)	paramRefT	
Parameter value	<none> (value of element "param")	M	String	String contains the parameter value represented as defined by the XML encoding rule of the data type of the parameter (see object parameter definitions in 9.1)

8.3.1.4 Message encoding

XML Schema of message bodies:

```
<!--
      Define base types for parameter values.
      From a messaging point of view, parameter values are treated as strings
      which contain the XML representation of the parameter value.
      The encoding rule is determined by the parameter type definition as
     provided in the resource model object detailed descriptions.
   -->

   <!-- Define type for non-empty parameter values -->
   <xsd:simpleType name="nonemptyValueT">
      <xsd:restriction base="xsd:string">
         <xsd:minLength value="1"/>
      </xsd:restriction>
   </xsd:simpleType>


   <!-- Define non-wildcard "parameter" type (empty value) -->
   <xsd:complexType name="emptyParamT">
      <xsd:attribute name="name" type="ori:paramRefT" use="required"/>
   </xsd:complexType>

   <!-- Define non-wildcard "parameter" type (non-empty value) -->
   <xsd:complexType name="nonemptyParamT">
      <xsd:simpleContent>
         <xsd:extension base="ori:nonemptyValueT">
            <xsd:attribute name="name" type="ori:paramRefT" use="required"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>

   <!-- Define wildcarded "parameter" type, value is empty -->
```

```
<xsd:complexType name="paramWcT">
   <xsd:attribute name="name" type="ori:paramRefWcT" use="required"/>
</xsd:complexType>

<!-- Message Body: Object Parameter Reporting Request -->
<xsd:complexType name="getParamReqT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="param" type="ori:paramWcT"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefWcT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!--  Object Parameter Report specific failure codes -->
<xsd:simpleType name="getParamResT">
   <xsd:union memberTypes="ori:resT">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
            <xsd:enumeration value="FAIL_UNKNOWN_PARAM"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>


<!-- Message Body: Object Parameter Reporting Response -->
<xsd:complexType name="getParamRespT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:getParamResT"/>
      <!-- 0..* objects possible in one message -->
      <xsd:element name="obj" minOccurs="0" maxOccurs="unbounded">
         <xsd:complexType>
            <!-- 0..* parameters possible in one object -->
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
               <xsd:element name="param" type="ori:nonemptyParamT"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
<!-- Example: Get Parameter Request (single object, single parameter) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>6789</msgUID>
    </header>
    <body>
        <getParamReq>
            <obj objID="RE:0">
                <param name="vendorID"/>
            </obj>
        </getParamReq>
    </body>
</msg>

<!-- Example: Get Parameter Response (single object, single parameter) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>6789</msgUID>
    </header>
    <body>
        <getParamResp>
            <result>SUCCESS</result>
            <obj objID="RE:0">
                <param name="vendorID">xyz</param>
```

```
            </obj>
        </getParamResp>
    </body>
</msg>


<!-- Example: Get Parameter Request (single object, all parameters) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>7891</msgUID>
    </header>
    <body>
        <getParamReq>
            <obj objID="RE:0">
                <param name="ALL"/>
            </obj>
        </getParamReq>
    </body>
</msg>

<!-- Example: Get Parameter Response (single object, all parameters -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>7891</msgUID>
    </header>
    <body>
        <getParamResp>
            <result>SUCCESS</result>
            <obj objID="RE:0">
                <param name="vendorID">xyz</param>
                <param name="productID">acme0815</param>
                <param name="productRev">A.01</param>
                <!-- etc. -->
            </obj>
        </getParamResp>
    </body>
</msg>
```

8.3.2 Object Parameter Modification

The Object Parameter Modification procedure allows the REC to configure the values of the parameters of the objects identified by the Resource Model.

The REC initiates the Object Parameter Modification procedure by sending a OBJECT PARAMETER MODIFICATION REQUEST message to the RE:

- The OBJECT PARAMETER MODIFICATION REQUEST message shall contain the following IEs: *Object ID* IE, which contains the Object ID of the object of which one or more parameters is to be modified.
- 1…n of [*Parameter ID* IE, *Parameter Value* IE*]* containing the parameter(s) to be modified, and the new value of the parameter(s), where each parameter and its allowed set of valid values is defined in clause 9.1.

In the present document, neither the *Object ID* IE nor the *Parameter ID* IE shall be set to value "*All*" in this message.

If the object identified by the *Object ID* IE has an administrative state (AST), if any of the parameters to be modified are attributed RW-LOCKED, the precondition shall be: AST for the object is in state "locked".

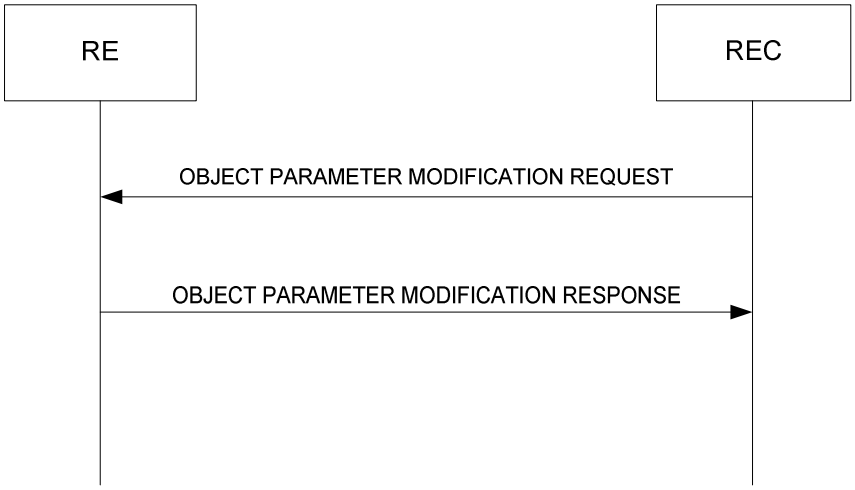

Figure 8.3.2-1: Object Parameter Modification procedure

When the RE receives the OBJECT PARAMETER MODIFICATION REQUEST message, for the object identified by the *Object ID* IE, the RE shall check the following:

- The syntax and range of each parameter individually.
- That the new parameter values of all parameters requested to be modified within the OBJECT PARAMETER MODIFICATION REQUEST message shall represent a configuration of the object that is within the capability of the RE.
- Availability of physical resources which match the requested configuration in conjunction with other configured objects. Available resources are resources that have not been allocated for usage. Resources shall not be considered unavailable just because they are faulty.

8.3.2.1 Success

If the check succeeds, the RE shall modify the existing configured value of each object parameter identified by the *Parameter ID* IE with the values provided in the corresponding *Parameter Value* IE. From that point on, the object shall be considered by the RE to be configured, and underlying physical resources shall be guaranteed by the RE to be available for use.

The RE shall then respond with a OBJECT PARAMETER MODIFICATION RESPONSE message, containing the following information:

- The *Object ID* IE corresponding to the object that was requested to be modified.
- For every parameter given that was requested to be modified in the request:
  - The *Parameter ID* IE corresponding to that parameter.
  - The *Result* IE containing the value "SUCCESS".
- The function's *Global Result* IE is set to "SUCCESS".

8.3.2.2 Failure

If a check fails, the RE shall consider the procedure as failed and maintain the existing (old) parameter configuration of the object. The RE shall send a OBJECT PARAMETER MODIFICATION RESPONSE message to the REC including the list of *Result* IEs as described below.

Two types of Result IEs shall be included:

- a *Result* IE per parameter, which informs about success or failure of the modification of an individual parameter
- a *Global Result* IE which informs about the result of the overall procedure

NOTE: Even if all individual parameters given in the list of parameters have been successfully modified (reflected by the *Result* IE for each parameter set to value "SUCCESS"), the result of the overall procedure may be a failure, as there exist other failure reasons like inconsistencies, or no physical resources to handle the requested combination of parameters.

For each parameter given in the list of parameters to be modified, the value of the *Result* IE shall be set to one of the following values:

- SUCCESS if the parameter is syntactically correct and in range
- FAIL_UNKNOWN_PARAM if the parameter is not part of the set of parameters defined for the given object
- FAIL_PARAM_READONLY if the given parameter is attributed to be read-only
- FAIL_PARAM_LOCKREQUIRED if the given parameter requires the AST of its object to be in "locked" state, but the AST is unlocked at the time of the request
- FAIL_VALUE_OUTOF_RANGE if the given parameter value exceeds its specified limits, or exceeds individually a capability of the RE
- FAIL_VALUE_TYPE_ERROR if the given parameter value does not match the specified type (for example a decimal point or an alphabetic character within an integer value)

The value of the *Global Result* IE shall be set to one of the failure cause values:

- FAIL_UNKNOWN_OBJECT if the value of the *Object ID* IE does not correspond to any object that is defined within the RE
- FAIL_PARAMETER_ERROR if one of the individual parameter checks already resulted in an error (see above)
- FAIL_NOSUCH_RESOURCE if generally no physical resource can be found available which matches the configured parameters

8.3.2.3 Message parameter details

Message	Encoded name
OBJECT PARAMETER MODIFICATION REQUEST	`modifyParamReq`
OBJECT PARAMETER MODIFICATION RESPONSE	`modifyParamResp`

Message parameter details for OBJECT PARAMETER MODIFICATION REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	objID	1	objRefT	
Object parameter	param/name (attribute "name" of element "param")	N = 1..*	paramRefT	
Parameter value	<none> (value of element "param")	N	String	The string contains the parameter value represented as defined by the XML encoding rule of the data type of the parameter (see object parameter definitions in clause 9.1)

Message parameter details for OBJECT PARAMETER MODIFICATION RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Global Result	globResult	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_PARAMETER_ERROR", "FAIL_NOSUCH_RESOURCE"
Object ID	objID	N = 0..1	objRefT	
Object parameter	param/name (attribute "name" of element "param")	M = N * (1..*)	paramRefT	
Result	result	M	String, enumerated	"SUCCESS" "FAIL_UNKNOWN_PARAM" "FAIL_PARAM_READONLY" "FAIL_PARAM_LOCKREQUIRED" "FAIL_VALUE_OUTOF_RANGE" "FAIL_VALUE_TYPE_ERROR"

8.3.2.4 Message encoding

XML Schema of message bodies:

```
<!-- param type definitions from clause 8.3.1 apply -->


 <!-- Message body: Object Parameter Modification Request -->
 <xsd:complexType name="modifyParamReqT">
    <xsd:sequence>
       <xsd:element name="obj">
          <xsd:complexType>
             <xsd:sequence maxOccurs="unbounded">
                <xsd:element name="param" type="ori:nonemptyParamT"/>
             </xsd:sequence>
             <xsd:attribute name="objID" type="ori:objRefT"/>
          </xsd:complexType>
       </xsd:element>
    </xsd:sequence>
 </xsd:complexType>


 <!--  Object Parameter Modification specific overall failure codes -->
 <xsd:simpleType name="modifyParamGlobResT">
    <xsd:union memberTypes="ori:resT">
       <xsd:simpleType>
          <xsd:restriction base="xsd:string">
             <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
             <xsd:enumeration value="FAIL_PARAMETER_ERROR"/>
             <xsd:enumeration value="FAIL_NOSUCH_RESOURCE"/>

          </xsd:restriction>
       </xsd:simpleType>
    </xsd:union>
 </xsd:simpleType>

 <!--  Object Parameter Modification specific per parameter failure codes -->
 <xsd:simpleType name="modifyParamResT">
    <xsd:restriction base="xsd:string">
       <xsd:enumeration value="SUCCESS"/>
       <xsd:enumeration value="FAIL_UNKNOWN_PARAM"/>
       <xsd:enumeration value="FAIL_PARAM_READONLY"/>
       <xsd:enumeration value="FAIL_PARAM_LOCKREQUIRED"/>
       <xsd:enumeration value="FAIL_VALUE_OUTOF_RANGE"/>
       <xsd:enumeration value="FAIL_VALUE_TYPE_ERROR"/>
    </xsd:restriction>
 </xsd:simpleType>


 <!-- Message body: Object Parameter Modification Response -->
 <xsd:complexType name="modifyParamRespT">
    <xsd:sequence>
       <xsd:element name="globResult" type="ori:modifyParamGlobResT"/>
```

```
      <xsd:element name="obj" minOccurs="0">
         <xsd:complexType>
            <xsd:sequence maxOccurs="unbounded">
               <xsd:element name="param" type="ori:emptyParamT"/>
               <xsd:element name="result" type="ori:modifyParamResT"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
<!-- Example: Modify Parameter Request (single parameter) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>7893</msgUID>
    </header>
    <body>
        <modifyParamReq>
            <obj objID="exampleObj:1">
                <param name="parameter 1">value 1</param>
            </obj>
        </modifyParamReq>
    </body>
</msg>

<!-- Example: Modify Parameter Response (single parameter) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>7893</msgUID>
    </header>
    <body>
        <modifyParamResp>
            <globResult>SUCCESS</globResult>
            <obj objID="exampleObj:1">
                <param name="parameter 1"/>
                <result>SUCCESS</result>
            </obj>
        </modifyParamResp>
    </body>
</msg>


<!-- Example: Modify Parameter Request (multiple parameters, success) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>7894</msgUID>
    </header>
    <body>
        <modifyParamReq>
            <obj objID="exampleObj:1">
                <param name="parameter 1">value 1</param>
                <param name="parameter 2">value 2</param>
                <param name="parameter 3">value 3</param>
            </obj>
        </modifyParamReq>
    </body>
</msg>

<!-- Example: Modify Parameter Response (multiple parameters, success) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>7894</msgUID>
    </header>
    <body>
        <modifyParamResp>
            <globResult>SUCCESS</globResult>
            <obj objID="exampleObj:1">
```

```
                <param name="parameter 1"/>
                <result>SUCCESS</result>
                <param name="parameter 2"/>
                <result>SUCCESS</result>
                <param name="parameter 3"/>
                <result>SUCCESS</result>
            </obj>
        </modifyParamResp>
    </body>
</msg>


<!-- Example: Modify Parameter Request (multiple parameters, failure) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>7894</msgUID>
    </header>
    <body>
        <modifyParamReq>
            <obj objID="exampleObj:1">
                <param name="parameter 1">value 1</param>
                <param name="parameter 2">value 2</param>
                <param name="parameter 3">a wrong value</param>
            </obj>
        </modifyParamReq>
    </body>
</msg>

<!-- Example: Modify Parameter Response (multiple parameters, failure) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>7894</msgUID>
    </header>
    <body>
        <modifyParamResp>
            <globResult>FAIL_PARAMETER_ERROR</globResult>
            <obj objID="exampleObj:1">
                <param name="parameter 1"/>
                <result>SUCCESS</result>
                <param name="parameter 2"/>
                <result>SUCCESS</result>
                <param name="parameter 3"/>
                <result>FAIL_VALUE_TYPE_ERROR</result>
            </obj>
        </modifyParamResp>
    </body>
</msg>
```

8.4 Object lifecycle

8.4.1 Object Creation

The Object Creation procedure allows the REC to create and initialize a new instance of the given object type on the RE.

The OBJECT CREATION REQUEST shall contain the following IEs:

- *Object Type* containing the object type to be created. Only object types that are specified as "dynamic" in clause 9.1 may be included in this IE.
- 0…n of [*Parameter ID* IE, *Parameter Value* IE], containing any parameter(s) to be configured, and the value of the parameter(s), where each parameter and its allowed set of valid values is defined in clause 9.1.

When the RE receives the OBJECT CREATION REQUEST message, it shall check the following:

- the syntax and range of each parameter individually;
- that the new parameter values of all parameters requested to be changed within the OBJECT CREATION REQUEST message shall represent a configuration of the object that is within the capability of the RE;

- availability of physical resources which match the requested configuration in conjunction with other configured objects.

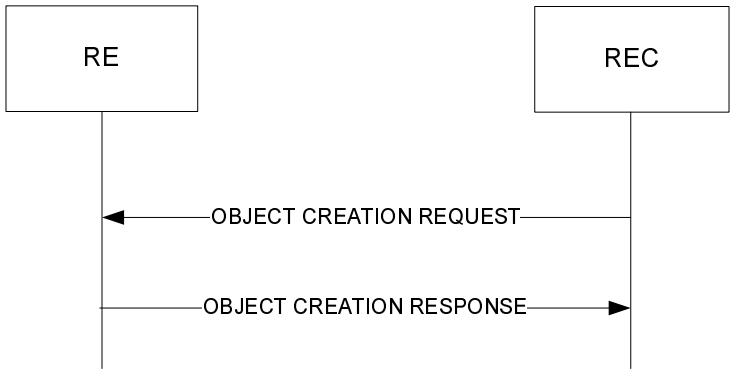

Figure 8.4.1-1: Object creation procedure

8.4.1.1 Success

If the check succeeds, the RE shall:

- Create or allocate an instance of the object type requested to be created. The value of the object instance shall be unique among objects of the same type.
- Where applicable, initialize the parameters of this instance with the parameter values provided in the argument list.
- For parameters of the object type that were not included in the message and for which a default value is specified, set the parameter value to the appropriate default value.
- Where applicable, initialize the object instances' states to appropriate default values.
  Unless specified otherwise, this is:
  - "LOCKED" for AST.
  - "NOT OPERATIONAL", "DISABLED", or "FAILED" for FST.
- Respond with an OBJECT CREATION RESPONSE message, containing the following information:
  - The *Object ID* IE corresponding to the object that was requested to be created.
  - For every parameter given in the request:
    - The *Parameter ID* IE corresponding to the parameter that was requested to be initialized.
    - The *Result* IE containing the value "SUCCESS".
  - The function's *Global Result* IE set to "SUCCESS".

8.4.1.2 Failure

The RE shall consider the procedure as failed and send an OBJECT CREATION RESPONSE message including the list of *Result* IEs as described below. The object instance shall not be created; no *Object ID* shall be provided.

Two types of Result IEs shall be included:

- a *Result* IE per parameter, which informs about success or failure of the initialization of an individual parameter
- a *Global Result* IE which informs about the result of the overall procedure

NOTE: Even if all individual parameters given in the list of parameters have been successfully initialized (reflected by the *Result* IE for each parameter set to value "SUCCESS"), the result of the overall procedure may be a failure, as there exist other failure reasons like inconsistencies, or no physical resources to handle the requested combination of parameters.

For each parameter given in the list of parameters to be initialized, the *Result* IE shall be set to one of the following values:

- SUCCESS if the parameter is syntactically correct and in range
- FAIL_UNKNOWN_PARAM if the parameter is not part of the set of parameters defined for the given object
- FAIL_PARAM_READONLY if the given parameter is attributed to be read-only
- FAIL_PARAM_LOCKREQUIRED if the given parameter requires the AST of its object to be in "locked" state, but AST is unlocked at the time of the request
- FAIL_VALUE_OUTOF_RANGE if the given parameter value exceeds its specified limits, or exceeds individually a capability of the RE
- FAIL_VALUE_TYPE_ERROR if the given parameter value does not match the specified type (for example a decimal sign or a alphabetic character within an integer value)

The value of the function's *Global Result* IE set to one of the failure cause values in the following cases:

- FAIL_UNKNOWN_OBJTYPE if the *Object Type* IE does not correspond to any known object type
- FAIL_STATIC_OBJTYPE if the object type addressed by the *Object Type* IE exists, but is not specified for dynamic creation/deletion
- FAIL_CHILD_NOTALLOWED if the designated parent object in *Object Type* IE is not specified to have contained (child) objects of the given type, or is not specified to have child objects in general.
- FAIL_OUTOF_RESOURCES if no more instances of the object type addressed by the *Object Type* IE can be created.
- FAIL_PARAMETER_ERROR if one of the individual parameter checks already resulted in an error (see above)
- FAIL_NOSUCH_RESOURCE if generally no physical resource is available which matches the configured parameters

8.4.1.3 Message parameter details

Message	Encoded name
OBJECT CREATION REQUEST	`createObjReq`
OBJECT CREATION RESPONSE	`createObjResp`

Message parameter details for OBJECT CREATION REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object type	objType	1	objTypeRefT	
Object parameter	param/name (attribute "name" of element "param")	N = 0..*	paramRefT	
Parameter value	<none> (value of element "param")	N	String	The string contains the parameter value represented as defined by the XML encoding rule of the data type of the parameter (see object parameter definitions in 9.1)

Message parameter details for OBJECT CREATION RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Global Result	globResult	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJTYPE", "FAIL_STATIC_OBJTYPE", "FAIL_CHILD_NOTALLOWED", "FAIL_OUTOF_RESOURCES" "FAIL_PARAMETER_ERROR", "FAIL_NOSUCH_RESOURCE",
Object ID	objID	N = 0..1	objRefT	
Object parameter	param/name (attribute "name" of element "param")	M = N * (0..*)	paramRefT	
Result	result	M	String, enumerated	"SUCCESS" "FAIL_UNKNOWN_PARAM" "FAIL_PARAM_READONLY" "FAIL_PARAM_LOCKREQUIRED" "FAIL_VALUE_OUTOF_RANGE" "FAIL_VALUE_TYPE_ERROR"

8.4.1.4 Message encoding

XML Schema of message bodies:

```
<!-- param type definitions from clause 8.3.1 apply -->

<!-- Message body: Object Creation Request  -->
<xsd:complexType name="createObjReqT">
   <xsd:sequence>
      <xsd:element name="objType">
         <xsd:complexType>
            <xsd:sequence minOccurs = "0" maxOccurs="unbounded">
               <xsd:element name="param" type="ori:nonemptyParamT"/>
            </xsd:sequence>
            <xsd:attribute name="objTypeID" type="ori:objTypeRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>


<!--  Object creation specific overall failure codes -->
<xsd:simpleType name="createObjGlobResT">
   <xsd:union memberTypes="ori:resT">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FAIL_UNKNOWN_OBJTYPE"/>
            <xsd:enumeration value="FAIL_STATIC_OBJTYPE"/>
            <xsd:enumeration value="FAIL_CHILD_NOTALLOWED"/>
            <xsd:enumeration value="FAIL_OUTOF_RESOURCES"/>
            <xsd:enumeration value="FAIL_PARAMETER_ERROR"/>
            <xsd:enumeration value="FAIL_NOSUCH_RESOURCE"/>
```

```
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:union>
   </xsd:simpleType>

   <!--  Object creation specific per parameter failure codes -->
   <xsd:simpleType name="createObjResT">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="SUCCESS"/>
         <xsd:enumeration value="FAIL_UNKNOWN_PARAM"/>
         <xsd:enumeration value="FAIL_PARAM_READONLY"/>
         <xsd:enumeration value="FAIL_PARAM_LOCKREQUIRED"/>
         <xsd:enumeration value="FAIL_VALUE_OUTOF_RANGE"/>
         <xsd:enumeration value="FAIL_VALUE_TYPE_ERROR"/>
      </xsd:restriction>
   </xsd:simpleType>

   <!-- Message body: Object Creation Response  -->
   <xsd:complexType name="createObjRespT">
      <xsd:sequence>
         <xsd:element name="globResult" type="ori:createObjGlobResT"/>
         <xsd:element name="obj" minOccurs="0">
            <xsd:complexType>
               <xsd:sequence minOccurs = "0" maxOccurs="unbounded">
                  <xsd:element name="param" type="ori:emptyParamT"/>
                  <xsd:element name="result" type="ori:createObjResT"/>
               </xsd:sequence>
               <xsd:attribute name="objID" type="ori:objRefT"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
```

Message examples:

```
<!-- Example: Object Creation Request (no parameter, success) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>8934</msgUID>
    </header>
    <body>
        <createObjReq>
            <objType objTypeID="exampleObj"/>
        </createObjReq>
    </body>
</msg>

<!-- Example: Object Creation Response (no parameter, success) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>8934</msgUID>
    </header>
    <body>
        <createObjResp>
            <globResult>SUCCESS</globResult>
            <obj objID="exampleObj:0"/>
        </createObjResp>
    </body>
</msg>


<!-- Example: Object Creation Request (multiple parameters, success) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>8936</msgUID>
    </header>
    <body>
        <createObjReq>
            <objType objTypeID="exampleObj">
                <param name="parameter 1">value 1</param>
                <param name="parameter 2">value 2</param>
                <param name="parameter 3">value 3</param>
            </objType>
```

```
        </createObjReq>
    </body>
</msg>

<!-- Example: Object Creation Response (multiple parameters, success) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>8936</msgUID>
    </header>
    <body>
        <createObjResp>
            <globResult>SUCCESS</globResult>
            <obj objID="exampleObj:1">
                <param name="parameter 1"/>
                <result>SUCCESS</result>
                <param name="parameter 2"/>
                <result>SUCCESS</result>
                <param name="parameter 3"/>
                <result>SUCCESS</result>
            </obj>
        </createObjResp>
    </body>
</msg>


<!-- Example: Object Creation Request (multiple parameters, parameter failure) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>8937</msgUID>
    </header>
    <body>
        <createObjReq>
            <objType objTypeID="exampleObj">
                <param name="parameter 1">value 1</param>
                <param name="parameter 2">value 2</param>
                <param name="parameter 3">a wrong value</param>
            </objType>
        </createObjReq>
    </body>
</msg>

<!-- Example: Object Creation Response (multiple parameters, parameter failure) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>8937</msgUID>
    </header>
    <body>
        <createObjResp>
            <globResult>FAIL_PARAMETER_ERROR</globResult>
            <obj objID="exampleObj:1">
                <param name="parameter 1"/>
                <result>SUCCESS</result>
                <param name="parameter 2"/>
                <result>SUCCESS</result>
                <param name="parameter 3"/>
                <result>FAIL_VALUE_TYPE_ERROR</result>
            </obj>
        </createObjResp>
    </body>
</msg>


<!-- Example: Object Creation Request (multiple parameters, other failure) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>8938</msgUID>
    </header>
    <body>
        <createObjReq>
            <objType objTypeID="aStaticObjType">
                <param name="parameter 1">value 1</param>
                <param name="parameter 2">value 2</param>
                <param name="parameter 3">value 3</param>
            </objType>
```

```
        </createObjReq>
    </body>
</msg>

<!-- Example: Object Creation Response (multiple parameters, other failure) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>8938</msgUID>
    </header>
    <body>
        <createObjResp>
            <globResult>FAIL_STATIC_OBJTYPE</globResult>
        </createObjResp>
    </body>
</msg>
```

8.4.2 Object Deletion

The Object Deletion procedure allows the REC to delete a given object instance and recusively its entire child objects on the RE.

The OBJECT DELETION REQUEST shall contain the following IE: *Object ID* IE identifying the object to be deleted. Only object instances that were previously created using the Object Creation procedure are allowed to be deleted using this procedure.

When the RE receives the OBJECT DELETION REQUEST message, where applicable, it shall check whether the object instance given in the message *Object ID* IE is in AST "LOCKED" state.

