OpenScape Business V2

Description

CSTA Interface Manual

Services for computer supported telecommunications applications (CSTA)

Phase III

Part 1: ECMA-269 related

Version 1.3

Content

Table of History 4
Legal Notice 5
Introduction 6
1 Establishing the CSTA Link 7
1.1 CSTA Link 7
1.1.1 SOFTWARE ARCHITECTURE AND PROTOCOL 7
1.2 Connection via LAN (TCP/IP) 8
1.3 Layer 5 header (OSI layer 5) 8
1.3.1 GENERAL INFORMATION 8
1.3.2 SESSION HEADER 8
1.4 Layer 7 9
1.4.1 CREDENTIALS 9
1.4.2 LAYER 7-ESTABLISHMENT/RELEASE THROUGH ACSE 9
1.4.3 LAYER 7- ESTABLISHMENT/RELEASE THROUGH ECMA-354 16
2 Layer 7 19
2.1 Event Report Service 19
2.2 Capability Exchange Services 20
2.2.1 SERVICES 20
2.2.2 SWITCHING FUNCTION DEVICES 32
2.3 System Services 33
~~2.3.1 REGISTRATION SERVICES~~ 33
2.3.2 SERVICES 34
2.4 Monitoring Services 35
2.4.1 SERVICES 35
2.5 Snapshot Services 39
2.5.1 SERVICES 39
2.6 Call Control Services and Events 42
2.6.1 SERVICES 42
2.6.2 EVENTS 55
2.7 Call Associated Features 72
2.7.1 SERVICES 72
2.7.2 EVENTS 75
~~2.8 Media Attachment Services and Events~~ 75
~~2.8.1 SERVICES~~ 75
~~2.8.2 EVENTS~~ 76
~~2.9 Routeing Services~~ 76
~~2.9.1 REGISTRATION SERVICES~~ 76
~~2.9.2 SERVICES~~ 76
2.10 Physical Device Features 76
2.10.1 SERVICES 76
2.10.2 EVENTS 88
2.11 Logical Device Features 90
2.11.1 SERVICES 90
2.11.2 EVENTS 100
2.12 Device Maintenance Events 105
2.12.1 EVENTS 105
~~2.13 Data Collection Services~~ 106
~~2.13.1 SERVICES~~ 106
2.14 Voice Unit Services & Events 106

2.14.1 SERVICES 107
2.14.2 EVENTS 109
2.15 Call Detail Record Services 110
2.15.1 SERVICES 110
2.16 Vendor Specific Extensions Services & Events 113
2.17 Samples 113
2.17.1 MAKECALL 113
2.17.2 KEYFEATURES 113
2.17.3 SAMPLE: GETSWITCHINGFUNCTIONCAPABILITIES 116
2.17.4 BUTTON AND LAMP MAP 126
2.17.5 KEY MODULE 55 134
2.18 OpenScape Business 135
2.18.1 OPENSCAPE BUSINESS PARAMETER CONSTRAINTS 135
2.18.2 OPENSCAPE BUSINESS CSTA LINK COMMUNICATION CONSTRAINTS 135
2.18.3 LIST OF MONITORABLE OPENSCAPE BUSINESS DEVICES 136
2.18.4 CSTA SERVICES OVERLOAD-REJECTION 136
2.18.5 DEVICE IDENTIFIER 136
2.18.6 DIAL STRING FORMATS 136
2.18.7 HOLD-SERVICE 137

Table of History

Version	*Changes*
1.0	First Version
1.1	New contact email-address for Technology Partner Team
1.2	Support of CSTA-XML, IP Phone 35/55
1.3	Formal release for OpenScape Business V2R0

Comments and corrections are welcome, please contact:

Technology Partner Team

technology.partners@unify.com

+49 (89) 7007-22386

Legal Notice

In this document partly text from ECMA-269 is copied.

Copyright notice from ECMA-269 9th Edition:

"COPYRIGHT NOTICE
This document may be copied, published and distributed to others, and certain derivative works of it may be prepared, copied, published, and distributed, in whole or in part, provided that the above copyright notice and this Copyright Licence and Disclaimer are included on all such copies and derivative works.
The only derivative works that are permissible under this Copyright License and Disclaimer are:
(i) works which incorporate all or portion of this document for the purpose of providing commentary or explanation (such as an annotated version of the document),
(ii) works which incorporate all or portion of this document for the purpose of incorporating features that provide accessibility,
(iii) translations of this document into languages other than English and into different formats and
(iv) works by making use of this specification in standard conformant products by implementing (e.g. by copy and paste wholly or partly) the functionality therein.
However, the content of this document itself may not be modified in any way, including by removing the copyright notice or references to Ecma International, except as required to translate it into languages other than English or into a different format.
The official version of an Ecma International document is the English language version on the Ecma International website. In the event of discrepancies between a translated version and the official version, the official version shall govern.
The limited permissions granted above are perpetual and will not be revoked by Ecma International or its successors or assigns.
This document and the information herein is provided on an "AS IS" basis and ECMA INTERNATIONAL DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."

Introduction

OpenScape Business provides a link to host computers using the CSTA protocol (Phase III) defined in ISO18051 and ISO18052 (ECMA 269, ECMA-285).

- Part 1 of this document describes the ECMA-269 related matters.
- Part 2 of this document describes the ECMA-285 related matters. The provided parameters are shown at chapter PICS.
- Part 3 of this document describes in analogy to ECMA-TR/82 typical call flows.
- Part