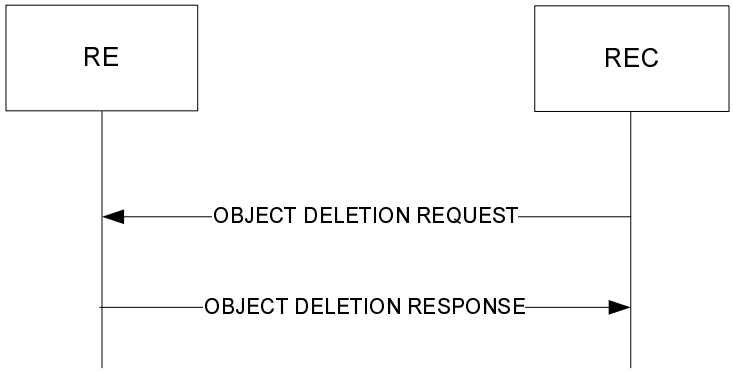

Figure 8.4.2-1: Object deletion procedure

8.4.2.1 Success

If the check is applicable and succeeds, or if the check is not applicable for this object type, then:

- Delete or de-allocate the object instance given in the message IE *Object ID*
- Respond with a OBJECT DELETION RESPONSE message, containing the *Result* IE containing the value "SUCCESS"

8.4.2.2 Failure

The RE shall consider the procedure as failed and send an OBJECT DELETION RESPONSE message including the *Result* IE with the value set to one of the following failure cause values in the following cases:

- FAIL_UNKNOWN_OBJECT if the value of the *Object ID* IE does not correspond to any known object instance
- FAIL_STATIC_OBJTYPE if the object type addressed by the *Object ID* IE exists, but is not specified for dynamic creation/deletion
- FAIL_LOCKREQUIRED if the AST of the object or one of its descendants is not in LOCKED state

8.4.2.3 Message parameter details

Message	Encoded name
OBJECT DELETION REQUEST	deleteObjReq
OBJECT DELETION RESPONSE	deleteObjResp

Message parameter details for OBJECT DELETION REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	objID	N = 0..1	objRefT	

Message parameter details for OBJECT DELETION RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_STATIC_OBJTYPE", "FAIL_LOCKREQUIRED"

8.4.2.4 Message encoding

XML Schema of message bodies:

```
<!-- Message body: Object Deletion Request  -->
   <xsd:complexType name="deleteObjReqT">
      <xsd:sequence>
         <xsd:element name="obj">
            <xsd:complexType>
               <xsd:attribute name="objID" type="ori:objRefT"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>


   <!--  Object deletion specific failure codes -->
   <xsd:simpleType name="deleteObjResT">
      <xsd:union memberTypes="ori:resT">
         <xsd:simpleType>
            <xsd:restriction base="xsd:string">
               <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
               <xsd:enumeration value="FAIL_STATIC_OBJTYPE"/>
               <xsd:enumeration value="FAIL_LOCKREQUIRED"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:union>
   </xsd:simpleType>


   <!-- Message body: Object Deletion Response  -->
   <xsd:complexType name="deleteObjRespT">
      <xsd:sequence>
         <xsd:element name="result" type="ori:deleteObjResT"/>
      </xsd:sequence>
   </xsd:complexType>
```

Message examples:

```
   <!-- Example: Object Deletion Request -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>REQ</msgType>
            <msgUID>9345</msgUID>
        </header>
        <body>
            <deleteObjReq>
```

```
            <obj objID="exampleObj:0"/>
        </deleteObjReq>
    </body>
</msg>

<!-- Example: Object Deletion Response -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>9345</msgUID>
    </header>
    <body>
        <deleteObjResp>
            <result>SUCCESS</result>
        </deleteObjResp>
    </body>
</msg>
```

8.5 Object State management

8.5.1 Object State Reporting

The Object State Reporting procedure allows the REC to acquire the current state of one or more objects of the RE resource model, as well as configure the RE to report when the state changes for any of the indicated objects.

The OBJECT STATE REPORTING REQUEST shall include the following IEs: *Object ID*, *State Type*.

The OBJECT STATE REPORTING REQUEST may include the following IE: *Event-driven reporting*.

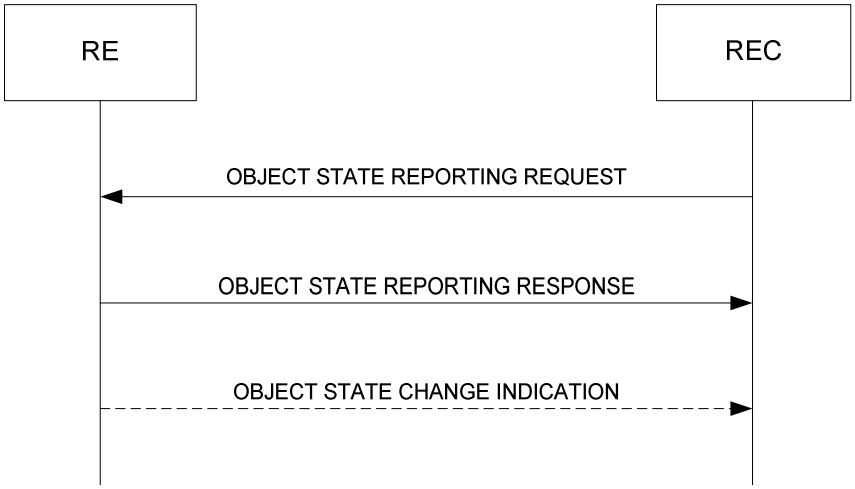

Figure 8.5.1-1: Object State Reporting procedure

8.5.1.1 Success

When the RE receives the OBJECT STATE REPORTING REQUEST message, it shall respond with a OBJECT STATE REPORTING RESPONSE message, containing the following information:

- The *Object ID* IE corresponding to the object for which the report was requested. If the value of the *Object ID* IE in the OBJECT STATE REPORTING REQUEST message contains a wildcard (as defined in clause 6.4), then the RE shall include the *Object ID* IE of every object covered by the wildcard.

  If the RE identifies that the wildcard is valid but does not match any objects, the RE shall not include any *Object ID* IE in the OBJECT STATE REPORTING RESPONSE message.

- The *State Type* IE indicating the type of state for each reported object. If the value of the *State Type* IE in the OBJECT STATE REPORT REQUEST message contains a wildcard (as defined in clause 6.7), then the RE shall include the *State Type* IE of every state covered by the wildcard, and which is part of the set of objects identified by the *Object ID* IE. If the RE identifies that the wildcard does not match any state type defined for the object, the RE shall not include any *State Type* IE in the OBJECT STATE REPORTING RESPONSE message.
- The *Current State* IE indicating the current state for every reported state type of each reported object instance.
- The *Result* IE containing the value "success".

If the OBJECT STATE REPORTING REQUEST includes the *Event-Driven Reporting* IE and the value is set to "*enabled*", the RE shall enable event-driven reporting for the set of objects identified by the *Object ID* IE.

If the OBJECT STATE REPORTING REQUEST includes the *Event-Driven Reporting* IE and the value is set to "*disabled*", the RE shall disable event-driven reporting for the set of objects identified by the *Object ID* IE.

If the OBJECT STATE REPORTING REQUEST does not include the *Event-Driven Reporting* IE, the RE shall make no change to any existing configuration of event-driven reporting for the set of objects identified by the *Object ID* IE.

The default configuration in the RE for event-driven reporting shall be "disabled".

8.5.1.2 Failure

The RE shall consider the procedure as failed and send a OBJECT STATE REPORT RESPONSE message including the *Result* IE with the value set to one of the failure cause values listed below in the following cases:

- FAIL_UNKNOWN_OBJECT if:
  - the value of the *Object ID* IE does not contain a wildcard and does not correspond to any object instance that exists within the RE.
  - the value of the *Object ID* IE contains a wildcard for child objects, but the included "objID" refers to a non-existing parent object.
- FAIL_UNKNOWN_STATETYPE if the state type given in the *State Type* IE does not contain a wildcard, and does not correspond to any state that is defined within the objects given by the *Object ID* IE.
- FAIL_VALUE_OUTOF_RANGE if the argument given in the *Event-Driven Reporting* IE does not match the specified values.

8.5.1.3 Event-triggered reporting of state change

If event-driven reporting has been configured by the REC for a particular state type and one or more objects (indicated by the value of the *State Type* IE and *Object ID* IE in the OBJECT STATE REPORT REQUEST message), upon evaluation of a stage change for the indicated state type of any of the indicated objects, the RE shall send a OBJECT STATE CHANGE INDICATION message to the REC, including the *State Type* IE and *Object ID* IE with the values identifying the object for which the state change of the state type occurred, as well as the *New State* IE indicating the new state of the object.

In the time between the RE receiving a OBJECT STATE REPORTING REQUEST message for one or more object and sending a corresponding OBJECT STATE REPORTING RESPONSE message, the RE shall defer the evaluation of events, and shall not transmit a OBJECT STATE CHANGE INDICATION message, until after the OBJECT STATE REPORTING RESPONSE message has been sent.

8.5.1.4 Message parameters

Message	Encoded name
OBJECT STATE REPORTING REQUEST	`getStateReq`
OBECT STATE REPORTING RESPONSE	`getStateResp`
OBJECT STATE CHANGE INDICATION	`stateChangeInd`

Message parameter details for OBJECT STATE REPORTING REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	objID	1	objRefWcT	
State type	state/type (attribute "type" of element "state")	1	stateTypeWcT	Valid values: "AST", "FST", "ALL"
Event-driven reporting	eventDrivenReporting	0..1	boolean	

Message parameter details for OBJECT STATE REPORTING RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_UNKNOWN_STATETYPE", "FAIL_VALUE_OUTOF_RANGE"
Object ID	objID	N = 0..*	objRefT	
State type	state/type (attribute "type" of element "state")	M = N * (0..*)	stateTypeT	Valid values: "AST", "FST"
State value	state (value of element "state")	M	string	Valid values: For state type = "AST": "LOCKED", "UNLOCKED" For state type = "FST": "PRE_OPERATIONAL", "OPERATIONAL", "DEGRADED", "FAILED", "NOT_OPERATIONAL", "DISABLED"

Message parameter details for OBJECT STATE CHANGE INDICATION message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	objID	1	objRefT	
State name	state/name (attribute "name" of element "state")	1	stateNameT	
State value	state (value of element "state")	1	string	See above.

The value of the message header IE *msgUID* of a OBJECT STATE CHANGE INDICATION message shall be set according to the following rules:

- If the state change was triggered by the REC using a OBJECT STATE MODIFICATION REQUEST, the msgUID shall be set to the value of the msgUID of the triggering OBJECT STATE MODIFICATION REQUEST.

- If the state change was triggered by the RE, the msgUID shall be set to "0".

8.5.1.5 Message encoding

XML Schema of message bodies:

```
<!-- Define non-wildcarded state name type -->
<xsd:simpleType name="stateTypeT">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="AST"/>
      <xsd:enumeration value="FST"/>
   </xsd:restriction>
</xsd:simpleType>

<!-- Define wildcarded state name type -->
<xsd:simpleType name="stateTypeWcT">
   <xsd:union memberTypes="ori:stateTypeT">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ALL"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>

<!-- Define type for non-empty state value -->
<!-- State values listed here for reference purposes.
   Validity checks of the [state name, state value] tupels
   shall be done at while RE is operating -->
<xsd:simpleType name="stateValueT">
   <xsd:restriction base="xsd:string">
      <!-- AST state values -->
      <xsd:enumeration value="LOCKED"/>
      <xsd:enumeration value="UNLOCKED"/>
      <!-- FST state values -->
      <xsd:enumeration value="PRE_OPERATIONAL"/>
      <xsd:enumeration value="OPERATIONAL"/>
      <xsd:enumeration value="DEGRADED"/>
      <xsd:enumeration value="FAILED"/>
      <xsd:enumeration value="NOT_OPERATIONAL"/>
      <xsd:enumeration value="DISABLED"/>
   </xsd:restriction>
</xsd:simpleType>

<!-- Define "non-wildcarded" state type, has non-empty state value -->
<xsd:complexType name="nonemptyStateT">
   <xsd:simpleContent>
      <xsd:extension base="ori:stateValueT">
         <xsd:attribute name="type" type="ori:stateTypeT"/>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>


<!-- Define "wildcarded" state type, has empty value -->
<xsd:complexType name="stateWcT">

   <xsd:attribute name="type" type="ori:stateTypeWcT"/>

</xsd:complexType>


<!-- Message Body: Object State Reporting Request -->
<xsd:complexType name="getStateReqT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="state" type="ori:stateWcT"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefWcT"/>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="eventDrivenReporting" type="xsd:boolean"

         minOccurs="0"/>
   </xsd:sequence>
```

```
</xsd:complexType>

<!-- Object State Reporting specific failure codes -->
<xsd:simpleType name="getStateRespResT">
   <xsd:union memberTypes="ori:resT">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
            <xsd:enumeration value="FAIL_UNKNOWN_STATETYPE"/>
            <xsd:enumeration value="FAIL_VALUE_OUTOF_RANGE"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>

<!-- Message Body: Object State Reporting Response -->
<xsd:complexType name="getStateRespT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:getStateRespResT"/>
      <xsd:element name="obj" minOccurs="0" maxOccurs="unbounded">
         <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
               <xsd:element name="state" type="ori:nonemptyStateT"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!-- Message body: Object State Change Notification -->
<xsd:complexType name="stateChangeIndT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="state" type="ori:nonemptyStateT"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
<!-- Example: Object State Reporting Request (wildcarded) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>34561</msgUID>
    </header>
    <body>
        <getStateReq>
            <obj objID="exampleObj:0">
                <state type="ALL"/>
            </obj>
            <eventDrivenReporting>true</eventDrivenReporting>
        </getStateReq>
    </body>
</msg>

<!-- Example: Object State Reporting Response -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>34561</msgUID>
    </header>
    <body>
        <getStateResp>
            <result>SUCCESS</result>
            <obj objID="exampleObj:0">
                <state type="FST">OPERATIONAL</state>
                <state type="AST">UNLOCKED</state>
            </obj>
        </getStateResp>
```

```
    </body>
</msg>

<!-- Example: Object State Change Indication -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>IND</msgType>
        <msgUID>0</msgUID>
    </header>
    <body>
        <stateChangeInd>
            <obj objID="exampleObj:0">
                <state type="FST">NOT_OPERATIONAL</state>
            </obj>
        </stateChangeInd>
    </body>
</msg>
```

8.5.2 Object State Modification

The Object State Modification procedure allows the REC to trigger a change in the state of a object of the RE Resource Model.

The OBJECT STATE MODIFICATION REQUEST shall include the following IE: *Object ID*, *State Type*, *Requested New State Value*.

The value of the *Object ID* IE shall not contain a wildcard. In the current version of the present document, the *State Type* IE shall be set to "*AST*".

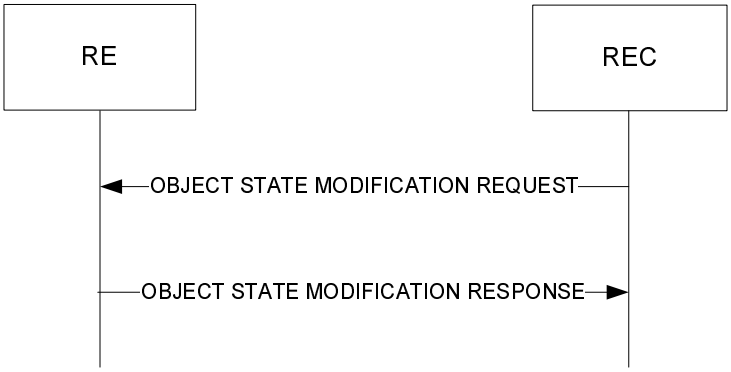

Figure 8.5.2-1: Object State Modification

8.5.2.1 Success

When the RE receives the OBJECT STATE MODIFICATION REQUEST message, the RE shall respond with a OBJECT STATE MODIFICATION RESPONSE message, containing the following information:

- The *Object ID* IE corresponding to the object for which the state modification was requested.
- The *State Type* IE indicating the type of state for which the modification was requested.
- The *Requested New State Value* IE indicating the requested state.
- The *Result* IE containing the value "success".

The RE shall then modify the state of the object to the requested state.

Once the state of the object has been modified, the REC shall acquire information about the new state of the object using the Object State Reporting procedure.

NOTE: If event-triggered state reporting has been configured for the object, the RE triggers an OBJECT STATE CHANGE INDICATION message to be sent to the REC immediately after the state change occurs, without the REC having to request the state information explicitly.

8.5.2.2 Failure

The RE shall consider the procedure as failed and shall send a OBJECT STATE MODIFICATION RESPONSE message including the *Result* IE with the value set to one of the failure cause values listed below in the following cases:

- `FAIL_UNKNOWN_OBJECT` if the value of the *Object ID* IE does not correspond to any object that is defined within the RE.
- `FAIL_UNKNOWN_STATETYPE` if the value of the *State Type* IE does not correspond to any state type that is defined for the object identified by the *Object ID* IE.
- `FAIL_UNKNOWN_STATEVALUE` if the value of the *Requested New State Value* IE does not correspond to any state value that is defined for the state type identified by the *State Type* IE.
- `FAIL_STATE_READONLY` if the state type identified by the *State Type* IE is read-only (not modifiable).
- `FAIL_RESOURCE_UNAVAILABLE` if, in case of unlocking, the RE cannot identify a physical resource which matches the parameters of the object identified by the *Object ID* IE (Parameter conflict).
- `FAIL_RESOURCE_INUSE` if, in case of unlocking, the RE cannot allocate a physical resource which matches the parameters of the object identified by the *Object ID* IE (Resource conflict, resources exhausted).
- `FAIL_PARENT_CHILD_CONFLICT` if the requested state transition stays in contradiction to a parent's or child's state per the rules described in clause 6.7.1.
- `FAIL_PRECONDITION_NOTMET` if the requested state transition cannot be executed because mandatory preconditions are not met.

The RE shall not attempt to modify the state of the object.

8.5.2.3 Abnormal operation

If the OBJECT STATE MODIFICATION RESPONSE indicated that the procedure was successful, but subsequently the state of the object was not reported to have changed, then the REC may choose to re-request the state change.

8.5.2.4 Message parameter details

Message	Encoded name
OBJECT STATE MODIFICATION REQUEST	`modifyStateReq`
OBJECT STATE MODIFICATION RESPONSE	`modifyStateResp`

Message parameter details for OBJECT STATE MODIFICATION REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	objID	1	objRef	
State type	state/type (attribute "type" of element "state")	1	stateTypeT	Valid values: see clause 8.5.1
Requested New State value	state (value of element "state")	1	string	Valid values: see clause 8.5.1

Message parameter details for OBJECT STATE MODIFICATION RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_UNKNOWN_STATETYPE", "FAIL_UNKNOWN_STATEVALUE", "FAIL_STATE_READONLY", "FAIL_RESOURCE_UNAVAILABLE", "FAIL_RESOURCE_INUSE", "FAIL_PARENT_CHILD_CONFLICT", "FAIL_PRECONDITION_NOTMET"
Object ID	objID	1	objRefT	
State type	state/type (attribute "type" of element "state")	1	stateTypeT	Valid values: see clause 8.5.1
Requested New State value	state (value of element "state")	1	string	Valid values: see clause 8.5.1

8.5.2.5 Message encoding

XML Schema of message bodies:

```
<!-- Message body: Object State Modification Request -->
   <xsd:complexType name="modifyStateReqT">
      <xsd:sequence>
         <xsd:element name="obj">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="state" type="ori:stateT"/>
               </xsd:sequence>
               <xsd:attribute name="objID" type="ori:objRefT"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <!-- Object State Modification Response specific failure codes -->
   <xsd:simpleType name="modifyStateRespResT">
      <xsd:union memberTypes="ori:resT">
         <xsd:simpleType>
            <xsd:restriction base="xsd:string">
               <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
               <xsd:enumeration value="FAIL_UNKNOWN_STATETYPE"/>
               <xsd:enumeration value="FAIL_UNKNOWN_STATEVALUE"/>
               <xsd:enumeration value="FAIL_STATE_READONLY"/>
               <xsd:enumeration value="FAIL_RESOURCE_UNAVAILABLE"/>
               <xsd:enumeration value="FAIL_RESOURCE_INUSE"/>
               <xsd:enumeration value="FAIL_PARENT_CHILD_CONFLICT"/>
               <xsd:enumeration value="FAIL_PRECONDITION_NOTMET"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:union>
   </xsd:simpleType>


   <!-- Message body: Object State Modification Response -->
   <xsd:complexType name="modifyStateRespT">
      <xsd:sequence>
         <xsd:element name="result" type="ori:modifyStateRespResT"/>
         <xsd:element name="obj">
            <xsd:complexType>
               <xsd:sequence minOccurs="0">
                  <xsd:element name="state" type="ori:stateT"/>
               </xsd:sequence>
               <xsd:attribute name="objID" type="ori:objRefT"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
```

```
<!-- Message body: Object State Change Notification -->
<xsd:complexType name="stateChangeIndT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="state" type="ori:stateT"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
<!-- Example: Object State Modification Request -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>34570</msgUID>
    </header>
    <body>
        <modifyStateReq>
            <obj objID="exampleObj:0">
                <state type="AST">UNLOCKED</state>
            </obj>
        </modifyStateReq>
    </body>
</msg>

<!-- Example: Object State Modification Response -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>34570</msgUID>
    </header>
    <body>
        <modifyStateResp>
            <result>SUCCESS</result>
            <obj objID="exampleObj:0">
                <state type="AST">UNLOCKED</state>
            </obj>
        </modifyStateResp>
    </body>
</msg>

<!-- Example: Object State Change Indication
     (here: following successful state modification when
        event driven reporting
    is enabled)
      Note that in this case the msgUID is set to the value of the
      msgUID of the triggering OBJECT STATE MODIFICATION REQUEST
-->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>IND</msgType>
        <msgUID>34570</msgUID>
    </header>
    <body>
        <stateChangeInd>
            <obj objID="exampleObj:0">
                <state type="AST">UNLOCKED</state>
            </obj>
        </stateChangeInd>
    </body>
</msg>
```

8.6 Fault management

8.6.1 Fault Reporting

The Fault Reporting procedure allows the REC to acquire information about all current active faults associated with a primary object, as well as configure the RE to report when the fault status changes for any of faults associated with the indicated primary object.

The FAULT REPORTING REQUEST shall include the following IE: *Object ID*.

The FAULT REPORTING REQUEST may include the following IE: *Event-driven reporting*.

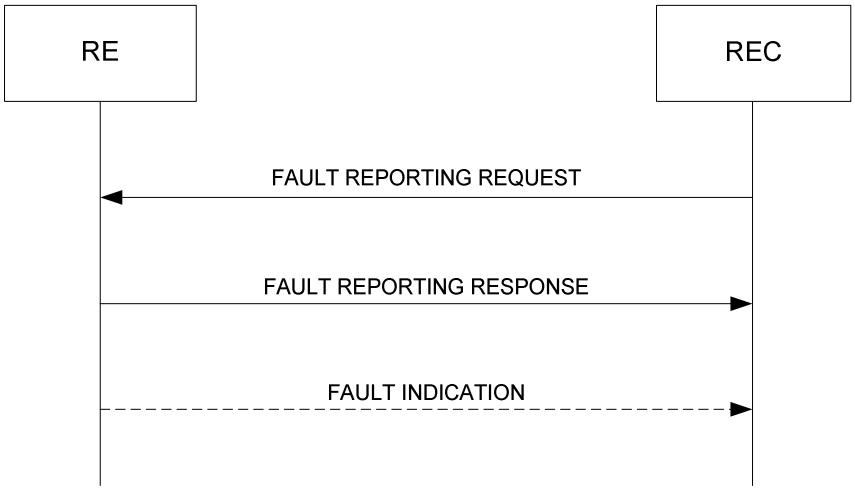

Figure 8.6.1-1: Fault Reporting procedure

8.6.1.1 Success

When the RE receives the FAULT REPORTING REQUEST message, it shall respond with an FAULT REPORTING RESPONSE message, containing the following information for all current active faults:

- The *Object ID* IE corresponding to the primary object for which the reporting of associated faults was requested. If the value of the *Object ID* IE in the FAULT REPORTING REQUEST message contains a wildcard (as defined in clause 6.4), then the RE shall include the *Object ID* IE of every object covered by the wildcard. If the RE identifies that the wildcard is valid but does not match any objects, the RE shall not include any *Object ID* IE in the FAULT REPORTING RESPONSE message.
- The *Fault ID* IE for each active fault associated with the primary object.
- The *Fault Severity* IE for each reported fault.
- When applicable, one or more *Affected Object* IE, corresponding to each additional object affected by the reported fault.
- When applicable, the *Text Description* IE associated to the reported fault.
- The *Result* IE containing the value "success".
- The *Time-Stamp* IE indicating the time of day that the state change was triggered (RE time reference).

If the FAULT REPORTING REQUEST includes the *Event-Driven Reporting* IE, and the value is set to "*enabled*", the RE shall enable event-driven reporting of the faults associated with the set of primary objects identified by the *Object ID* IE.

If the FAULT REPORTING REQUEST includes the *Event-Driven Reporting* IE and the value is set to "*disabled*", the RE shall disable event-driven reporting of the faults associated with the set of primary objects identified by the *Object ID* IE.

If the FAULT REPORTING REQUEST does not include the *Event-Driven Reporting* IE, the RE shall make no change to any existing configuration of event-driven reporting of the faults associated with the set of primary objects identified by the *Object ID* IE.

The default configuration in the RE for event-driven reporting for all objects shall be "disabled".

8.6.1.2 Failure

The RE shall consider the procedure as failed and send a FAULT REPORTING RESPONSE message including the *Result* IE with the value set to one of the failure cause values listed below in the following cases:

- FAIL_UNKNOWN_OBJECT if:
  - the value of the *Object ID* IE does not contain a wildcard, and does not correspond to any object instance that exists within the RE.
  - the value of the *Object ID* IE contains a wildcard for child objects, but the included "objID" refers to a non-existing parent object.
- FAIL_VALUE_OUTOF_RANGE if the argument given in the *Event-Driven Reporting* IE does not match the specified values.

8.6.1.3 Event-triggered reporting

If event-driven reporting has been configured by the REC for one or more objects (indicated by the value of the *Object ID* IE in the FAULT REPORTING REQUEST message), upon evaluation of change of fault state, fault severity, or change of additional affected objects of any of the faults associated with the indicated primary object(s), the RE shall send a FAULT INDICATION message to the REC, including, for each reported fault, the following information:

- *Object ID* IE.
- *Fault ID* IE.
- *Fault State* IE.
- *Fault Severity* IE.
- When applicable, one or more *Affected Object* IE, corresponding to each additional object affected by the reported fault.
- When applicable, the *Text Description* IE associated to the fault.
- The *Time-Stamp* IE indicating the time of day that the state change was triggered (RE time reference).

In the time between the RE receiving a FAULT REPORTING REQUEST message for one or more primary object and sending a corresponding FAULT REPORTING RESPONSE message, the RE shall defer the evaluation of events, and shall not transmit a FAULT INDICATION message, until after the FAULT REPORTING RESPONSE message has been sent.

8.6.1.4 Message parameter details

Message	Encoded name
FAULT REPORTING REQUEST	`getFaultReq`
FAULT REPORTING RESPONSE	`getFaultResp`
FAULT INDICATION	`faultInd`

Message parameter details for FAULT REPORTING REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	objID	1	objRefWcT	
Event-driven reporting	eventDrivenReporting	0..1	boolean	

Message parameter details for FAULT REPORTING RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_VALUE_OUTOF_RANGE"
Object ID	objID	N= 0..*	objRefT	
Fault ID	faultID	M = N * (0..n)	faultIdT	Valid values: see clause 6.8.5
Fault severity	severity	M	faultSeverityT	Valid values: see clause 6.8.2
Time stamp	timestamp	M	dateTime	
Text description	descr	M * (0..1)	string	
Affected object	affectedObj	M * (0..*)	objRefT	

Message parameter details for FAULT INDICATION message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	result	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_VALUE_OUTOF_RANGE"
Object ID	objID	1	objRefT	
Fault ID	faultID	1	faultIdT	Valid values: see clause 6.8.5
Fault state	state	1	faultStateT	Valid values: see clause 6.8.1
Fault severity	severity	1	faultSeverityT	Valid values: see clause 6.8.2
Time stamp	timestamp	1	dateTime	
Text description	descr	0..1	string	
Affected object	affectedObj	0..*	objRefT	

The value of the message header IE *msgUID* of a FAULT INDICATION message shall be set to "0".

8.6.1.5 Message encoding

XML Schema of message bodies:

```
<!-- 6.8.1 Define Fault states -->
<xsd:simpleType name="faultStateT">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="ACTIVE"/>
      <xsd:enumeration value="CLEARED"/>
   </xsd:restriction>
</xsd:simpleType>

<!-- 6.8.2 Define Fault severities -->
<xsd:simpleType name="faultSeverityT">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="FAILED"/>
      <xsd:enumeration value="DEGRADED"/>
      <xsd:enumeration value="WARNING"/>
   </xsd:restriction>
</xsd:simpleType>

<!-- 6.8.5 Define Fault IDs -->
<xsd:simpleType name="faultIdT">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="FAULT_EXT_SUPPLY_UNDERVOLT"/>
      <xsd:enumeration value="FAULT_RE_OVERTEMP"/>
      <xsd:enumeration value="FAULT_DIG_IN_OVERDRIVE"/>
      <xsd:enumeration value="FAULT_RF_OUT_OVERDRIVE"/>
      <xsd:enumeration value="FAULT_TX_GAIN_FAIL"/>
      <xsd:enumeration value="FAULT_RX_GAIN_FAIL"/>
      <xsd:enumeration value="FAULT_VSWR_OUTOF_RANGE"/>
      <xsd:enumeration value="FAULT_NON_AISG_TMA_MALFCT"/>
      <xsd:enumeration value="FAULT_AISG_MALFCT"/>
   </xsd:restriction>
</xsd:simpleType>
```

```
<!-- Define "Fault" data structure -->
<xsd:complexType name="faultT">
   <xsd:sequence>
      <xsd:element name="faultID" type="ori:faultIdT"/>
      <xsd:element name="state" type="ori:faultStateT"/>
      <xsd:element name="severity" type="ori:faultSeverityT"/>
      <xsd:element name="timestamp" type="xsd:dateTime"/>
      <xsd:element name="descr" type="xsd:string" minOccurs="0"/>
      <xsd:element name="affectedObj" type="ori:objRefT" minOccurs="0"

            maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<!-- Define "Fault" data structure (no state included) -->
<xsd:complexType name="faultNoStateT">
   <xsd:sequence>
      <xsd:element name="faultID" type="ori:faultIdT"/>
      <xsd:element name="severity" type="ori:faultSeverityT"/>
      <xsd:element name="timestamp" type="xsd:dateTime"/>
      <xsd:element name="descr" type="xsd:string" minOccurs="0"/>
      <xsd:element name="affectedObj" type="ori:objRefT" minOccurs="0"

            maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>


<!-- Message body: Fault Report Request -->
<xsd:complexType name="getFaultReqT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:attribute name="objID" type="ori:objRefWcT"/>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="eventDrivenReporting" type="xsd:boolean"

            minOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>

<!-- Fault Report specific failure codes -->
<xsd:simpleType name="getFaultRespResT">
   <xsd:union memberTypes="ori:resT">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
            <xsd:enumeration value="FAIL_VALUE_OUTOF_RANGE"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>


<!-- Message body: Fault Report Response -->
<xsd:complexType name="getFaultRespT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:getFaultRespResT"/>
      <xsd:element name="obj" minOccurs="0" maxOccurs="unbounded">
         <xsd:complexType>
            <!-- 0..multiple faults possible in one object -->
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
               <xsd:element name="fault" type="ori:faultNoStateT"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!-- Message body: Fault Indication -->
<xsd:complexType name="faultIndT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
```

```
            <xsd:sequence>
               <xsd:element name="fault" type="ori:faultT"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
<!-- Example: Fault Reporting Request -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>34650</msgUID>
    </header>
    <body>
        <getFaultReq>
            <obj objID="ALL"/>
            <eventDrivenReporting>true</eventDrivenReporting>
        </getFaultReq>
    </body>
</msg>

<!-- Example: Fault Reporting  Response -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>34650</msgUID>
    </header>
    <body>
        <getFaultResp>
            <result>SUCCESS</result>
            <obj objID="RE:0">
                <fault>
                    <faultID>FAULT_OVERTEMP</faultID>
                    <severity>DEGRADED</severity>
                    <timestamp>2012-02-12T16:35:00</timestamp>
                    <descr>PA temp too high; Pout reduced</descr>
                    <affectedObj>TxSigPath_EUTRA:0</affectedObj>
                    <affectedObj>TxSigPath_EUTRA:1</affectedObj>
                </fault>
                <fault>
                    <faultID>FAULT_VSWR_OUTOF_RANGE</faultID>
                    <severity>WARNING</severity>
                    <timestamp>2012-02-12T16:01:05</timestamp>
                </fault>
            </obj>
        </getFaultResp>
    </body>
</msg>

<!-- Example: Fault Reporting Indication -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>IND</msgType>
        <msgUID>0</msgUID>
    </header>
    <body>
        <faultInd>
            <obj objID="RE:0">
                <fault>
                    <faultID>FAULT_TX_GAIN_FAIL</faultID>
                    <state>ACTIVE</state>
                    <severity>FAILED</severity>
                    <timestamp>2012-03-31T09:30:05</timestamp>
                    <descr>PA failure</descr>
                    <affectedObj>TxSigPath_UTRAFDD:0</affectedObj>
                </fault>
            </obj>
        </faultInd>
    </body>
</msg>
```

8.7 Performance management

No performance counters or measurements are defined in the present document.

8.8 Logging

8.8.1 File Available Indication

If event driven reporting is enabled for log instances (parameter "Enable REC Notification" = "TRUE" in corresponding Log object), a FILE AVAILABLE INDICATION is provided to the REC to indicate that a log transfer file is available for upload. The REC should then in turn initiate the File Upload procedure (see clause 8.8.2) to upload the transfer file.

The FILE AVAILABLE INDICATION shall include the following IEs: *Object ID*, which shall contain the "objID" of the Log object for which a transfer file is available.

Figure 8.8.1-1: File Available Indication

8.8.1.1 Message Parameter details

Message	Encoded name
FILE AVAILABLE INDICATION	`fileAvailInd`

Message parameter details for FILE AVAILABLE INDICATION message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	objID	1	objRefT	Valid values: "Log:X"

8.8.1.2 Message encoding

XML Schema of message body:

```
<!-- Message body: File Available Indication -->
<xsd:complexType name="fileAvailableIndT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

Message example:

```
  <!-- Example: File Available Indication -->
 <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
     <header>
         <msgType>IND</msgType>
```

```
        <msgUID>0</msgUID>
    </header>
    <body>
        <fileAvailableInd>
            <obj objID="Log:1"></obj>
        </fileAvailableInd>
    </body>
</msg>
```

8.8.2 File Upload

This procedure is used to upload a log data transfer file to an FTP server. If a transfer file is available it will be uploaded, otherwise this procedure also initiates preparation of the transfer file prior to uploading it.

In addition, it can be used to upload arbitrary files from the RE to a given FTP server. The RE is operated in a FTP client role, so file transfer is provided using the FTP PUT (RFC 959 [14]: "STORE") procedure. The REC has the option to specify a size limit for the uploaded file; the procedure is terminated with an appropriate failure response if the size of the file to be uploaded exceeds that limit.

The REC can optionally specify a destination file name which will be used on the FTP server side instead of the RE local file name. This may be used to override the default log file names specified in ORI.

Log transfer files are identified by the object ID of the associated Log object; other files are identified by providing object ID "RE:0" and the RE local file path.

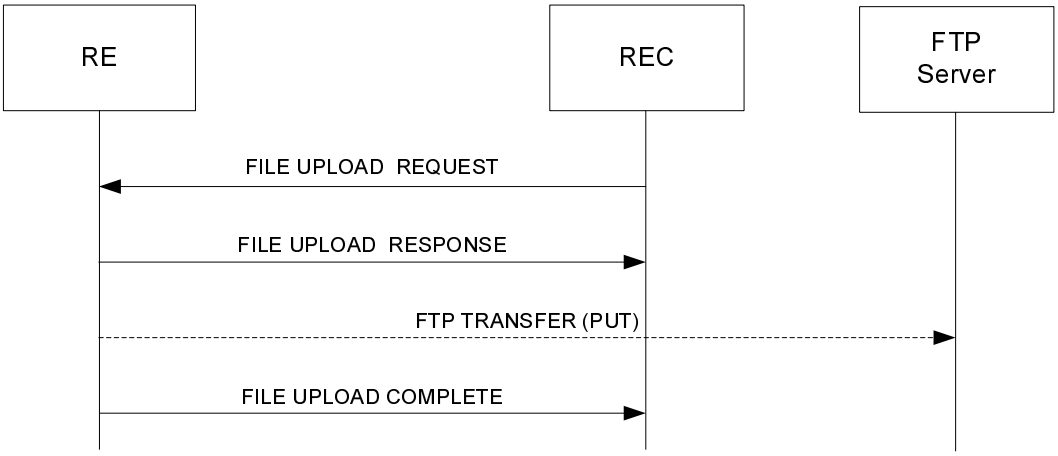

Figure 8.8.2-1: File Upload procedure

The FILE UPLOAD REQUEST message shall include the following IEs: *Object ID, FTP Server IP Address, FTP Username, FTP Server File Path,* and *FTP Password.*

The FILE UPLOAD REQUEST message shall case-dependent include the following IEs: *RE file path.*

The FILE UPLOAD REQUEST message may include the following IEs: *MaxUploadedFileSize.*

- The *Object ID* IE is:
  - in case of log transfer file upload: a valid objRef type identifier referring to an object instance of objType "Log";
  - in case of a generic file upload: a valid objRef type identifier referring to an object instance of objType "RE".

- The *FTP Server File Path* IE shall comprise:

  - Either a path to an existing directory on the FTP server;
    in this case, the file to be uploaded shall be transferred to that directory and its RE file name is preserved;

  - Or a path to a new or existing file on the FTP server;
    in this case, the file to be uploaded will be renamed on the server side to the file name provided in the IE. If a file of the same name already exists on the server, its content is replaced by the content of the file to be uploaded.

    Creation of new directories on the FTP server is not supported.

- The *FTP IP Address*, *FTP Username*, and *FTP Password* IEs, providing address and login information for the FTP server onto which the file is requested to be uploaded.

- The optional *MaxUploadedFileSize* IE indicating the maximum file size in kBytes (multiples of 1 024 bytes) being acceptable to the server. If the size of the file to be uploaded exceeds the value provided in this IE, the procedure shall respond with an appropriate failure code and no transfer attempt shall be started.

- In case a generic file upload is initiated (*Object ID* IE referring to an object instance of objType "RE"), the *RE file path* IE shall be included, containing a RE-specific valid file path up to and including the file name of the file to be uploaded.

When the FILE UPLOAD REQUEST is received, the RE shall determine whether a log file upload or a generic upload is requested. In case of log transfer file upload, the RE shall check:

- if, for the requested Log, a transfer file is available and, if not, check whether data is available to generate a transfer file;

- in case generic file upload is requested, whether the file addressed by the *RE file path* IE is available.

A file being available means the file is physically present and ready for upload by the FTP client.

8.8.2.1 Success

If the checks succeed, the RE shall send a FILE UPLOAD RESPONSE message, including the *Result* IE with the value set to "success" and, if no transfer file is already available for the requested Log, trigger the generation of a transfer file using the available data.

File transfer of the requested file or log transfer file is initiated by the RE using the FTP PUT procedure. The RE shall set the FTP transfer type to "binary/image".

8.8.2.2 Failure

The RE shall consider the procedure as failed and send a FILE UPLOAD RESPONSE message including the *Result* IE with the value set to one of the failure cause values in the following cases (in addition to the common failure causes defined in clause 7.4):

- FAIL_UNKNOWN_OBJECT if the value of the *Object ID* IE is not "RE:0" or does not correspond to any object instance of type "Log" that exists within the RE.

- FAIL_NOSUCH_FILE if at the time the FILE UPLOAD REQUEST is received:

  - for the requested Log, there is neither a transfer file available, nor any data available to generate a transfer file; or

  - if generic file upload is requested, the file addressed by the *RE file path* IE is not available.

- FAIL_SIZE_LIMIT if the size of the file designated for upload is greater than the size limit indicated by the *MaxUploadedFileSize* IE.

File Transfer shall not be initiated by the RE in this case.

8.8.2.3 File Upload Completion reporting

When the FTP upload procedure is completed, the RE shall send a FILE UPLOAD COMPLETE INDICATION to the REC to inform the REC about the completion and the result of the procedure.

The FILE UPLOAD COMPLETE INDICATION shall contain the following IEs: *Result.*

The FILE UPLOAD COMPLETE INDICATION shall case-dependant contain the following IEs: *FailureInfo*.

The *Result* IE shall be set to:

- SUCCESS if the transfer was successfully completed.
- FAIL_FTP_ERROR if the RE has detected any error during a file transfer.

The *FailureInfo* IE shall only be present in case of failure (FTP 4xx/5xx result codes), and shall contain the "FTP failure response message" (as defined in [14]) indicating the failure reason.

8.8.2.4 Message parameter details

Message	Encoded name
FILE UPLOAD REQUEST	`uploadFileReq`
FILE UPLOAD RESPONSE	`uploadFileResp`
FILE UPLOAD COMPLETE INDICATION	`uploadFileCmplInd`

Message parameter details for FILE UPLOAD REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	`objID`	1	String, enumerated or objRefT	Valid values: "RE:0" or "Log:X"
FTP Server IP Address	`ftpSrvIpAddress`	1	String	The IP address shall be in dotted decimal format, e.g. "192.168.0.1"
FTP Username	`ftpSrvUserName`	1	String	
FTP Password	`ftpSrvPassword`	1	String	
FTP Server File Path	`ftpSrvFilePath`	1	String	
RE File Path	`ReFilePath`	0..1	String	
Max Uploaded File Size	`maxUploadFileSize`	0..1	Unsigned Short	The size shall be in kBytes.

Message parameter details for FILE UPLOAD RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	`result`	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_NOSUCH_FILE", "FAIL_SIZE_LIMIT"
Object ID	`objID`	0..1	String, enumerated or objRefT	Valid values: "RE:0" or "Log:X"

Message parameter details for FILE UPLOAD COMPLETE INDICATION message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	`result`	1	String, enumerated	"SUCCESS", "FAIL_FTP_ERROR"
Object ID	`objID`	1	String, enumerated or objRefT	Valid values: "RE:0" or "Log:X"
Failure Info	`failInfo`	0..1	String	

8.8.2.5 Message encoding

XML Schema of message bodies:

```
<!-- Message body: File Upload Request -->
<xsd:complexType name="uploadFileReqT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="ftpSrvIpAddress" type="xsd:string"/>
               <xsd:element name="ftpSrvUserName" type="xsd:string"/>
               <xsd:element name="ftpSrvPassword" type="xsd:string"/>
               <xsd:element name="ftpSrvFilePath" type="xsd:string"/>
               <xsd:element name="ReFilePath" type="xsd:string"
                     minOccurs="0"/>
               <xsd:element name="maxUploadFileSize" type="xsd:unsignedShort"
                     minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!-- File Upload specific failure codes -->
<xsd:simpleType name="uploadFileRespResT">
   <xsd:union memberTypes="ori:resT">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
            <xsd:enumeration value="FAIL_NOSUCH_FILE"/>
            <xsd:enumeration value="FAIL_SIZE_LIMIT"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>


<!-- Message body: File Upload Response -->
<xsd:complexType name="uploadFileRespT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:uploadFileRespResT"/>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!-- File Upload Complete specific failure codes -->
<xsd:simpleType name="uploadFileCmplIndResT">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="SUCCESS"/>
      <xsd:enumeration value="FAIL_FTP_ERROR"/>
   </xsd:restriction>
</xsd:simpleType>

<!-- Message body: File Upload Complete Indication -->
<xsd:complexType name="uploadFileCmplIndT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:uploadFileCmplIndResT"/>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="failInfo" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
    <!-- Example: File Upload Request (log file) -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>REQ</msgType>
            <msgUID>39710</msgUID>
        </header>
        <body>
            <uploadFileReq>
                <obj objID="Log:0">
                    <ftpSrvIpAddress>192.168.10.10</ftpSrvIpAddress>
                    <ftpSrvUserName>RE</ftpSrvUserName>
                    <ftpSrvPassword>TopSecret</ftpSrvPassword>
                    <ftpSrvFilePath>/ftp/upload/logs/RE_0</ftpSrvFilePath>
                    <maxUploadFileSize>10000</maxUploadFileSize>
                </obj>
            </uploadFileReq>
        </body>
    </msg>

    <!-- Example: File Upload Response (log file) -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>RESP</msgType>
            <msgUID>39710</msgUID>
        </header>
        <body>
            <uploadFileResp>
                <result>SUCCESS</result>
                <obj objID="Log:0"/>
            </uploadFileResp>
        </body>
    </msg>

    <!-- Example: File Upload Complete Indication (log file) -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>IND</msgType>
            <msgUID>39710</msgUID>
        </header>
        <body>
            <uploadFileCmplInd>
                <result>SUCCESS</result>
                <obj objID="Log:0"></obj>
            </uploadFileCmplInd>
        </body>
    </msg>


    <!-- Example: File Upload Request (some file) -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>REQ</msgType>
            <msgUID>39715</msgUID>
        </header>
        <body>
            <uploadFileReq>
                <obj objID="RE:0">
                    <ftpSrvIpAddress>192.168.10.10</ftpSrvIpAddress>
                    <ftpSrvUserName>UnknownUser</ftpSrvUserName>
                    <ftpSrvPassword>Secret</ftpSrvPassword>
<ftpSrvFilePath>/ftp/upload/vendor_data/RE_0</ftpSrvFilePath>
                    <ReFilePath>/var/log/application/xyzdata.txt</ReFilePath>
                    <maxUploadFileSize>10000</maxUploadFileSize>
                </obj>
            </uploadFileReq>
        </body>
    </msg>

    <!-- Example: File upload Response (some file) -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>RESP</msgType>
            <msgUID>39715</msgUID>
        </header>
        <body>
            <uploadFileResp>
```

```
            <result>SUCCESS</result>
            <obj objID="RE:0"></obj>
        </uploadFileResp>
    </body>
</msg>

<!-- Example: File Upload Complete Indication (some file, failure) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>IND</msgType>
        <msgUID>39715</msgUID>
    </header>
    <body>
        <uploadFileCmplInd>
            <result>FAIL_FTP_ERROR</result>
            <obj objID="RE:0"/>
            <failInfo>530 Not logged in</failInfo>
        </uploadFileCmplInd>
    </body>
</msg>
```

8.8.3 Activate Configuration

The Activate Configuration procedure is used to activate a downloaded or RE local file as a configuration input to log instances. If file download is required, the configuration file is downloaded from a specified FTP server location to the RE using the FTP GET (RFC 959 [14]: "RETRIEVE") procedure. Alternatively, a file already existing locally on the RE can be specified. If a file is available, the RE may activate it as configuration input to the given log instance. The RE may delete the downloaded file after activation.

The details of the file format and activation procedure are implementation specific and beyond the scope of the present document.

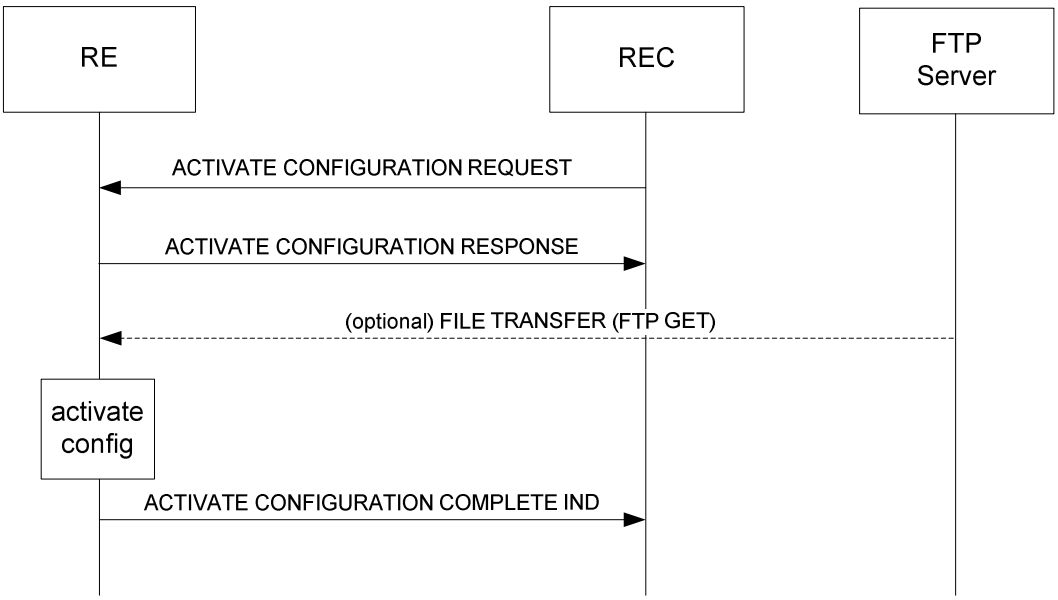

Figure 8.8.3-1: Activate Configuration procedure

The ACTIVATE CONFIGURATION REQUEST message shall include the following IEs: *Object ID, File Location,* and *File Path.*

The ACTIVATE CONFIGURATION REQUEST message shall conditionally include the following IEs: *FTP Server IP Address*, *FTP Username,* and *FTP Password.*

The IEs are defined as follows:

- The *Object ID* IE shall contain a valid objRef type identifier referring to an object instance of objType "Log".

- The *FileLocation* IE shall take one of the following values:
  - "FTP" if the configuration file shall be downloaded from an FTP server.
  - "RE" if the configuration file is expected to be present locally on the RE file system.
- Depending on the value of the *FileLocation* IE, the *File Path* IE shall either contain a path to an existing file on the FTP server, or contain a path to an existing file on the RE file system. A path to a file includes the directory path as well as the file name.
- The *FTP IP Address*, *FTP Username*, and *FTP Password* IEs, shall only be available if the *FileLocation* IE is set to "FTP". They shall provide address and login information for the FTP server from which the configuration file is requested to be downloaded.

When the ACTIVATE CONFIGURATION REQUEST is received, the RE shall check whether a potentially requested RE local file is present. Further checks and activities are as described below.

8.8.3.1 Success

When the RE receives the ACTIVATE CONFIGURATION REQUEST message, the RE shall send an ACTIVATE CONFIGURATION RESPONSE message, including the *Result* IE with the value set to "success".

The RE shall execute the following steps:

- If the *FileLocation* IE take the value "FTP":
  - A transfer of the requested file is initiated by the RE using the FTP GET (RFC 959 [14]: "RETRIEVE") procedure. The RE shall set the FTP transfer type to "binary/image".
  - In case of an FTP transfer failure, the RE shall terminate the procedure and send an ACTIVATE CONFIGURATION COMPLETE INDICATION with appropriate failure code to the REC.
- The downloaded or local file shall be activated as configuration file for the log identified by the *Object ID* IE.
- The RE shall send an ACTIVATE CONFIGURATION COMPLETE INDICATION with appropriate result code to the REC (success, or failure code).
- The RE may remove a downloaded file after activation.

8.8.3.2 Failure

The RE shall consider the procedure as failed and send an ACTIVATE CONFIGURATION RESPONSE message including the *Result* IE with the value set to one of the failure cause values in the following cases (in addition to the common failure causes defined in clause 7.4):

- FAIL_UNKNOWN_OBJECT if the value of the *Object ID* IE does not correspond to any object instance of type "Log" that exists within the RE.
- FAIL_NOSUCH_FILE if the *FileLocation* IE = "RE" and the file referenced by the *FilePath* IE is not available (does not exist or is not accessible).

Activation shall not be initiated by the RE in this case.

8.8.3.3 Activate Configuration completion reporting

When the activation procedure is completed, the RE shall send an ACTIVATE CONFIGURATION COMPLETE INDICATION to the REC to inform the REC about the completion and the result of the procedure.

The ACTIVATE CONFIGURATION COMPLETE INDICATION shall contain the following IEs: *Result.*

The ACTIVATE CONFIGURATION COMPLETE INDICATION shall case-dependant contain the following IEs: *FailureInfo.*

The *Result* IE shall be set to one of the following values:

- SUCCESS if the activation was successfully completed.
- FAIL_ACTIVATION_ERROR if the attempt to activate the file ended in a error.
- FAIL_FTP_ERROR if the RE has detected any error during a file transfer.

The *FailureInfo* IE shall only be present in case of failure and containing the "FTP response message" (as defined in [14]) or short failure information from the activation procedure indicating the failure reason.

8.8.3.4 Message Parameter details

Message	Encoded name
Activate Configuration Request	`activateConfigReq`
Activate Configuration Response	`activateConfigResp`
Activate Configuration Complete Indication	`activateConfigCmplInd`

Message parameter details for ACTIVATE CONFIGURATION REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	`objID`	1	objRefT	Valid values: "Log:X"
File Location	`fileLocation`	1	String, enumerated	Valid values: "FTP", "RE"
File Path	`FilePath`	1	String	
FTP Server IP Address	`ftpSrvIpAddress`	0..1	String	
FTP Username	`ftpSrvUserName`	0..1	String	
FTP Password	`ftpSrvPassword`	0..1	String	

Message parameter details for ACTIVATE CONFIGURATION RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	`result`	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_NOSUCH_FILE"
Object ID	`objID`	0..1	objRefT	Valid values: "Log:X"

Message parameter details for ACTIVATE CONFIGURATION COMPLETE INDICATION message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	`result`	1	String, enumerated	"SUCCESS", "FAIL_FTP_ERROR", "FAIL_ACTIVATION_ERROR"
Object ID	`objID`	1	objRefT	Valid values: "Log:X"
Failure Info	`failInfo`	0..1	String	

8.8.3.5 Message encoding

XML Schema of message bodies:

```
<!-- File Location IE type -->
<xsd:simpleType name="fileLocationT">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="FTP"/>
      <xsd:enumeration value="RE"/>
   </xsd:restriction>
</xsd:simpleType>
```

```
<!-- Message body: Activate Configuration Request -->
<xsd:complexType name="activateConfigReqT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="fileLocation" type="ori:fileLocationT"/>
               <xsd:element name="FilePath" type="xsd:string"/>
               <xsd:element name="ftpSrvIpAddress" type="xsd:string"

                  minOccurs="0"/>
               <xsd:element name="ftpSrvUserName" type="xsd:string"

                  minOccurs="0"/>
               <xsd:element name="ftpSrvPassword" type="xsd:string"

                  minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!-- Activate Configuration specific failure codes -->
<xsd:simpleType name="activateConfigRespResT">
   <xsd:union memberTypes="ori:resT">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
            <xsd:enumeration value="FAIL_NOSUCH_FILE"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>


<!-- Message body: Activate Configuration Response -->
<xsd:complexType name="activateConfigRespT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:activateConfigRespResT"/>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!-- Activate Configuration Complete specific failure codes -->
<xsd:simpleType name="activateConfigCmpltIndResT">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="SUCCESS"/>
      <xsd:enumeration value="FAIL_ACTIVATION_ERROR"/>
      <xsd:enumeration value="FAIL_FTP_ERROR"/>
   </xsd:restriction>
</xsd:simpleType>

<!-- Message body: Activate Configuration Complete Indication -->
<xsd:complexType name="activateConfigCmplIndT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:uploadFileCmplIndResT"/>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="failInfo" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
<!-- Example: Activate Configuration Request (file from FTP server) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>39780</msgUID>
    </header>
    <body>
        <activateConfigReq>
            <obj objID="Log:8">
                <fileLocation>FTP</fileLocation>
            <FilePath>/ftp/download/config/RE_0/config_debug.zip</FilePath>
                <ftpSrvIpAddress>192.168.10.10</ftpSrvIpAddress>
                <ftpSrvUserName>RE</ftpSrvUserName>
                <ftpSrvPassword>TopSecret</ftpSrvPassword>
            </obj>
        </activateConfigReq>
    </body>
</msg>

<!-- Example: Activate Configuration Response (file from FTP server) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>39780</msgUID>
    </header>
    <body>
        <activateConfigResp>
            <result>SUCCESS</result>
            <obj objID="Log:8"/>
        </activateConfigResp>
    </body>
</msg>

<!-- Example: Activate Configuration Complete Indication (file from FTP server) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>IND</msgType>
        <msgUID>39780</msgUID>
    </header>
    <body>
        <activateConfigCmplInd>
            <result>SUCCESS</result>
            <obj objID="Log:8"/>
        </activateConfigCmplInd>
    </body>
</msg>



<!-- Example: Activate Configuration Request (file on RE) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>39785</msgUID>
    </header>
    <body>
        <activateConfigReq>
            <obj objID="Log:10">
                <fileLocation>RE</fileLocation>
                <FilePath>/var/log/configs/config_debug.zip</FilePath>
            </obj>
        </activateConfigReq>
    </body>
</msg>

<!-- Example: Activate Configuration Response (file on RE) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>39785</msgUID>
    </header>
    <body>
        <activateConfigResp>
            <result>SUCCESS</result>
            <obj objID="Log:10"/>
        </activateConfigResp>
```

```
    </body>
</msg>

<!-- Example: Activate Configuration Complete Indication (file on RE) -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>IND</msgType>
        <msgUID>39785</msgUID>
    </header>
    <body>
        <activateConfigCmplInd>
            <result>SUCCESS</result>
            <obj objID="Log:10"/>
        </activateConfigCmplInd>
    </body>
</msg>
```

8.9 AISG specific procedures

8.9.1 Device scan

The AISG Device Scan procedure allows the REC to initiate an AISG Device Scan (see [5]) from the RE and, on completion of the scan, acquire from the RE the list of ALDs that have been discovered.

The REC initiates the procedure by sending an AISG DEVICE SCAN REQUEST to the RE. The AISG DEVICE SCAN REQUEST message shall contain the following IEs: *Object ID* - which shall be the "AISG Port" object instance in the RE for which the AISG Device Scan is required.

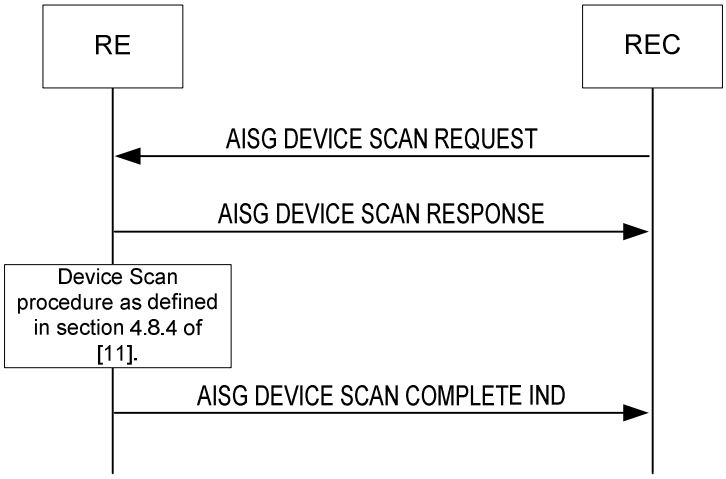

Figure 8.9.1-1: AISG Device Scan procedure

8.9.1.1 Success

When the RE receives an AISG DEVICE SCAN REQUEST message, it shall

- respond with a AISG DEVICE SCAN RESPONSE message, containing the following IEs:
  - The *Object ID* IE which corresponds to the Object ID IE provided in the request message.
  - The *Result* IE set to the value "success".
- Launch the Device Scan procedure from the required AISG Port as defined in clause 4.8.4 of [5].
  - Precondition: The AST of the AISGPort object shall be in state LOCKED.

As a result of the Device Scan procedure, the RE shall:

- assign an HDLC Address to all identified ALDs;

- for each newly identified ALD, create an associated ALD child object of the corresponding AISG port object;
- delete existing ALD objects for ALDs that are not anymore detected on the bus.

The RE may by default delete all AISGALD child objects and create new AISGALD child objects for all previously-identified and newly identified ALDs as result of the Device Scan. Therefore the REC shall not assume that the mapping defined between physical ALDs and their associated AISGALD objects before a Device Scan procedure is maintained after a successful Device Scan procedure.

8.9.1.2 Failure

The RE shall consider the request as failed, not launch the AISG device can, and send the AISG DEVICE SCAN RESPONSE message with the value set to one of the failure values listed below in the following cases:

- FAIL_UNKNOWN_OBJTYPE if the objTypeID part of the *Object ID* IE (objRefT) does not correspond to "AISGPort".
- FAIL_UNKNOWN_OBJECT if the object instance addressed by the *Object ID* IE does not exist.
- FAIL_LOCKREQUIRED if the AST of the AISG port is UNLOCKED at the time of the request.
- FAIL_PRECONDITION_NOTMET if the AISG port's bus power is disabled at the time of the request. Note that disabling the bus power after the request has been accepted leads to unspecified results, and is not recommended.
- FAIL_RE_BUSY if there is already an AISG DEVICE SCAN procedure going on for the same AISG port at the time the request is received.

8.9.1.3 Device Scan completion

The AISG Device Scan procedure always completes once it is launched. When the AISG Device Scan procedure is completed, the RE shall send an AISG DEVICE SCAN COMPLETE INDICATION message to the REC, including the *Object ID* IE corresponding to the AISGPort object that launched the scan, and the *Number of ALD* IE that indicating the number of identified ALDs associated with that AISGPort object. This number may include the value 0 if no ALDs have been found.

8.9.1.4 Message parameter details

Message	Encoded name
AISG DEVICE SCAN REQUEST	`scanDeviceReq`
AISG DEVICE SCAN RESPONSE	`scanDeviceResp`
AISG DEVICE SCAN COMPLETE INDICATION	`scanDeviceComplInd`

Message parameter details for AISG DEVICE SCAN REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	`objID`	1	objRefT	objID = "aisgPort:X"

Message parameter details for AISG DEVICE SCAN RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	`result`	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJTYPE", "FAIL_UNKNOWN_OBJECT", "FAIL_LOCKREQUIRED", "FAIL_PRECONDITION_NOTMET", "FAIL_RE_BUSY"
Object ID	`objID`	1	objRefT	

Message parameter details for AISG DEVICE SCAN COMPLETE INDICATION message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	objID	1	objRefT	
Number of ALDs found	numALDs	1	Unsigned byte	

8.9.1.5 Message encoding

XML Schema of message bodies:

```
<!-- Message body: Device Scan Request -->
 <xsd:complexType name="scanDeviceReqT">
    <xsd:sequence>
       <xsd:element name="obj">
          <xsd:complexType>
             <xsd:attribute name="objID" type="ori:objRefT"/>
          </xsd:complexType>
       </xsd:element>
    </xsd:sequence>
 </xsd:complexType>

 <!-- Device Scan specific failure codes -->
 <xsd:simpleType name="scanDeviceRespResT">
    <xsd:union memberTypes="ori:resT">
       <xsd:simpleType>
          <xsd:restriction base="xsd:string">
             <xsd:enumeration value="FAIL_UNKNOWN_OBJTYPE"/>
             <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
             <xsd:enumeration value="FAIL_LOCKREQUIRED"/>
             <xsd:enumeration value="FAIL_PRECONDITION_NOTMET"/>
             <xsd:enumeration value="FAIL_RE_BUSY"/>
          </xsd:restriction>
       </xsd:simpleType>
    </xsd:union>
 </xsd:simpleType>


 <!-- Message body: Device Scan Response -->
 <xsd:complexType name="scanDeviceRespT">
    <xsd:sequence>
       <xsd:element name="obj">
          <xsd:complexType>
             <xsd:attribute name="objID" type="ori:objRefT"/>
          </xsd:complexType>
       </xsd:element>
       <xsd:element name="result" type="ori:scanDeviceRespResT"/>
    </xsd:sequence>
 </xsd:complexType>


 <!-- Message body: Device Scan Complete Indication -->
 <xsd:complexType name="scanDeviceComplIndT">
    <xsd:sequence>
       <xsd:element name="obj">
          <xsd:complexType>
             <xsd:sequence>
                <xsd:element name="numALDs" type="xsd:unsignedByte"/>
             </xsd:sequence>
             <xsd:attribute name="objID" type="ori:objRefT"/>
          </xsd:complexType>
       </xsd:element>
    </xsd:sequence>
 </xsd:complexType>
```

Message examples:

```
<!-- Example: Device Scan Request -->
    <msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
        <header>
            <msgType>REQ</msgType>
            <msgUID>25250</msgUID>
        </header>
```

```
    <body>
        <scanDeviceReq>
            <obj objID="aisgPort:0"/>

        </scanDeviceReq>
    </body>
</msg>

<!-- Example: Device Scan Response -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>25250</msgUID>
    </header>
    <body>
        <scanDeviceResp>
            <result>SUCCESS</result>
            <obj objID="aisgPort:0"/>
        </scanDeviceResp>
    </body>
</msg>

<!-- Example:  Device Scan Complete Indication -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>IND</msgType>
        <msgUID>25250</msgUID>
    </header>
    <body>
        <scanDeviceComplInd>
            <obj objID="aisgPort:0">
                <numALDs>5</numALDs>
            </obj>
        </scanDeviceComplInd>
    </body>
</msg>
```

8.9.2 AISG Layer 7 message and Alarm transfer

The procedures in this clause allow AISG Layer 7 messages to be transferred between the REC and an ALD, as well as allowing AISG Alarms to be transferred from an ALD to the REC.

Precondition: The FST of the corresponding AISGALD object in the RE shall be in state OPERATIONAL (i.e. an AISG Layer 2 connection shall be established).

The AISG layer 7 messages are defined in TS 125 466 [11] and the AISG specifications [12]. Figure 8.9.2-1 shows how AISG Layer 7 messages are transferred between REC and ALD.

Further details of downlink AISG Layer 7 message transfer is described in clause 8.9.2.1.

Further details of uplink I-frame data transfer - including both AISG Layer 7 messages and alarms - are described in clause 8.9.2.2.

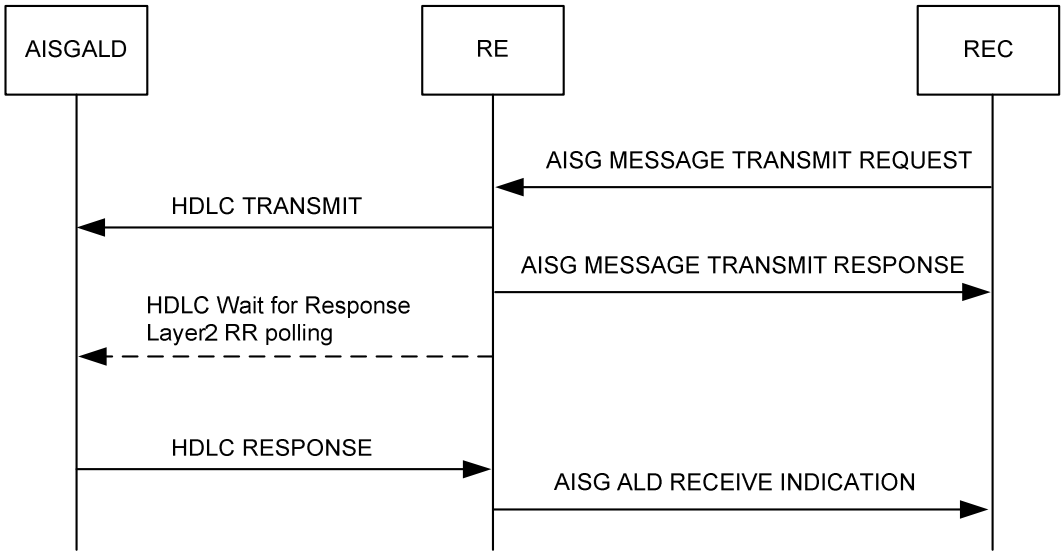

Figure 8.9.2-1: AISG Layer 7 message transfer in uplink and downlink - successful case

8.9.2.1 AISG message transmission (between REC and RE)

The AISG message transmission procedure is used to send a Layer 7 message I-Frame payload (Info Field) to the RE for a given ALD object.

The AISG MESSAGE TRANSMIT REQUEST shall include the following IEs: *AISG ALD Object* IE. *Layer 7 message* IE.

The RE shall create the HDLC Frame by performing the following steps:

- setting the HDLC Address of the specified ALD Object
- configuring the HDLC control octet (Frame numbering)
- calculating the FCS value
- adding the flags and transparency
- inserting the *Layer 7 message* IE into the HDLC frame as the payload

8.9.2.1.1 Success

Once the HDLC frame has been successfully created, the RE shall transmit the HDLC frame towards the required ALD, and send the AISG MESSAGE TRANSMIT RESPONSE to the REC.

8.9.2.1.2 Failure

If the RE was unable to create and/or transmit the HDLC message towards the ALD, it shall send an AISG MESSAGE TRANSMIT RESPONSE message including the *Result* IE with the value set to one of the failure cause values listed below in the following cases:

- FAIL_UNKNOWN_OBJTYPE if the objTypeID part of the *Object ID* IE (objRefT) does not correspond to "AISGALD".
- FAIL_UNKNOWN_OBJECT if the parent or child object instance addressed by the *Object ID* IE does not exist.
- FAIL_ALD_UNAVAILABLE if the FST of the AISGALD object is not in state OPERATIONAL.
- FAIL_BUS_UNAVAILABLE if the bus represented by the parent AISG Port object is powered down and/or if the AST of the parent AISG Port object is in state LOCKED at the time of the request.

8.9.2.2 AISG ALD receive indication

The AISG ALD RECEIVE INDICATION is used to send the payload content of an I-Frame to the REC.

If the RE receives an I-frame from the ALD, the RE shall send an AISG ALD RECEIVE INDICATION message to the REC for each I-Frame received, and shall include the I-Frame payload (Info Field) as the *AISG I-Frame Payload* IE.

In some cases it may happen that an ALD responds to an incoming Layer 7 message or poll with multiple I-Frames (e.g. if an ALD has a message response and a pending alarm).

The RE shall not interpret the info-field of incoming I-frames, they are forwarded by an AISG ALD RECEIVE INDICATION to the REC. This means that both Alarms and uplink Layer 7 messages shall be treated the same way by the RE. It is up to the REC to handle the payload content.

8.9.2.3 Message parameter details

Message	Encoded name
AISG MESSAGE TRANSMIT REQUEST	`aisgMsgTxReq`
AISG MESSAGE TRANSMIT RESPONSE	`aisgMsgTxResp`
AISG ALD RECEIVE INDICATION	`aisgALDRxInd`

Message parameter details for AISG MESSAGE TRANSMIT REQUEST message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	`obj`	1	objRefT	objID = "aisgPort:X/aisgALD:Y"
Layer 7 Message (HDLC Payload)	`L7message`	1	base64Binary	

Message parameter details for AISG MESSAGE TRANSMIT RESPONSE message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Result	`result`	1	String, enumerated	Defaults from 7.4. + "FAIL_UNKNOWN_OBJECT", "FAIL_UNKNOWN_OBJTYPE", "FAIL_ALD_UNAVAILABLE", "FAIL_BUS_UNAVAILABLE"
Object ID	`objID`	1	objRefT	

Message parameter details for AISG ALD RECEIVE INDICATION message:

Parameter	Encoded name	Cardinality	Type	Additional Info
Object ID	`objID`	1	objRefT	
AISG I-Frame Payload (HDLC Payload)	`L7message`	1	base64Binary	

The value of the message header IE *msgUID* of an AISG ALD RECEIVE INDICATION message shall be set to "0".

8.9.2.4 Message encoding

XML Schema of message bodies:

```
<!-- Message body: AISG Message Transmit Request -->
<xsd:complexType name="aisgMsgTxReqT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="L7message" type="xsd:base64Binary"/>
            </xsd:sequence>
```

```
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!-- AISG Message Transmit specific failure codes -->
<xsd:simpleType name="aisgMsgTxRespResT">
   <xsd:union memberTypes="ori:resT">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FAIL_UNKNOWN_OBJTYPE"/>
            <xsd:enumeration value="FAIL_UNKNOWN_OBJECT"/>
            <xsd:enumeration value="FAIL_ALD_UNAVAILABLE"/>
            <xsd:enumeration value="FAIL_BUS_UNAVAILABLE"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>


<!-- Message body: AISG Message Transmit Response -->
<xsd:complexType name="aisgMsgTxRespT">
   <xsd:sequence>
      <xsd:element name="result" type="ori:aisgMsgTxRespResT"/>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<!-- Message body: AISG ALD Receive Indication -->
<xsd:complexType name="aisgALDRxIndT">
   <xsd:sequence>
      <xsd:element name="obj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="L7message" type="xsd:base64Binary"/>
            </xsd:sequence>
            <xsd:attribute name="objID" type="ori:objRefT"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

Message examples:

```
<!-- Note that message data here is just an example of base64 encoded data
    and does not form a valid AISG L7 message   -->
<!-- Example: AISG Message Transmit Request -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>REQ</msgType>
        <msgUID>25375</msgUID>
    </header>
    <body>
        <aisgMsgTxReq>
            <obj objID="aisgPort:0/aisgALD:3">
                <L7message>

                        0A6694F673A2010619930307199631032001120220003

                </L7message>
            </obj>
        </aisgMsgTxReq>
    </body>
</msg>

<!-- Example: AISG Message Transmit Response -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>RESP</msgType>
        <msgUID>25375</msgUID>
    </header>
    <body>
```

```
        <aisgMsgTxResp>
            <result>SUCCESS</result>
            <obj objID="aisgPort:0/aisgALD:3"/>
        </aisgMsgTxResp>
    </body>
</msg>

<!-- Example:  AISG ALD Receive Indication -->
<msg xmlns="http://uri.etsi.org/ori/002-2/v1.1.1">
    <header>
        <msgType>IND</msgType>
        <msgUID>0</msgUID>
    </header>
    <body>
        <aisgALDRxInd>
            <obj objID="aisgPort:0/aisgALD:5">
                <L7message>16243A2018061964C315031962A2</L7message>
            </obj>
        </aisgALDRxInd>
    </body>
</msg>
```

9 Specified object types/parameters and fault types

9.1 Specified object types and their associated parameters

The following clauses describe the object types defined within the RE Resource Model, as well as their associated parameters and permitted states. The encoded name of each object type and parameter, and the data type and range of values applicable for each parameter shall be used for OCP encoding of the objects/parameters.

9.1.1 RE Object

Object Description:

The object type represents the RE itself, i.e. all capabilities or configuration items which are present once per RE.

Permitted States:

Not Applicable

Object lifecycle:

Static, single instance

Encoded Name:

"RE"

Parameters:

Table 9.1.1-1: RE object parameters

Parameter	Description	Type	Range
Vendor ID Encoded name: "vendorID"	Vendor ID of RE. The value shall be the same 3 character vendor code as used in the vendor identification code (code 201) of the DHCP procedure (see 10.1.1.1). Management type: RO	String, Length = 3	
Product ID Encoded name: "productID"	Product ID of RE. Management type: RO	String, maxLength = 80	
Product Revision Encoded name:	Product revision of RE. Management type: RO	String, maxLength = 80	

Parameter	Description	Type	Range
"productRev"			
Serial Number Encoded name: "serialNumber"	Serial Number of equipment. Management type: RO	String, maxLength = 40	
Protocol Version Encoded name: "protocolVer"	Version of the OCP protocol supported. In the present document, the Protocol Version shall be reported as "1.1" Management type: RO	String, maxLength = 10	
AGC Target RMS Level configuration granularity Encoded name: "agcTargetLevCfgGran"	Indicates whether configuration of Target RMS Level is "per Rx Signal Path" or "per RE". See [3] for the definition of Target RMS Level. Management type: RO	String, enumerated	Valid values: "RX_SIG_PATH", "RE"
AGC Settling Time configuration granularity Encoded name: "agcSettlTimeCfgGran"	Indicates whether configuration of Settling Time is "per Rx Signal Path" or "per RE". See [3] for the definition of Settling Time. Management type: RO	String, enumerated	Valid values: "RX_SIG_PATH", "RE"
AGC settling time capability Encoded name: "agcSettlTimeCap"	Indicates the supported values of N, where N is defined in the formula in clause 7.1.2 of [3] that defines the UL AGC Settling Time. b0 : N=0 0 : not supported 1 : supported b1 : N=1 0 : not supported 1 : supported b2 : N=2 0 : not supported 1 : supported b3 N=3 0 : not supported 1 : supported b4 : N=4 0 : not supported 1 : supported b5 : N=5 0 : not supported 1 : supported b6 : N=6 0 : not supported 1 : supported b7 : N=7 0 : not supported 1 : supported b8 : N=8 0 : not supported 1 : supported b9 : N=9 0 : not supported 1 : supported	HexBinary, length = 2 (16 bits = 2 octets)	Valid values: 0000 … 1FFF

Parameter	Description	Type	Range
	b10 : N=10 0 : not supported 1 : supported b11 : N=11 0 : not supported 1 : supported b12 : N=12 0 : not supported 1 : supported b13..b15: reserved (all "0") Management Type: RO		
Uplink AGC Target RMS Level Encoded name: "ulAgcTargetLev"	This is the configured Target RMS Level of the AxC (as defined in clause 7.1.2 of [3]). If "AGC Target RMS Level configuration granularity" takes the value "RX_SIG_PATH", the value shall be ignored. Management Type: R/W-locked	Unsigned Byte	6...32[dec], in steps of 1[dec].
Uplink AGC Settling Time Encoded name: "ulAgcSettlTime"	This is the configured value N, where N is defined in the formula in clause 7.1.2 of [3] that defines the UL AGC Settling Time for the AxC. If "AGC Settling Time configuration granularity" takes the value "RX_SIG_PATH", the value shall be ignored. Management Type: R/W-locked	Unsigned Byte	0,..,12 [dec]

9.1.2 Physical Antenna Port Object

Object Description:

The Physical Antenna Port object type reflects the RE antenna port. Each individual polarization maps to a different Physical Antenna Port. For arrays, the set of antenna elements that serve the same Signal Path shall be considered a single physical antenna port.

Permitted States:

Not Applicable

Object lifecycle:

Static

Encoded Name:

"antPort"

Parameters:

Table 9.1.2-1: Physical Antenna Port object parameters

Parameter	Description	Type	Range
Physical Antenna Port Label Encoded name: "portLabel"	This is the name provided to the Physical Antenna Port by the RE vendor, as indicated on the RE housing (see note). Management Type: RO	String. maxLength = 80	
NOTE: It is recommended that, for future ORI compliant products, the Physical Antenna Port Label is the Physical Antenna Port object name.			

9.1.3 TxSigPath Object

Object Description:

The Tx Signal Path is responsible for converting the AxC container contents for one AxC (see [1]) received from the reference ORI Link into a physical signal to be transmitted through the reference Physical Antenna Port and over the radio interface.

There are 2 types of Tx Signal Path object:

- TxSigPath_UTRA-FDD: Where the radio standard used is UTRA-FDD
- TxSigPath_E-UTRA: Where the radio standard used is E-UTRA

Permitted States:

All AST and FST states

AST initial state is LOCKED when the object is created.
FST initial state is NOT OPERATIONAL when the object is created.

Transition from LOCKED to UNLOCKED:

- Precondition:
  - The referenced ORI Link object shall be in FST state "Operational". Unlocking shall be denied otherwise.

Object lifecycle:

Dynamic

Encoded name:

TxSigPath_UTRA-FDD: "TxSigPath_UTRAFDD"
TxSigPath_E-UTRA: "TxSigPath_EUTRA"

Parameters:

Table 9.1.3-1a: TxSigPath UTRA-FDD object parameters

Parameter	Description	Type	Range
DL Cal_{RE}_MAX Encoded name: "dlCalRFMax"	Max possible buffer in RE for DL timing calibration expressed in T_C/16 (minimum value 1*Tc). Management type: RO	Unsigned short	16…8191 [Tc/16]
T2a Encoded name: "t2a"	Time delay (as defined in CPRI). Management type: RO	Unsigned Int	0…66667 Units: nanoseconds
DL Cal_{RE} Encoded name: "dlCalRF"	Time delay value to enable DL timing delay calibration expressed in T_C/16. Management type: R/W-Locked	Unsigned short	0… DL Cal_{RE}_MAX [Tc/16]
Max Transmission Power Encoded name: "maxTxPwr"	Maximum Power of the Tx signal Path (AxC) at the RE (Tx) antenna port, as specified in clause 7.1.2 of [3]. Management type: R/W	Unsigned short, fractionDigits = 1	Unit: dBm Range: 0..50 Step: 0.1 dB Example: 42,5 dBm is encoded as "425"
AxC container start position parameter (W) Encoded name: "axcW"	The parameter W used together with parameter B defines the position of the first bit of the AxC container within the Basic Frame. Parameter W is defined in CPRI specification [1]. (Also see note.) Management type: R/W-Locked	Unsigned byte	1…15 Step: 1
AxC container start position parameter (B) Encoded name: "axcB"	The parameter B used together with parameter W defines the position of the first bit of the AxC container within the Basic Frame. Parameter B is defined in CPRI specification [1]. (Also see note.) Management type: R/W-Locked	Unsigned byte	0…79 Step: 1
ORI Link Encoded name: "oriLink"	The ORI Link on which the required AxC container is mapped. Management type: R/W-Locked	objRefT	"oriLink:n"
UARFCN Encoded name: "uarfcn"	The downlink UARFCN of the UTRA carrier, as defined in clause 5.4.3 in TS 125 104 [19]. Management type: R/W-Locked	Unsigned Int	0…16383
Physical Antenna Port Encoded name: "antPort"	Reference to antenna port object representing the antenna port through which the RF signal of this TxSigPath shall be transmitted. Management type: R/W-Locked	objRefT	"antPort:x"
NOTE: Despite that AxC container mapping Option 1 (packed position) is chosen for downlink in clause 7.1.2 of [3], the REC selects the values of the parameters W and B to indicate the position of the first bit of the AxC container in the way described in [1] for flexible AxC container position (Option 2) instead of using AxC number.			

Table 9.1.3-1b: TxSigPath_E-UTRA object parameters

Parameter	Description	Type	Range
E-UTRA Channel Bandwidth Encoded name: "chanBW"	This is the channel bandwidth of the E-UTRA carrier, as defined in TS 136 104 [20]. Management type: RW-Locked	Unsigned short, fraction digits = 1	1,4 / 3 / 5 / 10 / 15 / 20 Units: MHz
DL Cal_{RE}_MAX Encoded name: "dlCalREMax"	Max possible buffer in RE for DL timing calibration expressed in T_C/16 (minimum value 1*Tc). Management type: RO	Unsigned short	16…8191 [Tc/16]
T2a Encoded name: "t2a"	Time delay (as defined in CPRI). Management type: RO	Unsigned int	0…66667 Units: nanoseconds
DL Cal_{RE} Encoded name: "dlCalRE"	Time delay value to enable DL timing delay calibration expressed in T_C/16. Management type: R/W-Locked	Unsigned short	0… DL Cal_{RE}_MAX [Tc/16]
Max Transmission Power Encoded name: "maxTxPwr"	Maximum Power of the Tx signal Path (AxC) at the RE (Tx) antenna port, as specified in 7.1.1 of [3]. Management type: R/W	Unsigned short, fractionDigits = 1	Unit: dBm Range: 0..50 Step: 0,1 dB Example: 42,5 dBm is encoded as "425"
AxC container start position parameter (W) Encoded name: "axcW"	The parameter W used together with parameter B defines the position of the first bit of the AxC container within the Basic Frame. Parameter W is defined in CPRI specification [1]. (Also see note.) Management type: R/W-Locked	Unsigned byte	1…15 Step: 1
AxC container start position parameter (B) Encoded name: "axcB"	The parameter B used together with parameter W defines the position of the first bit of the AxC container within the Basic Frame. Parameter B is defined in CPRI specification [1]. (Also see note.) Management type: R/W-Locked	Unsigned byte	0…79 Step: 1
ORI Link Encoded name: "oriLink"	The ORI Link on which the required AxC container is mapped. Management type: R/W-Locked	objRefT	"oriLink:n"
EARFCN Encoded name: "earfcn"	The downlink EARFCN of the E-UTRA carrier, as defined in TS 136 104 [20]. Management type: R/W-Locked	Unsigned short	0…65535
Physical Antenna Port Encoded name: "antPort"	Reference to Physical Antenna Port object representing the antenna port through which the RF signal of this TxSigPath shall be transmitted. Management type: R/W-Locked	objRefT	"antPort:x"
NOTE: Despite that AxC container mapping Option 1 (packed position) is chosen for downlink in clause 7.1.1 in [3], the REC selects the values of parameters W and B to indicate the position of the first bit of the AxC container in the way described in [1] for flexible AxC container position (Option 2) instead of using AxC number.			

9.1.4 RxSigPath Object

Object Description:

The Rx Signal Path is responsible for converting the physical signal received from the radio interface through the reference Physical Antenna Port into the contents of the AxC container for one AxC (see [1]) sent over the reference ORI Link and, for UTRA-FDD, mapping the RTWP measurement report to the correct location within the RTWP control word.

There are 2 types of Rx Signal Path object:

- RxSigPath_UTRA-FDD: Where the radio standard used is UTRA-FDD
- RxSigPath_E-UTRA: Where the radio standard used is E-UTRA

Permitted States:

All AST and FST states

AST initial state is LOCKED when the object is created.
FST initial state is NOT OPERATIONAL when the object is created.

Transition from LOCKED to UNLOCKED:

- Precondition:
  - The referenced ORI Link object shall be in FST state "Operational". Unlocking shall be denied otherwise.

Object lifecycle:

Dynamic

Encoded name:

RxSigPath_UTRA-FDD: "RxSigPath_UTRAFDD"
RxSigPath_E-UTRA: "RxSigPath_EUTRA"

Parameters:

Table 9.1.4-1a: RxSigPath UTRA-FDD object parameters

Parameter	Description	Type	Range
UL Cal_{RE}_MAX Encoded name: "ulCalREMax"	Max possible buffer in RE for UL timing calibration expressed in T_C/2. Management type: RO	Unsigned short	2…1023 [Tc/2]
Ta3 Encoded name: "ta3"	Time delay (as defined in CPRI). Management type: RO	Unsigned int	0…66667 Units: nanoseconds
UL Cal_{RE} Encoded name: "ulCalRE"	Time delay value to enable UL timing delay calibration expressed in T_C/2. Management type: R/W-Locked	Unsigned short	0… UL Cal_{RE}_MAX [Tc/2]
AxC container start position parameter (W) Encoded name: "axcW"	The parameter W used together with parameter B defines the position of the first bit of the AxC container within the Basic Frame. Parameter W is defined in CPRI specification [1]. (Also see note.) Management type: R/W-Locked	Unsigned byte	1…15 Step: 1

Parameter	Description	Type	Range
AxC container start position parameter (B) Encoded name: "axcB"	The parameter B used together with parameter W defines the position of the first bit of the AxC container within the Basic Frame. Parameter B is defined in CPRI specification [1]. (Also see note.) Management type: R/W-Locked	Unsigned byte	0…79 Step: 1
AxC RTWP group Encoded name: "rtwpGroup"	Identifies location of the RTWP measurement value for this AxC within the RTWP control word. Management type: R/W-Locked	Unsigned byte	1…64
ORI Link Encoded name: "oriLink"	The ORI Link on which the required AxC container and AxC RTWP group are mapped. Management type: R/W-Locked	objRefT	"oriLink:n"
Physical Antenna Port Encoded name: "antPort"	Reference to Physical Antenna Port object representing the antenna port from which the RF signal of this RxSigPath shall be received. Management type: R/W-Locked	objRefT	"antPort:x"
UARFCN Encoded name: "uarfcn"	The uplink UARFCN of the UTRA carrier, as defined in clause 5.4.3 in TS 125 104 [19]. Management type: R/W-Locked	Unsigned short	0…16383
Uplink Feeder Adjustment Encoded name: "ulFeedAdj"	The adjustment value to be applied by RE to the uplink RTWP measurement, to compensate for loss or gain in feeder cable. Management type: R/W	short, fractionDigits = 1	-50….+50 Unit: dB Step: 0,1 dB Example: -2,5 dB is encoded as "-25"
Uplink AGC Target RMS Level Encoded name: "ulTgtRMSLvl"	This is the configured Target RMS Level of the AxC (as defined in clause 7.1.2 of [3]). Management Type: R/W-locked	Unsigned byte	6...32[dec], in steps of 1[dec].
Uplink AGC settling time Encoded name: "ulAGCSetlgTime"	This is the configured value N, where N is defined in the formula in clause 7.1.2 of [3] that defines the UL AGC Settling Time for the AxC. Management Type: R/W-locked	Unsigned byte	0,..,12
NOTE: The REC selects the values of the parameters W and B to indicate the position of the first bit of the AxC container in line with the uplink AxC container mapping in clause 7.1.1 of [3].			

Table 9.1.4-1b: RxSigPath_E-UTRA object parameters

Parameter	Description	Type	Range
E-UTRA Channel Bandwidth Encoded name: "chanBW"	This is the channel bandwidth of the E-UTRA carrier, as defined in TS 136 104 [20]. Management type: RW-Locked	Unsigned short, fractionDigits = 1	1,4 / 3 / 5 / 10 / 15 / 20 Units: MHz Example: 1,4 MHz is encoded as "14"; 10 MHz is encoded as "100".
UL Cal_{RE}_MAX Encoded name: "ulCalREMax"	Max possible buffer in RE for UL timing calibration expressed in $T_C/2$. Management type: RO	Unsigned short	2…1023 [Tc/2]
Ta3	Time delay (as defined in CPRI).	Unsigned int	0…66667

Parameter	Description	Type	Range
Encoded name: "ta3"	Management type: RO		Units: nanoseconds
UL Cal_{RE} Encoded name: "ulCalRE"	Time delay value to enable UL timing delay calibration expressed in $T_C/2$. Management type: R/W-Locked	Unsigned short	0… UL Cal_{RE}_MAX [Tc/2]
AxC container start position parameter (W) Encoded name: "axcW"	The parameter W used together with parameter B defines the position of the first bit of the AxC container within the Basic Frame. Parameter W is defined in CPRI specification [1]. (Also see note.) Management type: R/W-Locked	Unsigned byte	1…15 Step: 1
AxC container start position parameter (B) Encoded name: "axcB"	The parameter B used together with parameter W defines the position of the first bit of the AxC container within the Basic Frame. Parameter B is defined in CPRI specification [1]. (Also see note.) Management type: R/W-Locked	Unsigned byte	0…79 Step: 1
ORI Link Encoded name: "oriLink"	The ORI Link on which the required AxC container is mapped. Management type: R/W-Locked	objRefT	"oriLink:n"
Physical Antenna Port Encoded name: "antPort"	Reference to Physical Antenna Port object representing the antenna port from which the RF signal of this RxSigPath shall be received. Management type: R/W-Locked	objRefT	"antPort:x"
EARFCN Encoded name: "earfcn"	The uplink EARFCN of the E-UTRA carrier, as defined in TS 136 104 [20]. Management type: R/W-Locked	Unsigned short	0…65535
NOTE: Despite that AxC container mapping Option 1 (packed position) is chosen for uplink in clause 7.1.2 in [3], the REC selects the values of the parameters W and B to indicate the position of the first bit of the AxC container in the way described in [1] for flexible AxC container position (Option 2) instead of using AxC number.			

9.1.5 ORI Link Object

Object Description:

The ORI Link object represents the ORI link and carries per-ORI link information as well as physical ORI port information of the individual ORI links.

In the current version of the specification, the RE shall only contain ORI ports of type "slave". For ORI ports supported by the RE that are not of type "slave", no corresponding instance of the ORI Link object shall be created by the RE.

It is assumed that the physical transmission link/transceiver module for the ORI Link may be inserted into the ORI port during RE operation or before the RE is operating.

Permitted States:

AST:

In state LOCKED:

- The link shall be in CPRI state A (Standby); see [1], clause 4.5.3.

In state UNLOCKED:

- The ORI link object shall be in one of CPRI states A, B, C, D, E, F (see [1]) if the link is an active link.
- The ORI link object shall be in one of CPRI states A, B, C, D, E, G (see [1]) if the link is a passive link.

Transition from UNLOCKED to LOCKED:

- **Preconditions:**
  - The ORI Link shall be a passive link. Locking shall be denied with failure code FAIL_RESOURCE_INUSE if the link is an active link
- **Actions:**
  - The link shall be shut down, and transition to CPRI state A (Standby)

Transition from LOCKED to UNLOCKED:

- **Preconditions**
  - none
- **Actions:**
  - Unlocking triggers transition to CPRI state B if the FST is not in state "disabled".

NOTE: If the port is not equipped (FST = "disabled"), the unlock request itself is **not** denied.

Default AST is "unlocked".

FST:

The functional state of this object shall be reported as:

- Disabled, if there is no physical transmission link/transceiver module equipped in the physical ORI port. All other functional states assume the physical port is equipped, and a state change shall occur to one of those states if a physical transmission link/transceiver module is inserted during RE operation.
- notOperational, if the link is in one of the CPRI states A, B. C, D, E.
- Operational, if the link is in CPRI state F or G.
- Failed, if failure of the physical ORI port hardware is detected, or if LOS or LOF (clause 4.2.10 of [1]) is detected.

Object lifecycle:

Static; one instance per physical ORI (slave) port (equipped or not).

Encoded name:

"oriLink"

Parameters:

Table 9.1.5-1: ORI Link object parameters

Parameter	Description	Type	Range
Physical ORI link port label Encoded name: "portLabel"	The name of the physical ORI link port as indicated on the RE housing. Management type: RO	String maxLength = 80	
Supported line bit rates	This indicates the set of ORI line bit rates supported on this ORI link (see note).	hexBinary, Length = 2	0002 … 003E

Parameter	Description	Type	Range
Encoded name: "bitRateSupport"	"0000 0000 $0b_6b_5b_4b_3b_2b_1b_0$" b0 : reserved ("0") b1 : line bit rate Option 2 (1 228,8 Mbps): 0 : not supported 1 : supported b2 : line bit rate Option 3 (2 457,6 Mbps): 0 : not supported 1 : supported b3 : line bit rate Option 4 (3 072,0 Mbps): 0 : not supported 1 : supported b4 : line bit rate Option 5 (4 915,2 Mbps): 0 : not supported 1 : supported b5 : line bit rate Option 6 (6 144,0 Mbps): 0 : not supported 1 : supported b15-b6: reserved (all "0") Management Type: RO		
Local PORT_ ID Encoded name: "localPortID"	Unique port identifier of the local end of the ORI link. See clause 6.3.1 of [3] for further definition. Management Type: RO	hexBinary, Length = 8	0000000000000000 … FFFFFFFFFFFFFFFF
Remote PORT_ID Encoded name: "remotePortID"	Unique port identifier of the remote end of the ORI link. See clause 6.3.1 of [3] for further definition. Management Type: RO	hexBinary, Length = 8	0000000000000000 … FFFFFFFFFFFFFFFF
Toffset Encoded name: "toffset"	Time delay component as defined in CPRI specification [1], clause 4.2.9.2. The delay value is reported in nanoseconds. Management Type: RO	Unsigned int	0…66667 Units: nanoseconds
ORI link type Encoded name: "oriLinkType"	Determines the ORI link type (active /passive). It shall be set by the RE to reflect the current ORI link type of this port when the FST becomes Operational. Management Type: RO	String. enumerated	Enumerated: "ACTIVE", "PASSIVE"
NOTE: Upon insertion of a ORI link transceiver module the RE implementation may further restrict the line bit rates supported on the ORI link based on the capabilities of the transceiver module.			

9.1.6 External Event Port Object

Object Description:

The External Event Port represents one physical contact that is used to monitor an external event, e.g. "cabinet door open/closed", "air conditioning on/off", "antenna tower light on/off". The physical contact shall be either "open" (not connected) or "closed" (connected).

There shall be an object instance for each physical external event port.

Permitted States:

AST: Locked, Unlocked.

Default AST is "locked".

FST: The functional state of this object shall be reported as:

- Disabled, if the object is AST locked.
- notOperational, if the object is AST unlocked and the contact is "open" or "not connected".
- Operational, if the object is AST unlocked and the contact is "closed".

Default FST is "disabled".

Object lifecycle:

Static

Encoded name:

"extEvent"

Parameters:

Table 9.1.6-1: External Event Port object parameters

Parameter	Description	Type	Range
Physical External Event Port Label Encoded name: "portLabel"	This is the name provided to the External Event Port by the RE vendor. Management Type: RO	String. maxLength = 80	

9.1.7 AISGPort Object

Object Description:

This static object represents a physical AISG port. Antenna Line Devices (ALDs) which are connected on this AISG Port are contained as child objects.

Permitted States:

AST:

In state LOCKED:

- AISG Layer 7 message/alarm transferto the bus/port represented by the object is blocked
- Device scan is permitted
- Creation/Deletion of AISGALD objects is possible
- Powering down of the bus (AISG Bus Power enable = False) is permitted

In state UNLOCKED:

- Bus is in service, Layer 7 AISG messaging is enabled
- Device scan is not permitted
- Modification and creation/deletion of AISGALD objects is not permitted
- Powering down the bus is not permitted

Transition from LOCKED to UNLOCKED:

When requested to unlock, the AST shall transit from LOCKED to UNLOCKED when the following preconditions are met:

- Precondition: Bus is powered (AISG Bus Power enable = True)
- Precondition: There is no on-going Device Scan procedure.

The transition shall fail with code FAIL_PRECONDITION_NOTMET if one of the preconditions above is not met.

Transition from UNLOCKED to LOCKED:

When requested to lock, the AST shall transit from UNLOCKED to LOCKED when the following preconditions are met:

- Precondition: There is no AISG Message Transmission procedure is on-going in the RE., If there is, then this procedure shall be completed normally before locking (i.e. the AST shall change after the transfer is completed).

The AST is initially LOCKED when RE is powered up or after RE reset

FST: none.

Object lifecycle:

Static, one object instance per physical AISG port.

Containment:

Contains: [0..*] of AISGALD objects

Encoded name:

"aisgPort"

Parameters:

Table 9.1.7-1: AISGPort object parameters

Parameters	Description	Type	Range
AISG Port Label Encoded name: "portLabel"	This is the name provided to the Physical AISG Port by the RE vendor, as indicated on the RE housing (see note). In case the AISG Port is of type "Modem" this label shall be the same as the corresponding Antenna Port. Management type: RO	String maxLength = 80	
AISG Bus Power enable Encoded name: "busPowerEnable"	Controls the output power of the AISG port. The bus power can only be disabled when the AISGPort is in state LOCKED. As the bus needs to be powered to provide service, an unlock-request shall fail if the bus power is disabled. Management type : RW-locked Default value: True (Power enabled)	Boolean	
NOTE: If the AISG Port is not of type "Modem", it is recommended that the AISG Port Label is the same as the AISG Port object name.			

9.1.8 AISGALD Object

Object Description:

This object represents a single ALD on an AISG bus connected to the RE.

The AISGALD object is contained in the AISGPort object instance which represents the AISG port/bus on which the ALD has been found.

Permitted States:

FST:

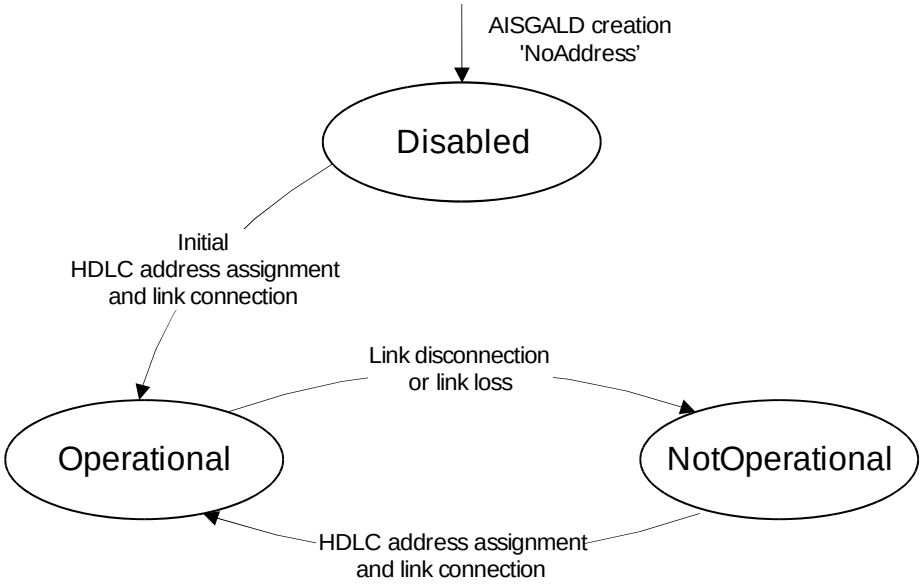

Figure 9.1.8-1: AISGALD FST state diagram

- Disabled
  The ALD object stays in this state if the RE was not able to initially assign a HDLC Address after creation of the object (AISG connection in state "NoAddress" or "AddressAssigned" as defined in clause 4.6 "State Model" of [10]).
  The RE shall undertake periodic attempts to assign the HDLC address and to establish the link. The minimum period shall not be less than 1 minute.

  AISG Layer 7 messages requested to be sent to ALDs shall be rejected by the RE.

NOTE 1: Successful link establishment leads to transition to FST "Operational".

NOTE 2: AISGALD objects that have been configured in advance, but whose respective ALD has not been installed remain in this state.

- Operational
  The ALD is in state Connected as defined in clause 4.6 "State Model" of [5].

  The RE shall ensure that no link timeout (as described in clause 4.10 of [5]) occurs. This includes transmission of polling messages to the ALD if no L7 messages were targeted to this ALD. The polling period is out of the scope of ORI it shall be fast enough that no ALD reaches link timeout.

  AISG Layer 7 message transfer to ALDs is permitted.

- NotOperational
  The ALD is in states NoAddress or AddressAssigned as defined in clause 4.6 "State Model" of [5].
  The ALD object shall change its FST state to notOperational if it gets disconnected or runs into link timeout.
  The RE shall undertake periodic attempts to re-assign the address and to re-establish the link. The minimum period shall not be less than 1 minute.

  AISG Layer 7 messages requested to be sent to ALDs shall be rejected by the RE.

NOTE 3: This state may either be transient (e.g. after lightning strike or ALD internal reset), or permanent (ALD configured, but defective or removed).

FST shall be initially DISABLED when the object is created.

AST: inherited from parent AISGPort object.

Containment:

Object contained in AISGPort object.

Object lifecycle:

Dynamic.

The object is created/deleted/reconfigured in the RE using one of the following approaches:

- Created explicitly by the REC using the procedure in clause 8.4.1, including at least the *Unique ID* or *Device Type* parameters to allow an address assignment according 3GPP specification. The object shall only be deleted or re-configured by the REC using this approach.

- Created by the RE as result of a device scan procedure (clause 8.9.1). In this approach, the object shall only be deleted or re-configured by the RE, and only following a subsequent device scan procedure.

Mixing the two approaches is beyond the scope of the present document and not recommended.

Encoded name:

"aisgALD"

Parameters:

Table 9.1.8-1: AISGALD object parameters

Parameters	Description	Type	Range
Device Type Encoded name: "deviceType"	Device Type of the ALD according to AISG/3GPP Management type: RW-Locked	Unsigned Byte	
UID Encoded name: "UID"	Unique ID according to AISG/3GPP As defined in [5], 3.1 Definitions Management type: RW-Locked	List of unsigned byte, minLength= 2, maxLength = 19 (See note)	
3GPP Release ID Encoded name: "releaseID"	Release of the 3GPP protocol, as defined in [10], 4.8.2. Protocol version. AISG1.1 devices shall set their 3GPP Release to 255. Management type: RO	Unsigned Byte	0 ..255
AISG version Encoded name: "aisgVersion"	Version of the used AISG Protocol, as defined in [12], 7.2, Protocol Version. AISG1.1 devices shall set their AISG version to 255. Management type: RO	Unsigned Byte	2 ..255
AISG Device Type Substance Version Parameter Encoded name: "deviceTypeVersion"	As defined in [12], 7.3, Device Type Substance Version Parameter Management type: RO	List of Unsigned Byte, length = 2 (See note)	0 ..255
Frame length Encoded name: "frameLength"	Maximum framelength for AISG Layer 7 message payload [Bytes] As defined in [5], 4.8.1 HDLC Parameters Management type: RO	Unsigned Short	min = 74
HDLC address Encoded name: "hdlcAddress"	Actual assigned HDLC Address Management type: RO	Unsigned Byte	

NOTE: Lists (arrays) of bytes shall be encoded as if they were defined according to the following XML Schema:

```
<xsd:simpleType name="byteArrayT">
    <xsd:list itemType="xsd:unsignedByte"/>
</xsd:simpleType>

<!-- Example: AISG UID: list with length 2...19  -->
<xsd:simpleType name="UidT">
    <xsd:restriction base="byteArrayT">
        <xsd:minLength value="2"/>
        <xsd:maxLength value="19"/>
    </xsd:restriction>
</xsd:simpleType>
```

An example of an encoded UID value with 4 bytes according to the schema given above is:

```
[…]  <param name="UID">127 80 20 5</param> […]
```

Individual elements of a list are separated using a "space" character (Unicode U+0020).

NOTE 4: AISG1.1 Devices do not have a dedicated version parameter at Layer 2. They may be detected by their XID answer type.

9.1.9 Log Object

Object Description:

The ORI Log object represents ORI defined and implementation specific logs on the RE. It is used to report and configure the log file upload related behaviour of the advertised log types.

For a more detailed description of the log concept (file content, log configuration), see clause 6.10 of the present document.

Permitted States:

none

Object lifecycle:

Static; one instance per log

Encoded name:

"Log"

Parameters:

Table 9.1.9-1: Log object parameters

Parameter	Description	Type	Range
Log Type ID Encoded name: "logTypeID"	Indicates the type of the log, which is either one of the ORI defined log types (ORI::FAULT in the present document), or a vendor specific log type. The type identifier shall consist of a 3-letter vendor prefix, and a log type identifier separated by a double colon. For ORI defined log types, the vendor prefix shall be "ORI"; for vendor specific log types, the prefix shall be the 3 letter vendor code as used in code 201 of the DHCP request (see clause 10.1.1). Management type: RO	String	maxLength = 40 Examples: "ORI::FAULT" "ABC::SYSLOG"
Description Encoded name: "description"	Textual short description of the log content (e.g. "Debug log", "OS system log"). This shall help service personnel to identify the log content. Management type: RO	String	maxLength = 80
Log Category Encoded name: "logCategory"	Management type: RO	String, enumerated	Valid values: "FAULT", "PERF", "SYSTEM", "CRASH", "APPLICATION", "DEBUG", "OTHER"
Maximum RE File size Encoded name: "maxREfileSize"	Maximum file size supported by the RE for this type of log in kBytes (multiples of 1 024 Bytes). Management type: RO	Unsigned Integer	"0" indicates that no limit is set.
Maximum REC File size Encoded name: "maxRECfileSize"	Maximum file size supported by the target FTP server for this type of log in kBytes (multiples of 1 024 Bytes). Management type: RW	Unsigned Integer	"0" indicates that no limit is set.
Enable REC Notification Encoded name: "enableNotification"	Enables/disables REC notification upon transfer file availability. If enabled (Enable REC Notification = "TRUE"), a FILE AVAILABLE INDICATION shall be generated by the RE whenever a transfer file becomes available in the first position of the upload queue. Management type: RW	Boolean	Default: "FALSE"
File Available Encoded name: "fileAvailable"	This parameter shall be set to "TRUE" by RE whenever one or more transfer files are available in the upload queue. It shall be reset to "FALSE" by the RE when no more transfer files are available. This parameter may by polled by the REC to determine whether there is a need to upload transfer files in case event driven reporting is not used. Management type: RO	Boolean	Default: "FALSE"

Parameter	Description	Type	Range
Overflow Behaviour Encoded name: "overflowBehaviour"	This parameter is used to report, and, where applicable, configure the behaviour in case of log buffer overflow. "STOP": the log stops recording until the buffer gets emptied by uploading the transfer file. "FIFO": oldest entries get discarded in favour of newer ones. For vendor logs the value reported by the RE is determined by the log implementation. For ORI defined log types, the default and possible behaviours are defined in conjunction with the log type definition. The REC can modify this where applicable. In case this is not applicable, the RE shall respond to modification attempts with failure code "FAIL_VALUE_OUTOF_RANGE" Management type: RW	Enumerated	Values: "STOP", "FIFO"
Recording Enabled Encoded name: "recordingEnabled"	This parameter is used to enable/disable logging for this log, where applicable. For vendor logs the default value reported by the RE is determined by the log implementation. The REC can modify this where applicable. In case this is not applicable, the RE shall respond to modification attempts with failure code "FAIL_VALUE_OUTOF_RANGE" Management type: RW	Boolean	
Log Period Encoded name: "logPeriod"	Determines the expiration period of the log. The expiration period is provided in units of seconds. The REC can modify this where applicable. In case this is not applicable, the RE shall respond to modification attempts with failure code "FAIL_VALUE_OUTOF_RANGE" Management type: RW	Unsigned long	"0" indicates no expiration (timer deactivated) Default: "0"
Timer Type	Determines the type of Log Period expiration behaviour. See clause 6.10.4 for a detailed description. The REC can modify this where applicable. In case this is not applicable, the RE shall respond to modification attempts with failure code "FAIL_VALUE_OUTOF_RANGE" Management type: RW	String, enumerated	Valid values: "ONE_SHOT", "PERIODIC"

9.2 Specified fault types

The table 9.2-1 indicates the specified faults used by RE and their relevance.

Table 9.2-1: Specified faults used by RE

Fault Type	Fault ID	Description	Fault Severity	Primary Object	Additional affected Object
RE power supply input under-voltage	FAULT_EXT_SUPPLY_UNDERVOLT	RE power supply input voltage below pre-defined and vendor specific range.	Failed	RE	*Affected Signal Paths*
			Degraded	RE	*Affected Signal Paths*
			Warning	RE	*Affected Signal Paths*
Over-temperature	FAULT_RE_OVERTEMP	Temperature reaches upper limit of pre-defined and vendor specific range.	Failed	RE	*Affected Signal Paths*
			Degraded	RE	*Affected Signal Paths*
			Warning	RE	*Affected Signal Paths*
Signal processing related faults					
Digital input signal level overdrive	FAULT_DIG_IN_OVERDRIVE	Digital signal level above pre-defined and vendor specific range for a significant period of time.	Failed	RE	*Affected Signal Paths*
			Degraded	RE	*Affected Signal Paths*
			Warning	RE	*Affected Signal Paths*
RF output overdrive	FAULT_RF_OUT_OVERDRIVE	Excessive RF level above pre-defined and vendor specific range for a significant period of time.	Failed	RE	*Affected Signal Paths*
			Degraded	RE	*Affected Signal Paths*
			Warning	RE	*Affected Signal Paths*
Tx gain control failure	FAULT_TX_GAIN_FAIL	Tx gain cannot be adjusted to match required output power	Failed	RE	*Affected Signal Paths*
			Degraded	RE	*Affected Signal Paths*
			Warning	RE	*Affected Signal Paths*
Rx gain control failure	FAULT_RX_GAIN_FAIL	Rx gain cannot be adjusted to match required signal level (Rx path failure)	Failed	RE	*Affected Signal Paths*
			Degraded	RE	*Affected Signal Paths*
			Warning	RE	*Affected Signal Paths*
Antenna Port related faults					
VSWR out of range	FAULT_VSWR_OUTOF_RANGE	The *Voltage Standing Wave Ratio* at the affected antenna port has left pre-defined and vendor specific range	Failed	Physical Antenna Port	*Associated Signal Paths*
			Degraded	Physical Antenna Port	*Associated Signal Paths*
			Warning	Physical Antenna Port	*Associated Signal Paths*
Non AISG TMA malfunction	FAULT_NONAISG_TMA_MALFCT	Non AISG TMA malfunction; may affect transmission and reception performance	Failed	Physical Antenna Port	*Associated Signal Paths*
			Degraded	Physical Antenna Port	*Associated Signal Paths*
			Warning	Physical Antenna Port	*Associated Signal Paths*

Fault Type	Fault ID	Description	Fault Severity	Primary Object	Additional affected Object
AISG Port related fault					
AISG malfunction	FAULT_AISG_MALFCT	Hardware malfunction that affects an AISG port	Failed	AISG Port	*Associated Signal Paths*
			Degraded	AISG Port	*Associated Signal Paths*
			Warning	AISG Port	*Associated Signal Paths*

10 RE management procedures

10.1 RE Device management

10.1.1 OCP layer establishment and supervision between REC and RE

The procedure below describes the establishment and supervision of the OCP layer between REC and RE. Additional requirements for DHCP options and TCP options are described in clause 10.1.1.1 and clause 10.1.1.2 respectively.

Precondition:

The REC wants to establish the OCP layer to a specific RE and an ORI link (active) in CPRI state F exists to the specific RE.

Normal flow of events:

The normal OCP layer establishment procedure proposed is as follows.

1) The RE sets the TCP Connection Establishment (T_{TCE}) and the DHCP Server Response (T_{DSR}) timers to their default values.

2) The RE requests an IP address via DHCP according to RFC 2131 [16], and sets the DHCP Server Response Timer (T_{DSR}) for each server request.

3) The DHCP server receives the DHCP DISCOVER request, and evaluates the received data for compatibility and support and provides the RE with an IP address.

NOTE: The interface between the DHCP server and the REC application layers is out if the scope of the present document.

4) On receiving the IP address, the RE re-sets T_{TCE} and clears T_{DSR}.

5) The REC initiates a TCP Connection Establishment to the RE based on the information from DHCP server, according to RFC 793 [18]. The REC also sets the TCP Synchronization Timer (T_{TSY}) to be able to detect a TCP Connection Establishment failure.

6) When the TCP connection has been established, the REC clears T_{TSY}, the RE clears T_{TCE} and sets the TCP Link Monitoring Timer ($T_{TLM)}$ to a previously stored value, if available. If not, the default value shall be used.

7) The REC shall configure the TCP Link Monitoring Timer (T_{TLM}) of the RE by providing a timeout value through an initial Health Check Request. This also starts the Health Check Idle Timer (T_{HCI}) on the REC. The timeout value for (T_{HCI}) used by the REC should be at least 5 seconds lower than the timeout value of T_{TLM}.

8) The RE shall store the received new value of the TCP Link Monitoring timeout, and re-start the TCP Link Monitoring Timer using the new timeout value. The RE sets T_{TLM} to the stored value and sends a Health Check Response to REC.

End condition:

The OCP layer is successfully established and functioning in REC and RE.

The following steps are then taken by REC and RE to supervise the existence and functioning of the OCP layer:

- The RE shall re-start T_{TLM} after reception of any message.
  (See "abnormal conditions" clause below for RE action upon expiry of T_{TLM}).

- The REC shall re-start T_{HCI} after transmission of any message.
  If T_{HCI} expires, the REC shall send a Health Check Request message to the RE.

Timer definitions and timeout values:

Timers and default values shall be as defined in table 10.1.1-1.

Table 10.1.1-1: Timers and default values for OCP layer establishment/supervision

Name	Symbol	Default value	Located on
TCP Connection Establishment Timer	T_{TCE}	10 min	RE
DHCP Server Response Timer	T_{DSR}	10 sec	RE
TCP Synchronization Timer	T_{TSY}	15 sec	RE
TCP Link Monitoring Timer	T_{TLM}	50 sec; Minimum value: 30 sec	RE
Health Check Idle Timer	T_{HCI}	$T_{HCI} \leq T_{TLM}$ - 5 sec	REC

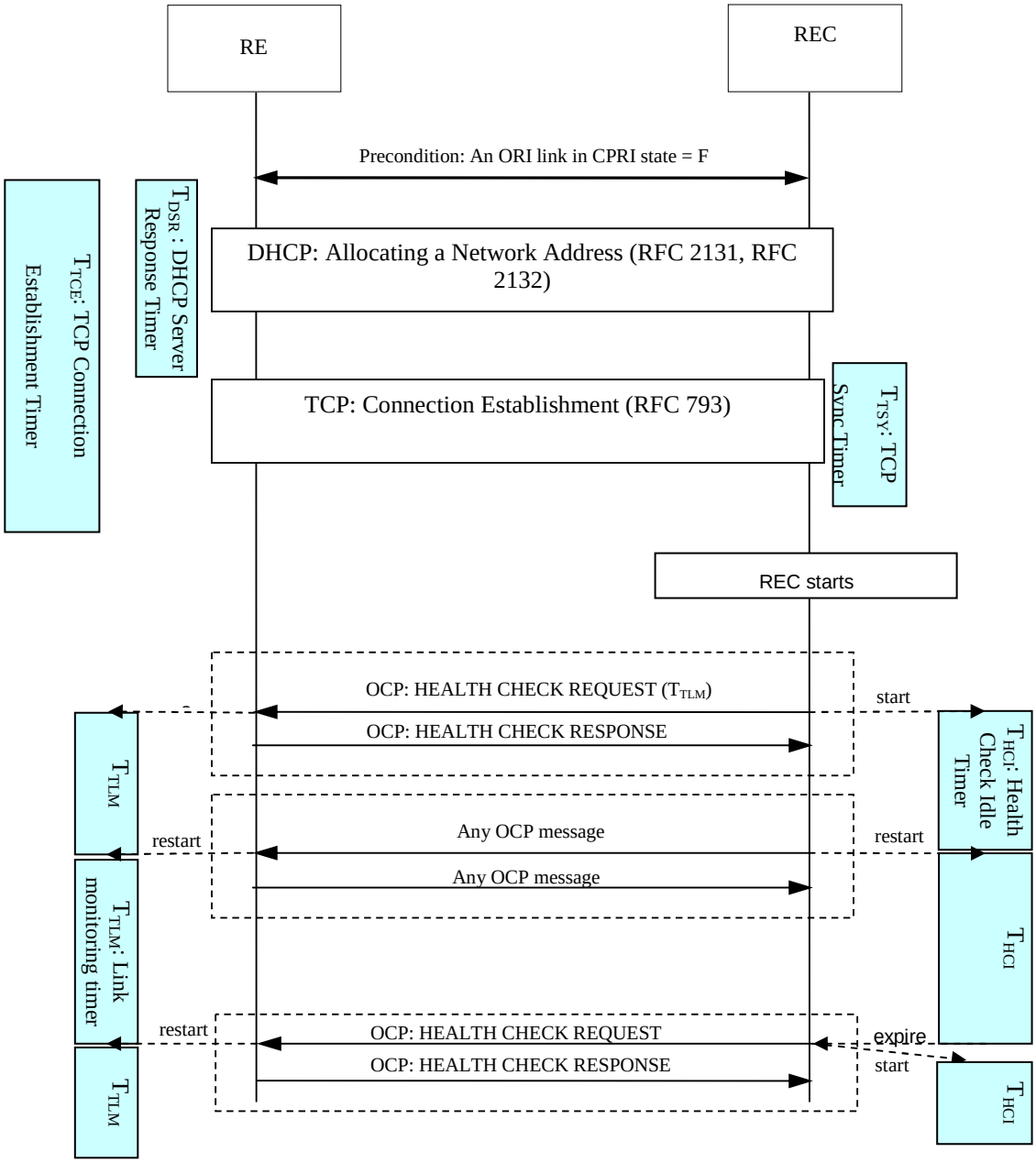

Figure 10.1.1-1: High level normal flow of events

Alternative and failure flow of events:

a) The REC/DHCP server detects a DHCP parameter issue.
Action: For the case of an ORI protocol version mismatch the typical and the intended behavior is that the normal flow of events continues and that the REC initiates a software download and restart of the RE in order to resolve the mismatch. For other issues this is unspecified.

b) The RE does not receive an expected response from the DHCP server within T_{DSR}
Action: Repeat the DHCP request. The repetitions shall continue until a response is received.

c) The TCP Connection Establishment timer (T_{TCE}) expires in RE.
Action: The RE shall initiate a restart corresponding to a power on restart.

d) The TCP Synchronization Timer ($T_{TSY)}$) expires in the REC implying that the TCP connection establishment failed.

Action: Is unspecified. However, a typical behavior would be to repeat the establishment attempts a number of times before further fault handling is invoked.

e) The TCP Link Monitoring timer (T_{TLM}) expires in RE.
Action: The RE shall perform a local end release of the TCP connection and prevent transmission on the radio interface. The RE may expect the REC to restart the TCP connection establishment at step 5.

f) The REC has sent a Health Check Request but does not receive a Health Check Response.
Action: The action taken by REC is unspecified. However, a typical behavior would be to repeat the health check attempts a number of times before further RE recovery action is invoked.

g) The REC has detected a TCP failure.
Action: Unspecified. The RE is expected to detect the same failure and will wait again on its TCP port (see failure d) above). The REC may consider performing some more TCP connection trials before further RE recovery action is invoked.

10.1.1.1 DHCP options

The following DHCP options (defined in RFC 2132 [17]) and content shall be used by the REC and RE as specified below.

- **DHCP option 60: Vendor class identifier**
  This option shall be used by ORI DHCP clients (RE) to identify the fact that the product is ORI compliant. The information shall contain the string "ORI-RE" of 6 characters.

  The DHCP server (REC) shall use option 43 to return the vendor-specific information to the client.

```
 Code   Len   Vendor class Identifier

+-----+-----+-----+-----+-----+-----+-----+-----+
|  60 |  6  |  O  |  R  |  I  |  -  |  R  |  E  |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

- **DHCP option 61: Client identifier**
  This option shall be used by DHCP clients (RE) to specify their unique identifier. It consists of a hardware type (set to 1) and hardware address (set to the MAC address).

```
 Code   Len   Type  Client-Identifier

+-----+-----+-----+-----+-----+---
|  61 |  7  |   1 |  i1 |  i2 | ...
+-----+-----+-----+-----+-----+---
```

- **DHCP option 43: Vendor specific information**
  This option shall be used by clients and servers to exchange the following information.

  The RE shall include the following Vendor Specific fields in the DHCP DISCOVER and the DHCP REQUEST:

  - **Code 200: ORI protocol version. Length = 2.**
    - Byte 1: Version ID (0..255)
    - Byte 2: Edition ID (0..255)

    ORI protocol version/edition described in the present document is set to 1.1

    A change of the 'Edition' field implies compatibility with previous editions.

    This option may also be used to add new ORI vendor specific bytes definitions.

  - **Code 201 : ORI Vendor. Length = 3.**
    - Byte 1..3: ASCII encoded Vendor Code

The list of ORI vendor codes is listed in annex B, and provided at:

http://portal.etsi.org/ORI/ORIvendor_codes.pdf

Code 202 : RE TCP port. Length = 2.

Shall be used to request the REC to connect to this TCP port.

EXAMPLE:

Code	Len	Code	Value	Code	Value	Code	Value
43	13	200	2	1	1	201	3	O	R	I	202	2	1300

All other Vendor Specific records shall be ignored by the REC.

10.1.1.2 TCP options

The following TCP socket options shall be used by the REC and RE:

- SO_KEEPALIVE: OFF
  - This option is used to detect the dead connection. After the expiration of a timer, TCP try to transmit a zero length TCP segment (with a bad sequence number) to the peer socket. This type of segment shall force the distant to reply. If waited response is returned TCP knows that the connection is always active, it does not need to test it again before the end of the timer. If no response from the peer, TCP try to resend this segment several times before to stop and to declare this connection broke down.
  - The issue with this option is that the timer used to send these messages is usually global for all TCP applications in the system, and is not easily manageable. Therefore this option is not required, and the OCP Health Check procedure is used to provide an equivalent service.
- SO_LINGER: OFF
  - This option has an impact on the close process. The goal is to implement a "graceful" disconnection without loss of packets. With this option the TCP socket tries to send packets stored in the emission queue before starting the close process.
  - As the socket is basically closed due to an error, it is not necessary to wait for transmission of remaining data, as the remote end is unable to read them.
- TCP_NODELAY: ON
  - This option allows delivering messages immediately. This option is to be used for real-time protocols that require delivery of many small messages. The mechanism provides by the TCP_NODELAY option bypasses the congestion-avoidance algorithm (Naggle algorithm). With this option any frame is sent if there is the room in the send window of a socket.

10.1.2 RE startup procedure / Alignment

Whenever REC and RE get (re-)connected, the steps described below shall be executed in order to align REC and RE before any other C&M activity takes place.

NOTE 1: No specific measures are taken in the RE to prevent premature C&M activity.

The RE shall not take any additional recovery action upon failure of one or more of the steps described below.

Precondition: successful establishment of the OCP layer (clause 10.1.1).

Steps:

1) Time alignment:
   Execute the SET TIME function [clause 8.1.2]

2) REC - RE Software alignment:
   Execute the REC<->RE Software Alignment procedure [clause 10.1.3.3]

3) Resource Model alignment:
   With this step, the REC gets aligned on

   - the resource model objects instances existing on the RE
   - the parameter values of these objects
   - the objects' states where applicable
   - any active fault reports

NOTE 2: Based upon the ORI line bit rates supported, as reported as part of the REC<->RE resource model object parameter value alignment, the REC may trigger the ORI link to move back to CPRI state B (to allow it to negotiate the use of desired ORI line bit rate).

NOTE 3: The delay calibration capabilities and initial configuration are reported as part of the REC<->RE resource model object parameter value alignment described here. Timing calibration procedures are further defined in clause 10.2.4.

Sequence of activities:

a) Object instance and parameter reporting:
   Execute OBJECT PARAMETER REPORTING (Object ID = "ALL", Parameter = "ALL")
   [clause 8.3.1]

b) Object state reporting:
   Execute OBJECT STATE REPORTING (Object ID = "ALL", State Type = "ALL")
   [clause 8.5.1]
   Optionally, event driven state reporting can be activated in this step
   OBJECT STATE REPORTING (Object ID = "ALL", State Type = "ALL",
   Event-Driven Reporting = "Enabled")

c) Active fault report alignment:
   FAULT REPORTING (Object ID = "ALL") [clause 8.6.1]
   Optionally, event driven fault reporting can be activated in this step
   FAULT REPORTING (Object ID = "ALL", Event-Driven Reporting = "Enabled")

After successful execution of the steps described above RE and REC are aligned. The RE shall be prepared that the REC may repeat these steps in whole or part subsequently at any time.

Subsequent configuration steps are driven by actual RE resource availability and may include AISG port configuration, and Signal Path configuration. The exact order of the individual configuration steps is not specified.

10.1.3 Software management

10.1.3.1 RE restart triggers

A restart of the RE is triggered by the ORI Low Layer Reset defined in [3], the RE RESET procedure, power-up or any other reset. For an REC initiated restart, it is recommended to use the RE RESET procedure.

10.1.3.2 Restart actions

When the RE restart is triggered, following the restart the RE shall operate using the Active software image.

Abnormal and failure case:

The RE detects that the Active Software Image is not working correctly for whatever reason. It is out of the scope for the present document to specify all possible reasons for this, but one reason shall be that 3 subsequent TCP connection establishment timer (T_{TCE}) expirations have occurred.
Action: The RE shall set the *Passive Software image* to be the *Active Software Image* and vice versa and trigger a restart of itself.

10.1.3.3 REC<->RE Software Alignment

1) REC performs the Version Query procedure (see step 1 in figure 10.1.3.3-1)

Normal flow of events:

The REC detects that the reported software upgrade package version identifier of the active software image is the expected one.

End condition:

Software is correctly aligned between REC and RE.

Alternative flow of events:

a) The REC detects that the reported software upgrade package version identifier of the active software image differs from what the REC expected, but the software upgrade package version identifier of the passive software image is the expected one. The REC may perform a Software Activation procedure.

or

b) The REC detects that the reported software upgrade package version identifiers of the respective active software image and passive software image differs from what the REC expected. Further actions taken by the REC is unspecified but may involve a RE Software Upgrade, see step 2.

or

c) Any other scenario occurs which is not covered by any of the previous cases in bullet 1. The action taken by the REC is unspecified. It may involve REC-RE Vendor ID compatibility rule checking and specific fault handling. {This may be the case for the initial (first) start-up of the REC- RE pair}.

2) If the REC decides that a Software Upgrade is required, the following shall apply:

Normal flow of events for successful software upgrade:

i) The REC performs the Software Update Preparation Procedure (see step 2 in figure 10.1.3.3-1) using the information reported in the Version Query Report to select the FTP Software Upgrade Package Directory Path. How the REC knows the Software Upgrade Package Version identifier is unspecified, but this is expected to be provided by the REC vendor.

ii) The RE performs the Software Download Procedure (see step 3 in figure 10.1.3.3-1).

iii) The REC Performs the Software Activation procedure (see step 4 and 5 in figure 10.1.3.3-1).

iv) The REC<->RE pair starts up according to clauses 10.1.1. and 10.1.2.

Alternative and failure flow of events:

The action taken by the REC is out of the scope of the present document.

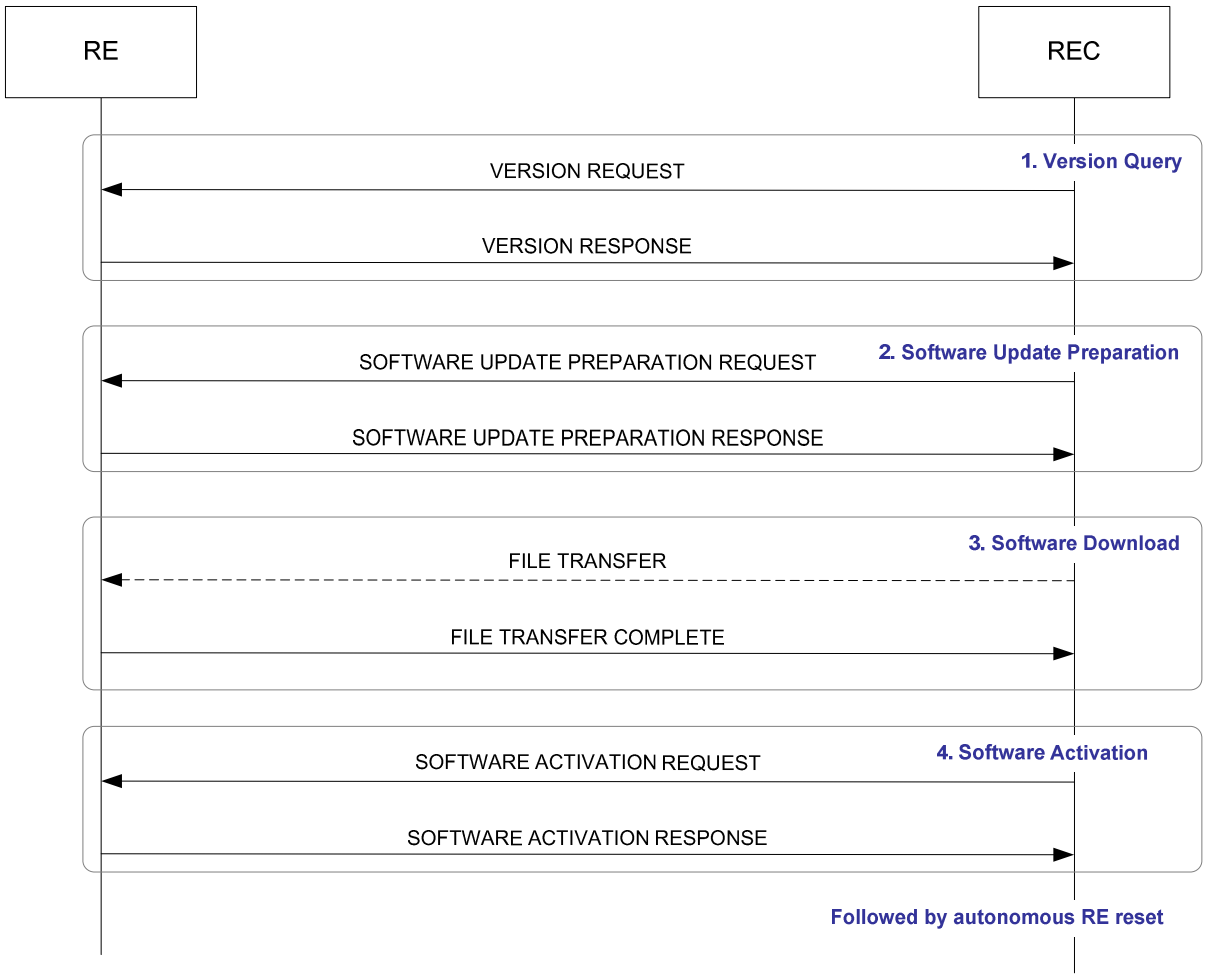

Figure 10.1.3.3-1: RE Software Alignment

10.2 RE operation

10.2.1 Dynamic object initial status alignment

Figure 10.2.1-1 describes the procedures applicable for initial alignment between REC and RE of the status of a dynamic object, following indication from RE to REC that the object has been successfully created.

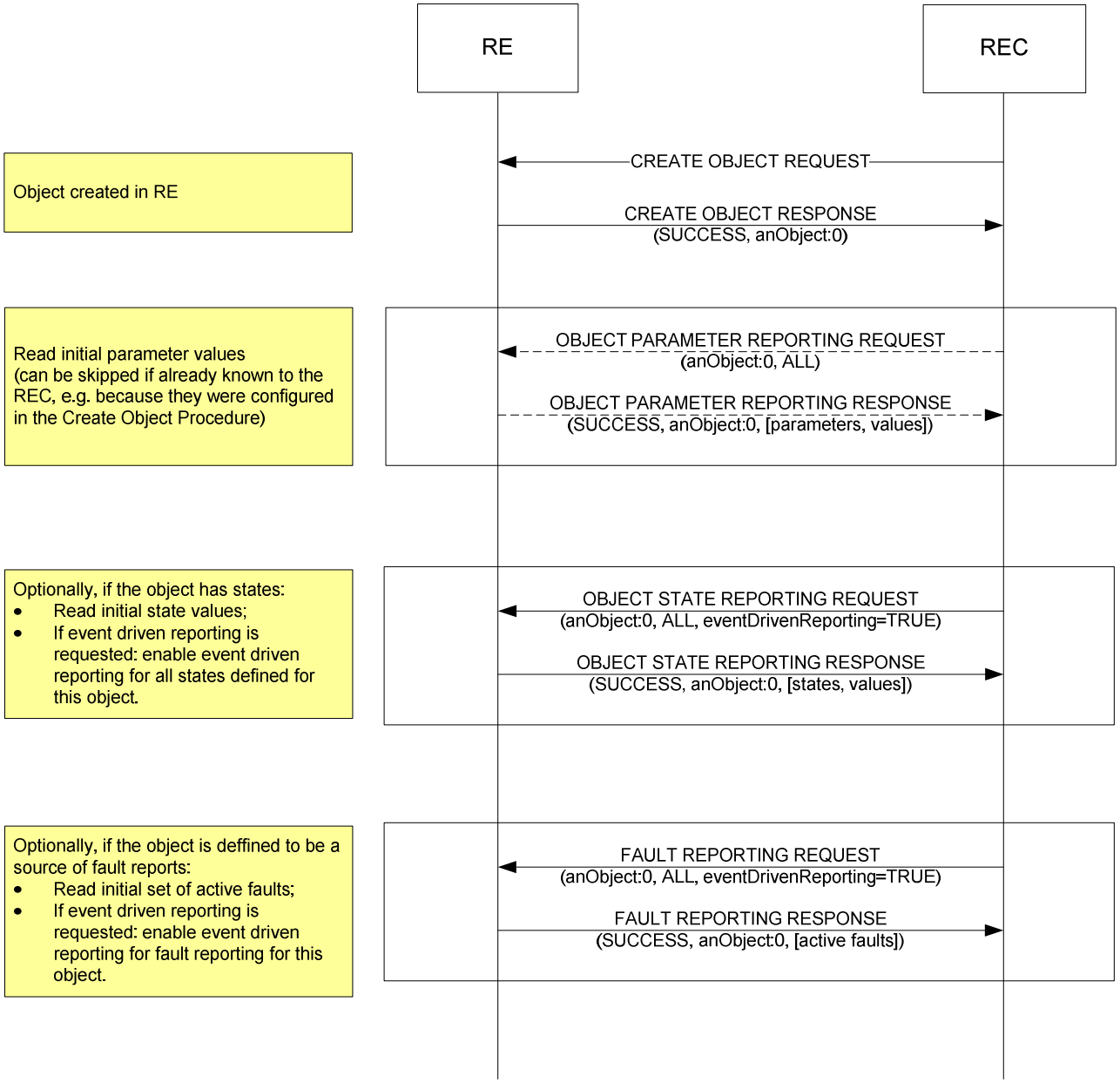

Figure 10.2.1-1: Dynamic object initial status alignment

10.2.2 ORI link configuration

10.2.2.1 General

In order for ORI links to change from CPRI State A to State F (for active links) or State G (for passive links), the REC/RE shall follow clause 8 of [3]. Once CPRI State F/G has been reached, for the corresponding ORI Link object the FST shall change to "operational".

For passive links, if the given ORI Link object in the RE has been locked by the REC, then it shall be unlocked before the ORI link is able to leave CPRI State A.

10.2.2.2 Transceiver module inserted during RE operation

The following sequence of events shall only be possible once the OCP layer has been established on a different ORI Port.

If a physical transmission link/transceiver module is inserted into an ORI Port on the RE during RE operation, then the following applies (also shown in figure 10.2.2.2-1):

1) When the physical transmission link/transceiver module gets inserted, the FST of the corresponding ORI Link object changes from "disabled" to "not operational".

2) The general procedure applies, as described in clause 10.2.2.1.

3) In order to ensure that the line bit rates previously indicated as supported by the RE for the ORI Link object is still valid after insertion of a transceiver module, the REC may optionally re-acquire the value of the "Supported Line Bit Rates" parameter from the RE at any time.

When the physical transmission link/transceiver module is removed from the ORI Port, the FST of the corresponding ORI Link object changes to "disabled".

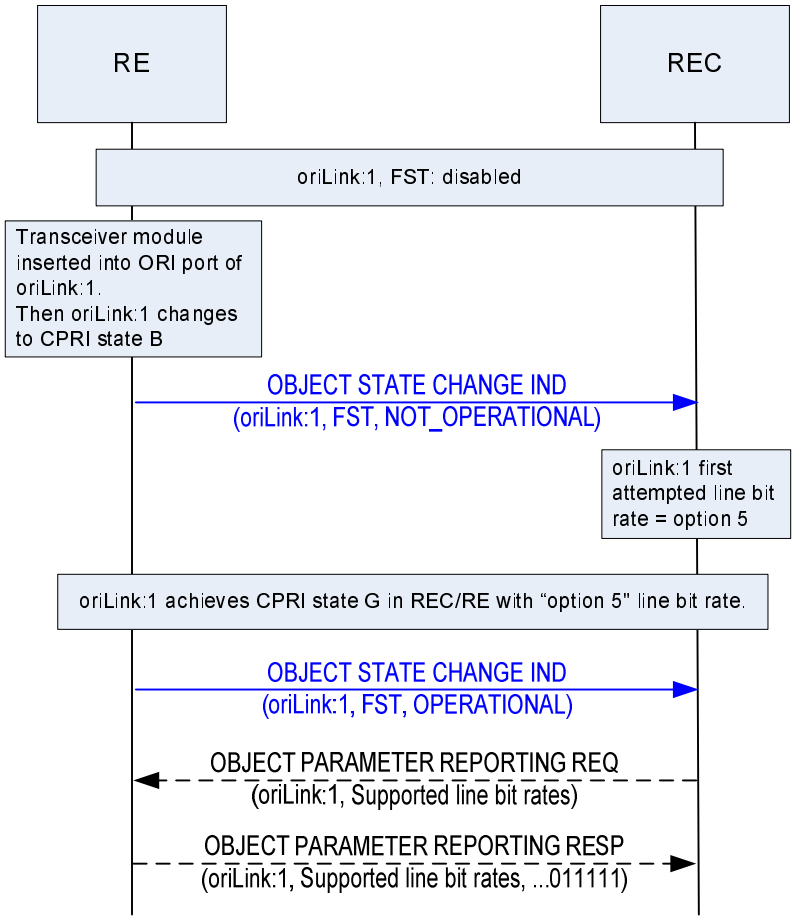

Figure 10.2.2.2-1: Bringing ORI (passive) Link into operation (physical transmission link/transceiver module inserted during RE operation)

10.2.3 ORI link maintenance

No specific C&M plane functionality is specified for this in the present document.

10.2.4 Delay calibration

Time components and their relation for delay calibration for the single-hop case are shown in figures 10.2.4-1 and 10.2.4-2.

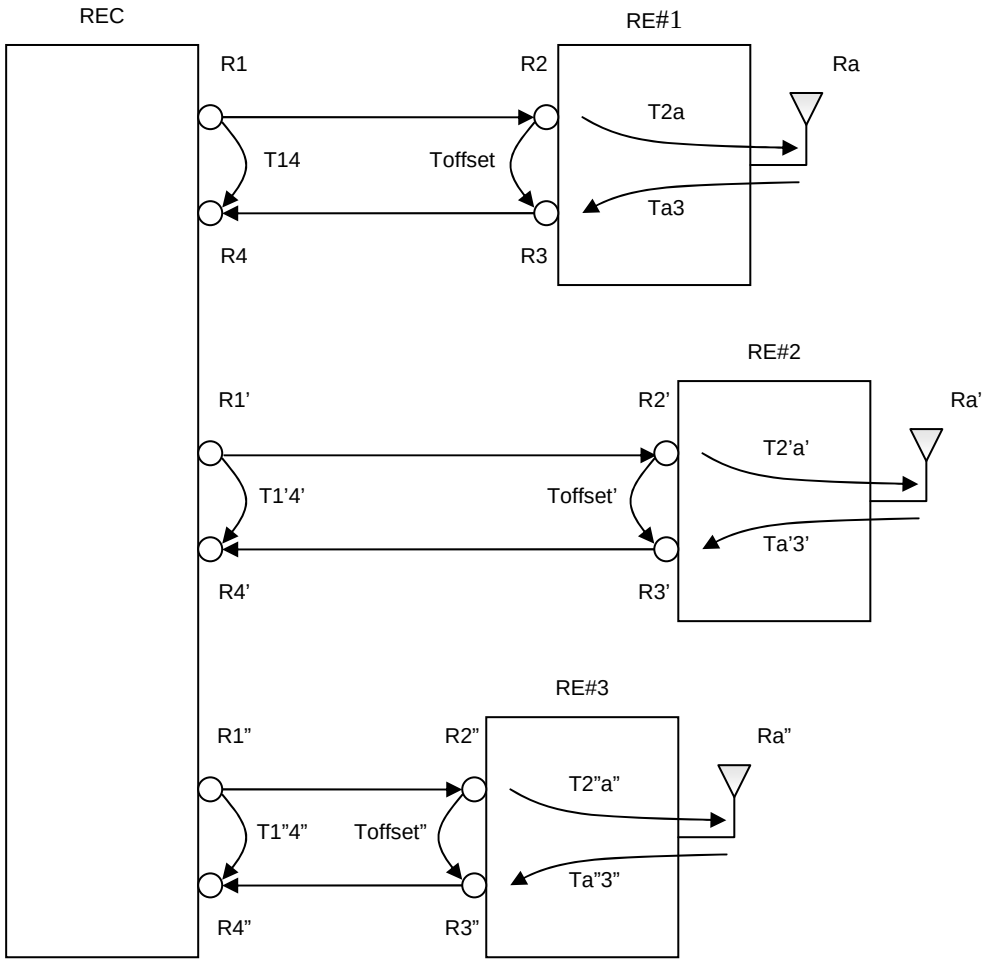

Figure 10.2.4-1: Time components for single hop

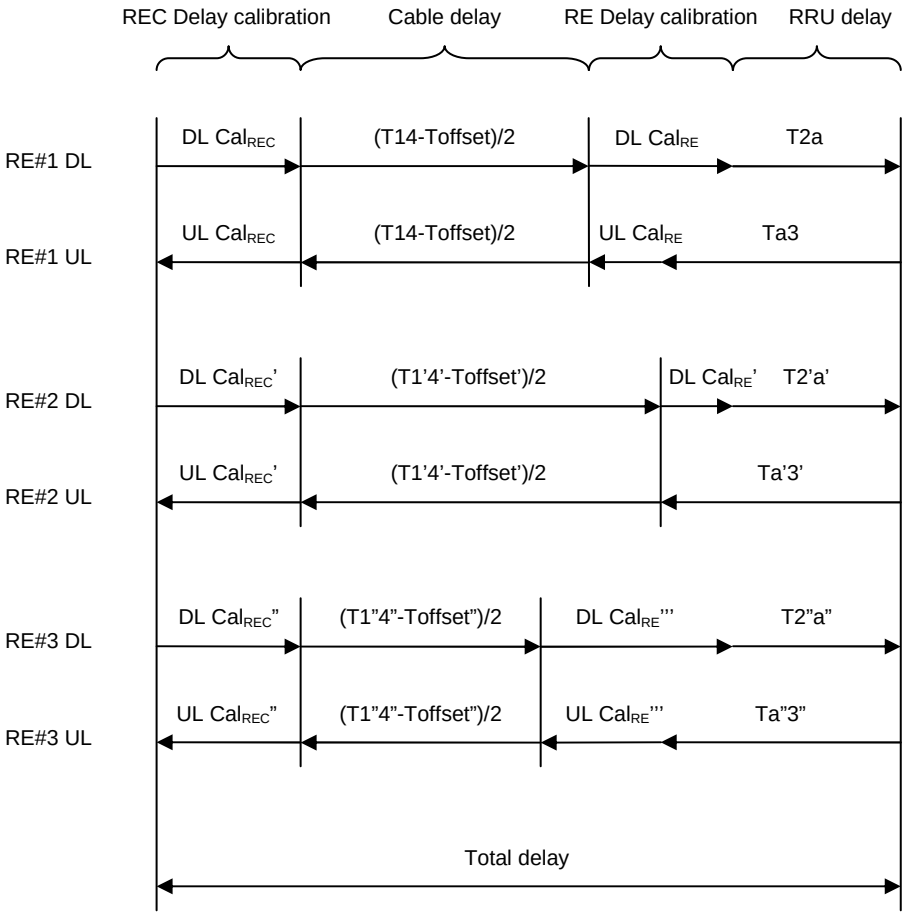

Figure 10.2.4-2: Relation of time components (including delay calibration) for single hop

The 'total delay' is a configuration parameter given to the REC to allow it to synchronize the time delay of different REs, while 'T14' is the delay (frame timing difference) measured by the REC.

'Toffset', 'T2a', 'Ta3', 'DL Cal_{RE}_MAX' and 'UL Cal_{RE}_MAX' are reported from the RE to the REC. The value of 'Toffset' shall be fixed per ORI link. The values of 'T2a', 'Ta3', 'DL Cal_{RE}_MAX' and 'UL Cal_{RE}_MAX' are defined per Tx and Rx Signal Path respectively and shall not change while the Signal Paths are operational/unlocked.

Immediately prior to unlocking the Signal Paths, reporting of the values 'T2a' and 'Ta3' shall be requested by the REC. The REC shall then calculate the required DL and UL buffering values in the RE and, while the Signal Paths are locked, it shall configure the corresponding delay calibration parameters in the RE, with a granularity of $T_C/16$ in the DL and $T_C/2$ in the UL.

If the maximum buffer capability in the RE ('DL Cal_{RE}_MAX' and/or 'UL Cal_{RE}_MAX') is not sufficient to meet the delay calibration requirement (i.e. the 'total delay'), the REC shall provide the additional delay calibration internally, applying DL Cal_{REC} and/or UL Cal_{REC}.

The minimum value for DL Cal_{RE}_MAX that the RE is required to support is 16 (i.e. 1*Tc) while the minimum value for UL Cal_{RE}_MAX is 2 (i.e.1*Tc).

The timing acquisition and the delay calibration configuration procedure for single hop are shown in figure 10.2.4-3.

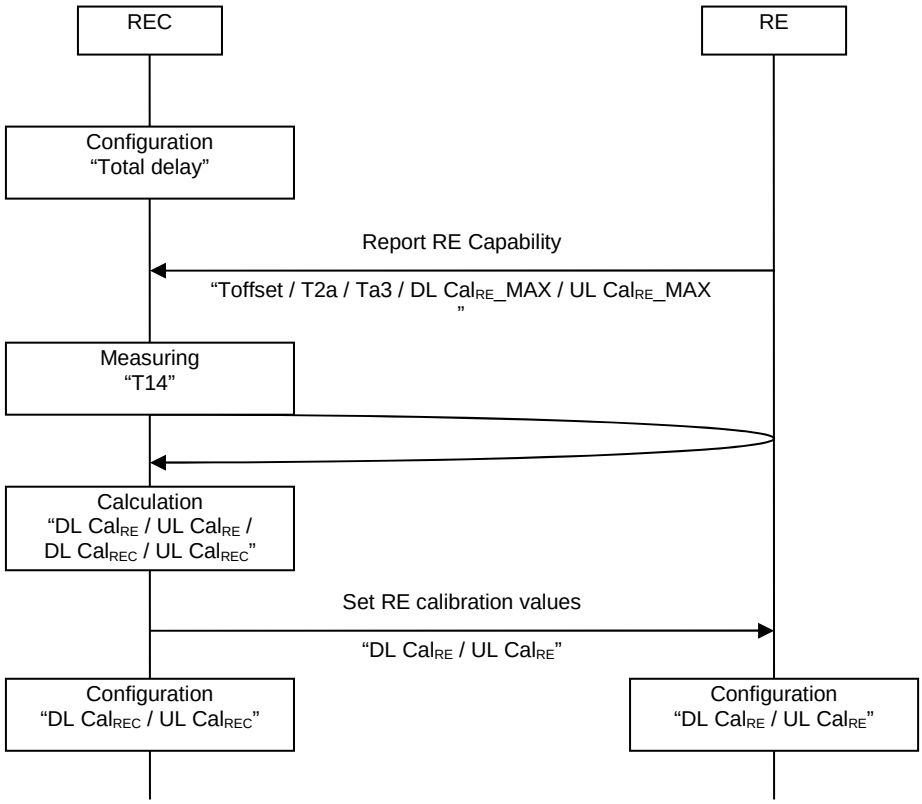

Figure 10.2.4-3: Timing acquisition and delay calibration configuration procedure for single hop

Table 10.2.4-1: REC Delay calibration parameter

Category	Items	Remarks
REC measuring	T14	Measured by REC
System configuration	Total delay	Given to REC for synchronizing the time delay of all REs
Delay calibration	DL Cal_{RE}	DL buffering size in the RE according to the following calculated value [configured by REC and sent to RE] DL Cal_{RE} = MAX(Total delay - (T14-Toffset)/2 - T2a, DL Cal_{RE}_MAX) [T_C/16]
	UL Cal_{RE}	UL buffering size in the RE according to the following calculated value [configured by REC and sent to RE] UL Cal_{RE} = MAX(Total delay - (T14-Toffset)/2 - Ta3, UL Cal_{RE}_MAX) [T_C/2]
	DL Cal_{REC}	DL buffering size in the REC according to the following calculated value DL Cal_{REC} = Total delay - (T14-Toffset)/2 - T2a - DL Cal_{RE}
	UL Cal_{REC}	UL buffering size in the REC according to the following calculated value UL Cal_{REC} = Total delay - (T14-Toffset)/2 - Ta3 - UL Cal_{RE}

Table 10.2.4-2: RE Delay calibration parameter

Category	Items	Remarks
RE capability report	Toffset	Reported from RE to REC "Reported Delay" [fixed value per ORI link]
	T2a	Reported from RE to REC "Reported Delay" [fixed value per TxSigPath]
	Ta3	Reported from RE to REC "Reported Delay" [fixed value per RxSigPath]
	DL Cal_{RE}_MAX	Max possible buffer size in RE for the DL calibration (unit=T_C/16). "Reported Capability" [fixed value per TxSigPath]
	UL Cal_{RE}_MAX	Max possible buffer size in RE for the UL calibration (unit= T_C/2). "Reported Capability" [fixed value per RxSigPath]
Calibration values set by the REC	DL Cal_{RE}	The fraction of the DL calibration in the RE, configured by REC and sent to RE. "Configured Delay" [configured by REC via OCP to perform fine calibration per TxSigPath] DL Cal_{RE} = MAX(Total delay - (T14-Toffset)/2 - T2a, DL Cal_{RE}_MAX) [T_C/16]
	UL Cal_{RE}	The fraction of the UL calibration in the RE, configured by REC and sent to RE. "Configured Delay" [configured by REC via OCP to perform fine calibration per RxSigPath] UL Cal_{RE} = MAX(Total delay - (T14-Toffset)/2 - Ta3, UL Cal_{RE}_MAX) [T_C/2]

Delay calibration for multi-hop is out of scope of the present document.

10.2.5 Signal Path control (Cell configuration)

This clause describes the procedures how the Signal Paths of a sector are set up, modified, deleted, and switched off/on. The procedures use a Tx Signal Path as example, but the same procedure also applies to Rx Signal Paths.

Note that a sector (cell) consists at least of a Tx and a Rx signal path, each of which processing a single AxC. For diversity or MIMO setups, a sector (cell) may consist of multiple of these signal paths. So a full cell setup consists of two or more of the procedures shown below.

10.2.5.1 Signal Path setup

The following steps are needed to set up a Tx Signal Path:

1) Create a Tx Signal Path object of the requested radio standard, initialize the parameters of the object with the desired configuration, and optionally configure event-driven reporting of state changes and faults.

2) Unlock the object to take it into service.

Steps 1) and 2) do not necessarily need to be executed in close relation; it might even be helpful for the RE's resource management to defer step 2) until all required Signal Path objects are configured.

Steps to set up a signal path:

Scenario A: successful setup

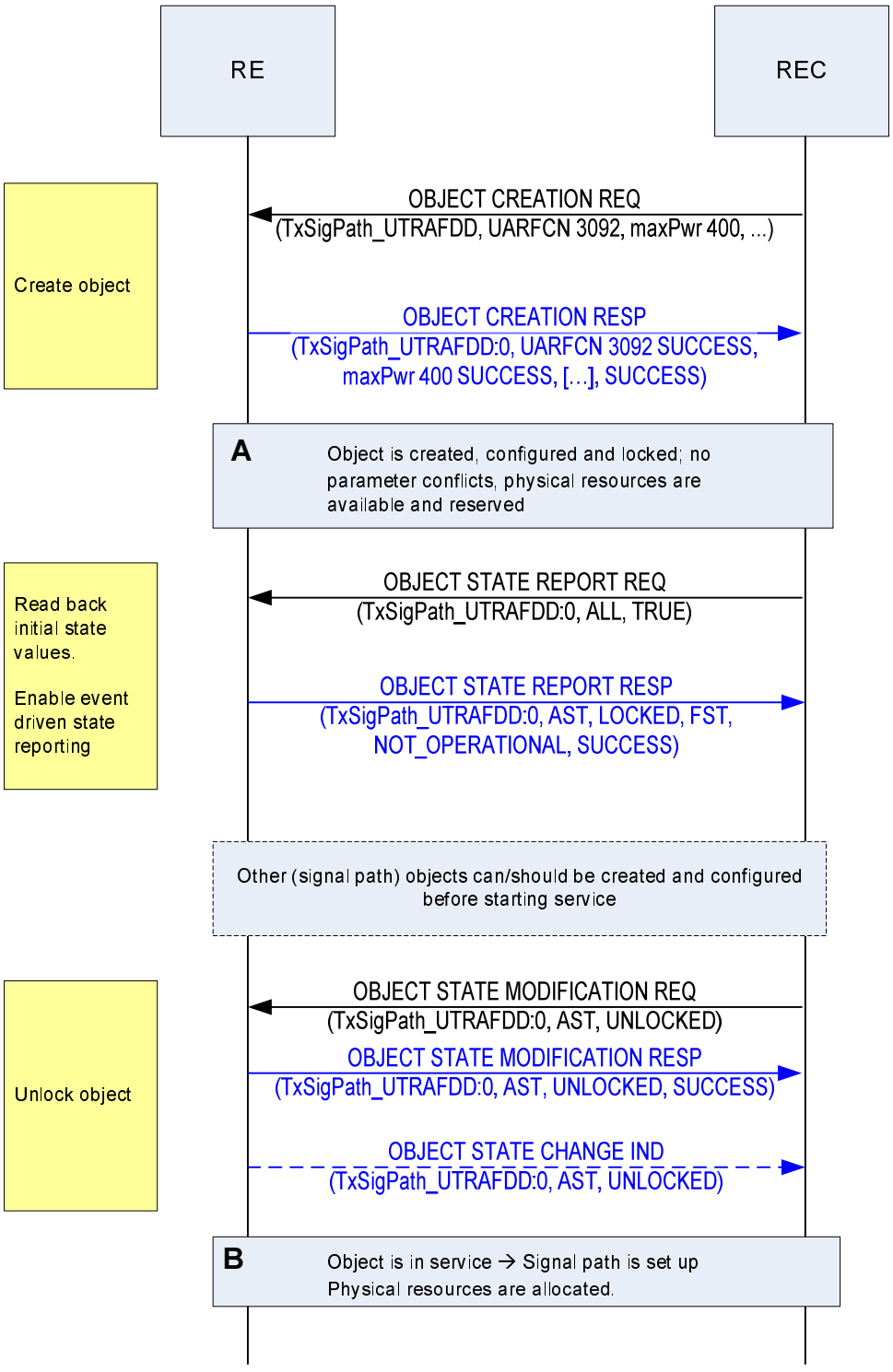

Figure 10.2.5.1-1: Sequence successful signal path setup

Scenario B: failure because of resource conflict

A resource conflict means that the parameters provided for the cell/sector are correct, but does not fit the possible carrier combinations of a given RE.

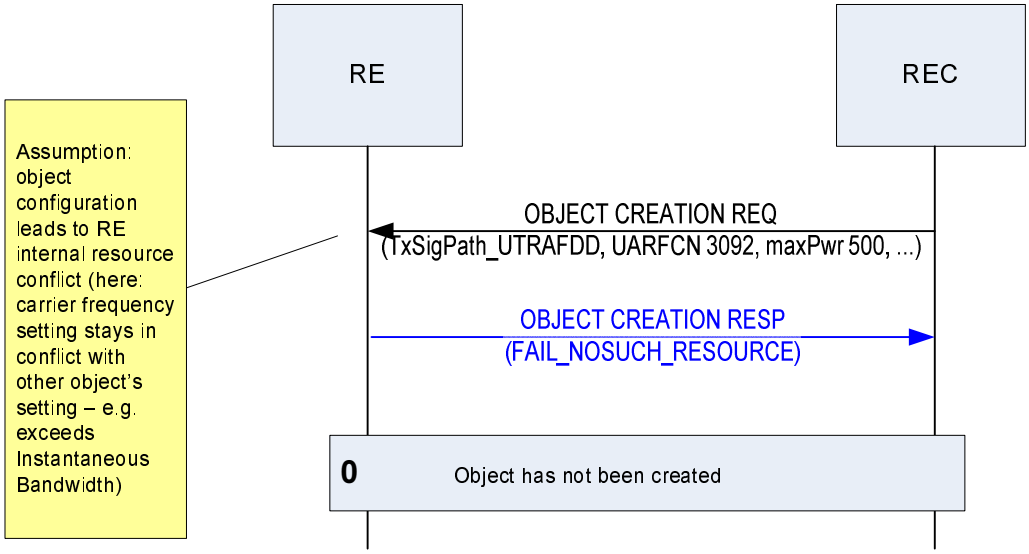

Figure 10.2.5.1-2: Sequence signal path setup failure

10.2.5.2 Modify a Signal Path parameter when locking required

The following steps are needed to modify such parameters of a Tx Signal Path which require the Signal Path to be taken out of service:

1) Lock the object to take it out of service for re-configuration.

2) Re-configure the parameters of the object with the desired new values.

3) Unlock the object to allocate physical resources and take it into service.

Scenario A: successful re-configuration

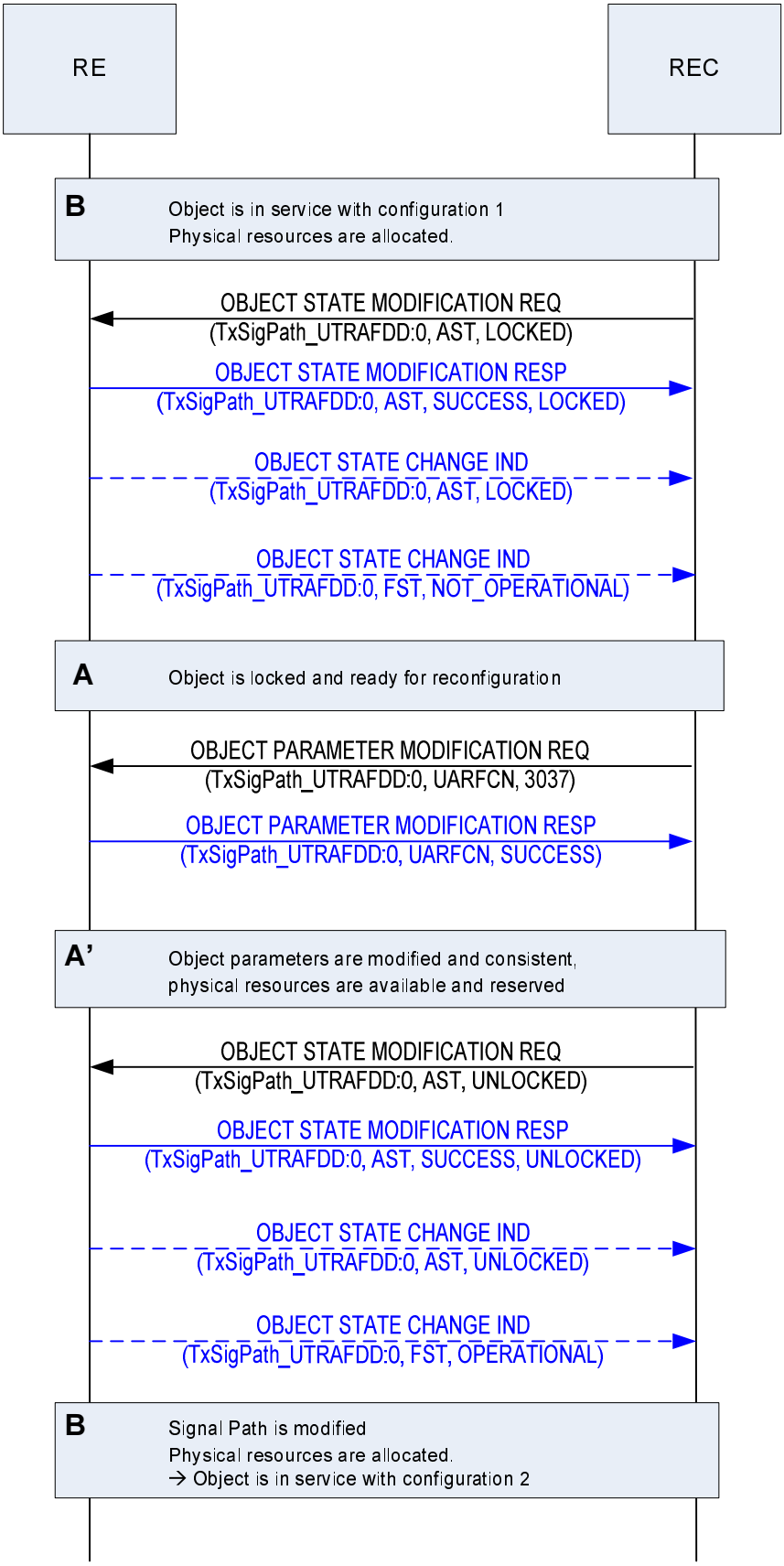

Figure 10.2.5.2-1: Sequence successful signal path modification

Scenario B: reconfiguration not successful

Per the definition of a non-successful reconfiguration, the object remains entirely unchanged. Taking it back into service after such an attempt takes its original configuration back in use.

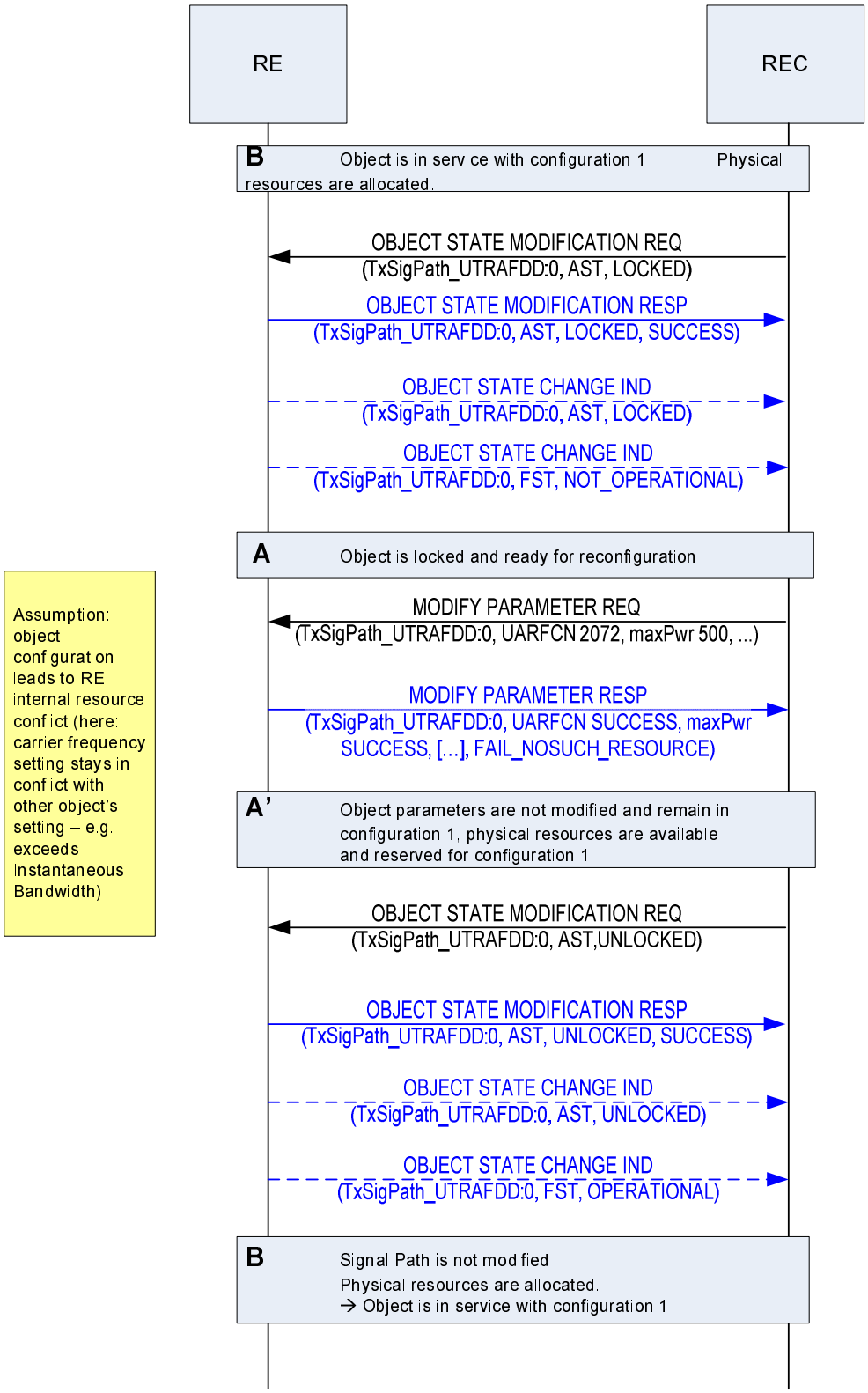

Figure 10.2.5.2-2: Sequence failed signal path modification

10.2.5.3 Modify a Signal Path parameter when locking not required

The following steps are needed to modify such parameters of a Tx Signal Path which do not require the Signal Path to be taken out of service:

1) Re-configure the parameters of the object with the desired new values

Scenario A: successful re-configuration

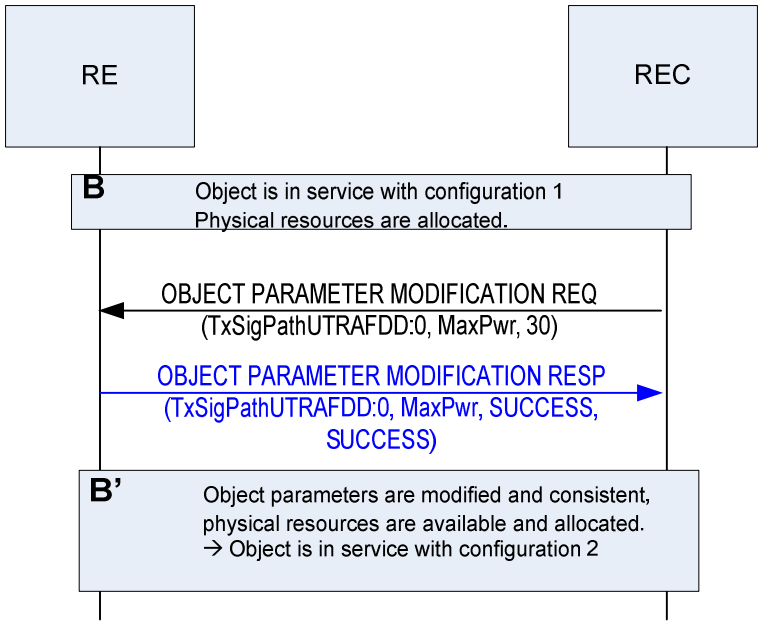

Figure 10.2.5.3-1: Sequence successful signal path modification

Scenario B: reconfiguration not successful

Per the definition of a non-successful reconfiguration, the object remains entirely unchanged.

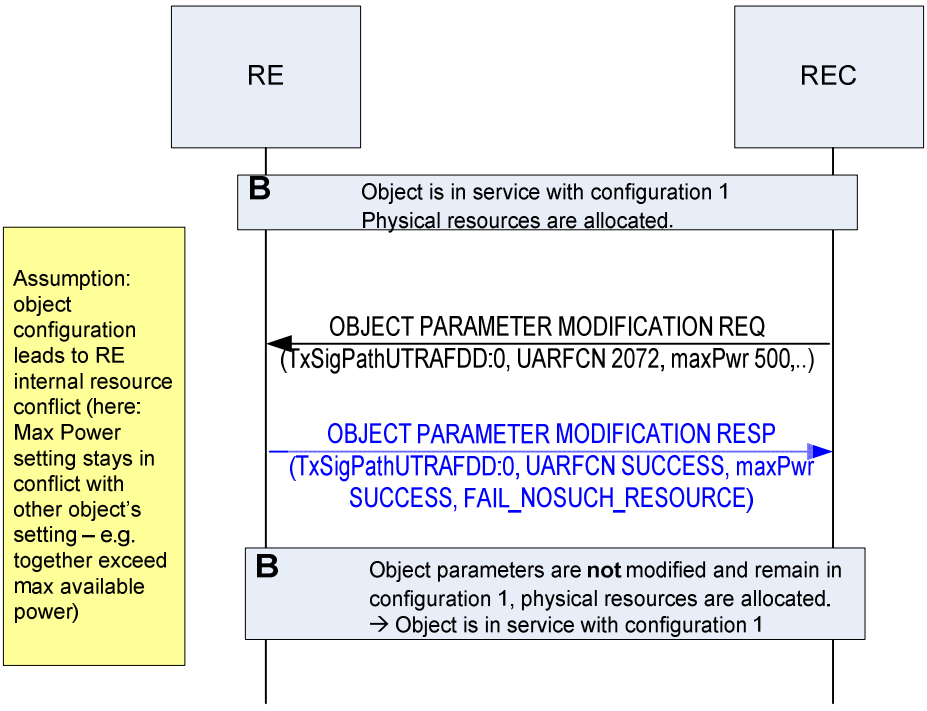

Figure 10.2.5.3-2: Sequence failed signal path modification

10.2.5.4 Delete a Signal Path

The following steps are needed to delete a Tx Signal Path:

1) Lock the object to take it out of service

NOTE: Other related objects, like associated Rx objects can be locked as well before executing step 2).

2) Delete the object

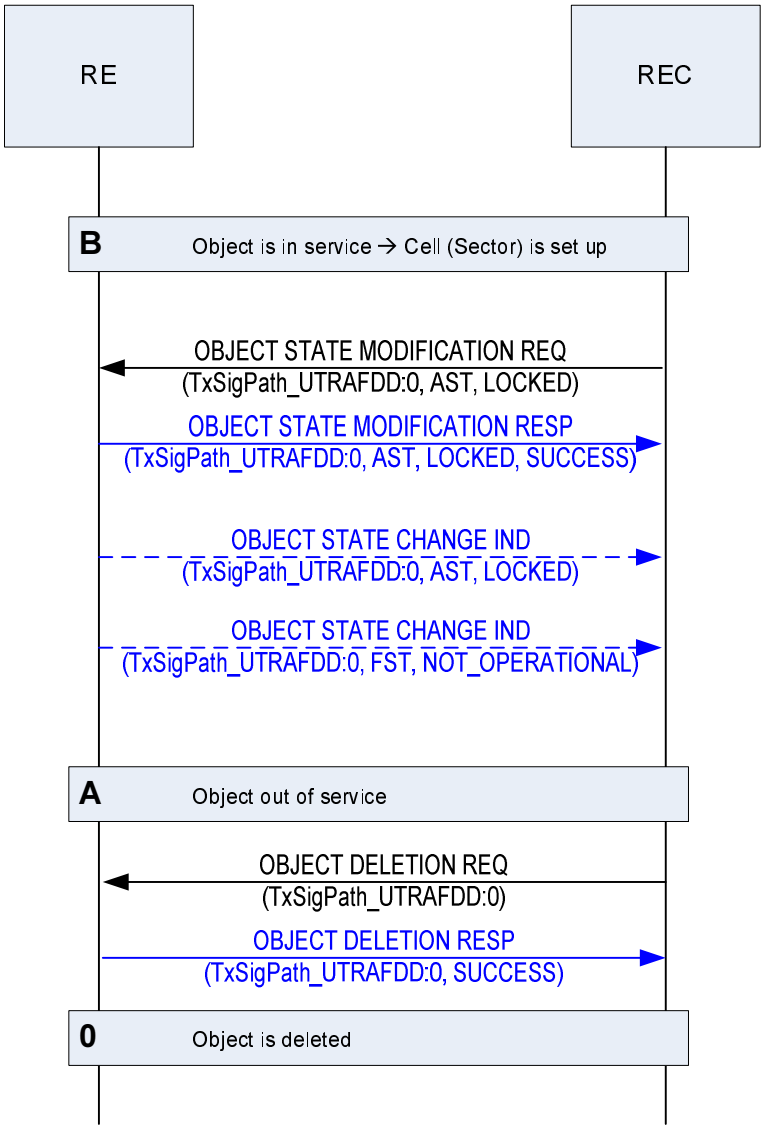

Figure 10.2.5.4-1: Sequence signal path deletion

10.2.5.5 Switch a Signal Path off and on

The following steps are needed to switch a signal path's RF processing off and on:

1) Lock the object to take it out of service (remains configured, and physical resources remain reserved).

2) Unlock the object to take it back into service.

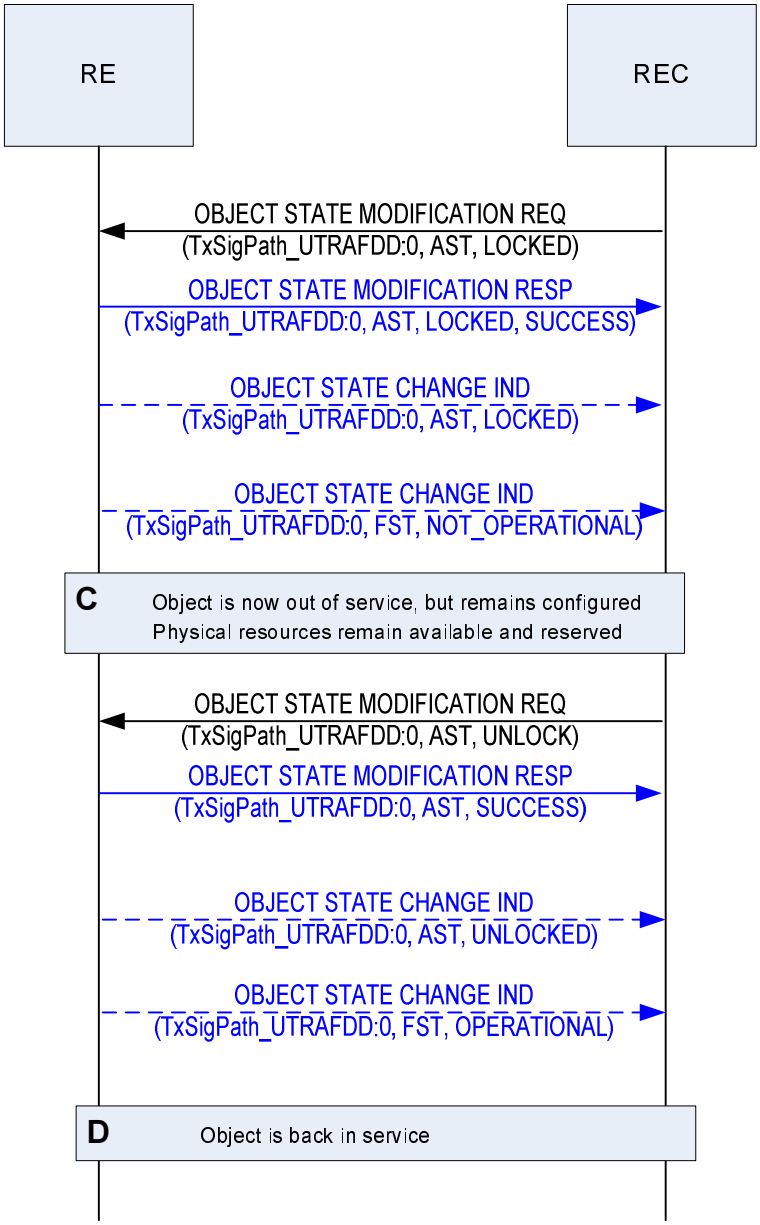

Figure 10.2.5.5-1: Sequence signal path on / off

10.2.6 AISG/3GPP Iuant management

10.2.6.1 AISG/3GPP Iuant Layer 2 establishment

The control and management of AISG ports and their connected ALDs is performed via the AISG port object(AISGPort, see clause 9.1.7) and the associated ALD (AISGALD, see clause 9.1.8) child objects.

Connected ALDs and their assigned HDLC addresses are known by the REC via one of 2 approaches:

1) Device Scan approach.

2) By the REC being pre-configured with information available about the installed, or planned to be subsequently installed, ALD devices, and configuring this in the RE by creating an AISGALD object.

After the HDLC address assignment, the RE may determine the maximum frame length for message transmission (according to clause 4.8.1 in [10]). A message length of minimum 74 octets shall be supported.

10.2.6.1.1 Layer 2 connection establishment using Device Scan

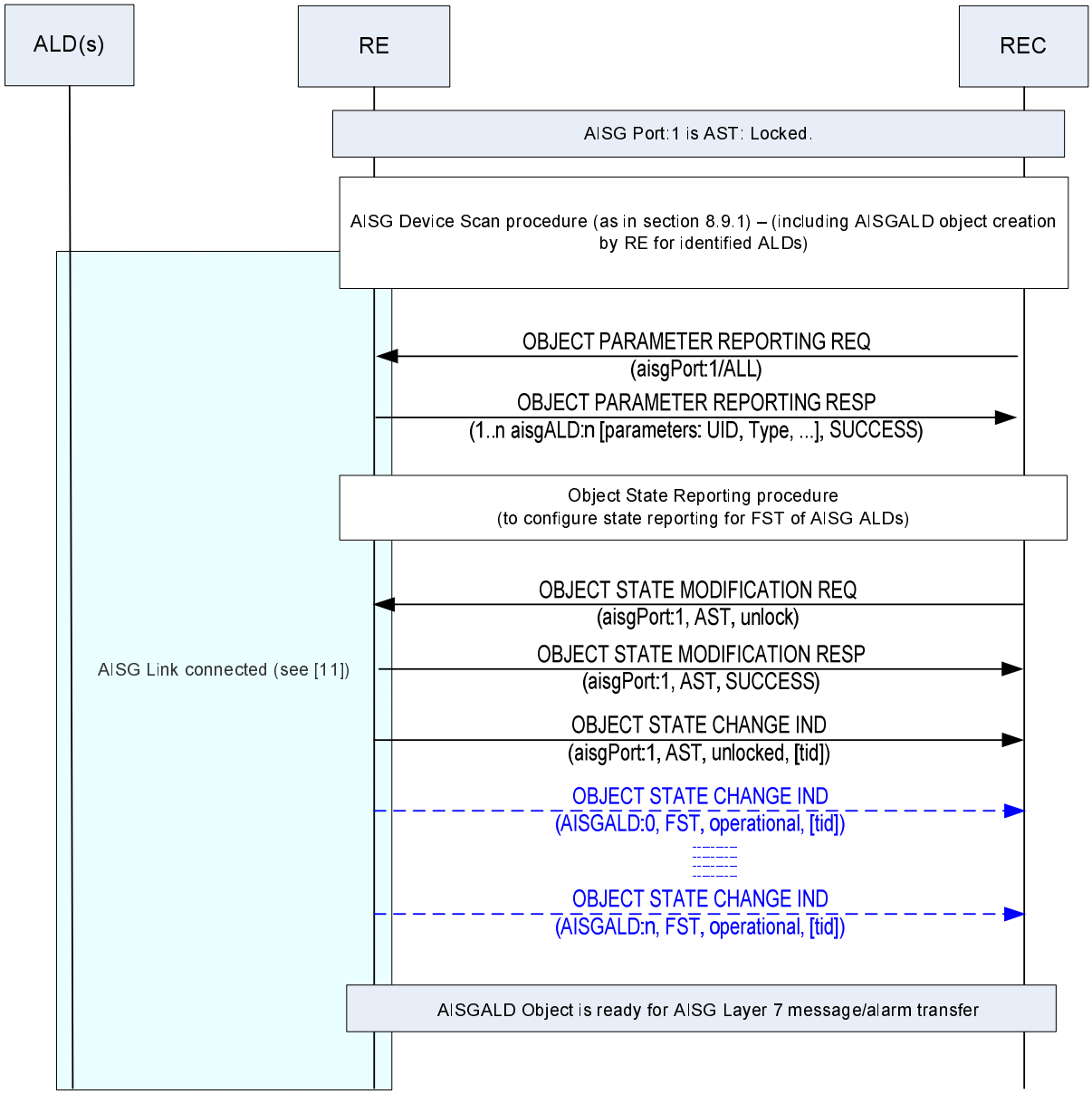

Figure 10.2.6.1.1-1: Layer 2 connection establishment using Device Scan procedure

1) The REC shall AST Lock the AISGPort in the RE if the AISGPort is unlocked.

2) The REC triggers the AISG Device Scan procedure towards the RE. During this procedure, the RE creates as many AISGALD objects as detected, and set the parameters in each AISGALD.

3) Once the AISG Device Scan procedure is indicated as being completed to the REC, the REC reads the AISGALD objects from an AISG Port object by using the OBJECT PARAMETER REPORTING procedure.

4) The REC may configure event-triggered reporting on FST states of the AISGALD objects, as necessary.

5) The REC may then trigger the RE to AST Unlock the AISGPort object. Once this is done, the AISG Bus is ready to be used.

6) The FST of AISGALD objects change to FST Operational, which means that the corresponding ALD is ready for the transfer of Layer 7 messages and alarms. If and only if event-triggered reporting of FST state changes has been configured by the REC, the RE indicates the change of FST state autonomously to the REC.

10.2.6.1.2 Layer 2 connection establishment using AISGALD object creation requested by the REC

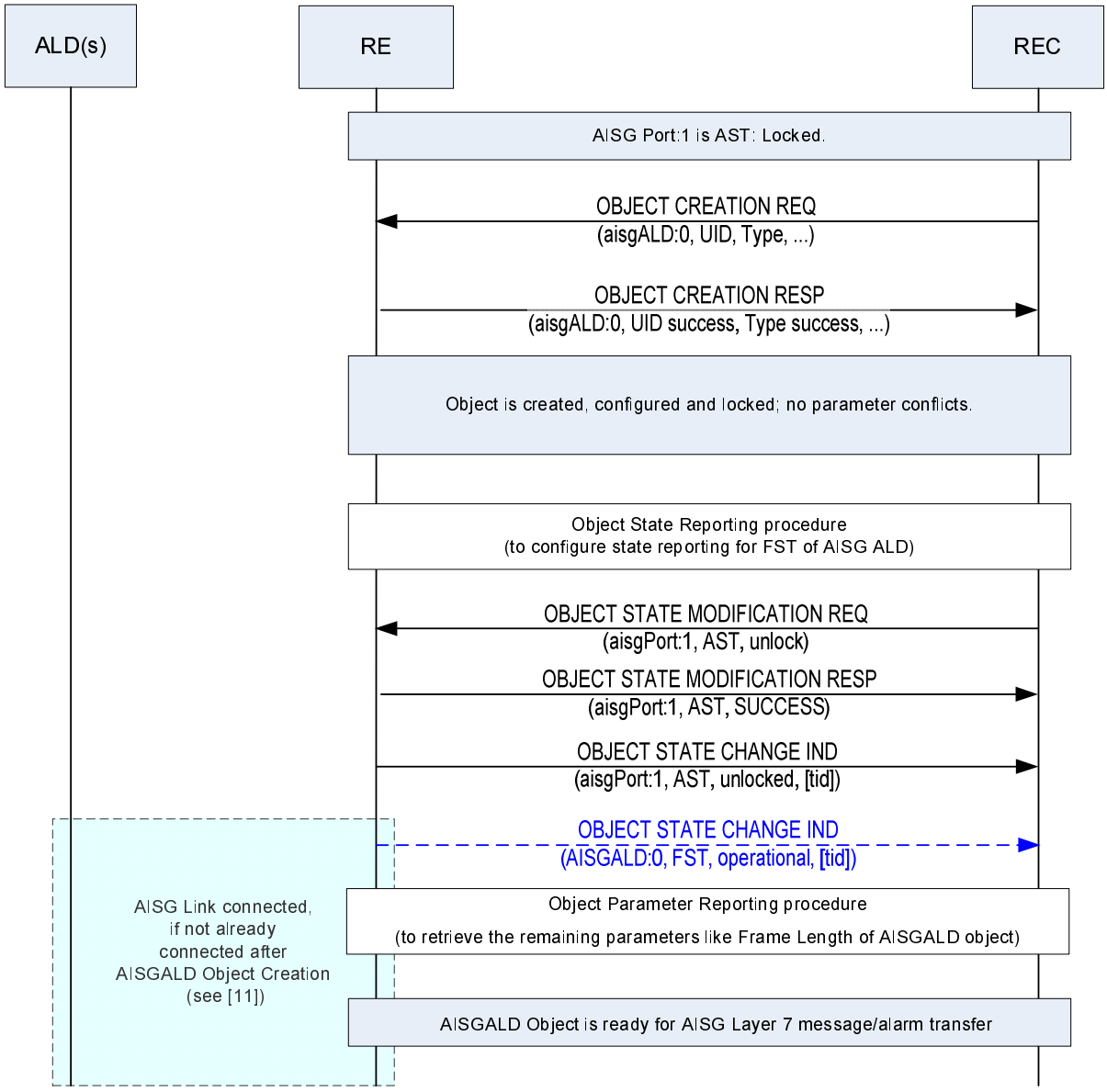

Figure 10.2.6.1.2-1: Layer 2 connection establishment using AISGALD object creation by REC

1) The REC shall AST Lock the AISGPort in the RE if the AISGPort is unlocked.

2) The REC triggers the creation of ALD objects, using the Object Creation procedure described in clause 8.4.1. The REC creates as many AISGALD objects as required, and creates each AISGALD object at the AISGPort where it is installed, or planned to be subsequently installed. The REC shall set a valid "Unique ID" and/or "device type" within the created AISGALD object, to allow the RE to assign an HDLC address to this ALD using the AISG Address Assignment procedure (clause 4.8.3 in [10]). The given information shall be unambiguous so that only a single ALD is addressed, as otherwise the RE would be unable to assign an address, and the AISGALD object remains in state disabled.

3) Once the AISGALD objects have been created, the REC may configure event-triggered reporting on FST states of the AISGALD objects, as necessary.

4) The REC may then trigger the RE to AST Unlock the AISGPort object. Once this is done, the AISG Bus is ready to be used.

5) The FST of AISGALD objects change to FST Operational, which means that the corresponding ALD is ready for the transfer of Layer 7 messages and alarms. If and only if event-triggered reporting of FST state changes has been configured by the REC, the RE indicates the change of FST state autonomously to the REC.

If not done already, the remaining parameters of each newly created AISGALD object shall be retrieved using the Object Parameter Reporting procedure, such that the REC is ready to transfer Layer 7 messages towards the ALD.

Annex A (informative): Example for a vendor specific parameter in the RE resource model

Vendor "XYZ Networks" (Vendor code "xyz") provides an additional LED at each of their antenna ports to allow antenna port identification by highlighting the associated LED. This LED can be controlled through an additional parameter "LEDenabled" in object type "Physical Antenna Port" (clause 9.1.2). The RE would then report and allow usage of the extended Physical Antenna Port Object as if its definition was as shown in the example below (vendor specific parameter highlighted in bold).

Table A-1: Example: Physical Antenna Port Object Parameters with vendor specific parameter

Parameter	Description	Type	Range
Physical Antenna Port Label	This is the name provided to the Physical Antenna Port by the RE vendor. (see note)	Text String	
XYZ::LEDenabled	Setting this parameter to "TRUE" enables (lights up) the LED attached to the physical antenna port represented by the related instance of this object. This is to support port identification for service and installation. Management type: RW	Boolean	TRUE, FALSE Default: FALSE
NOTE: It is recommended that, for future ORI compliant products, the Physical Antenna Port Label is the Physical Antenna Port object name.			

Annex B (normative): ORI vendor codes

Table B-1 provides the list of vendor codes allocated within ORI at the time of publication of the present document. The most recent version of this list is available at: http://portal.etsi.org/ORI/ORIvendor_codes.pdf.

Other vendors wishing to manufacture equipment conforming to the present document may request the assignment of a Vendor Code by following the procedure defined at: http://portal.etsi.org/ORI/ORIvendor_codes.pdf.

Device manufacturers shall use their assigned Vendor Code and NOT the Vendor Code of any other company.

Table B-1: ORI vendor codes

Vendor Code	Organisation Name
ALU	Alcatel-Lucent
FJT	Fujitsu
HWT	Huawei Technologies Co. Ltd
KAT	Kathrein-Werke KG
UBD	Ubidyne GmbH
ZTE	ZTE Corporation

NOTE: The existence of an assigned Vendor Code does not indicate that the company to which it was assigned is still manufacturing or trading under the name shown, nor that is a member or participant of ORI.

History

Document history		
V1.1.1	August 2012	Publication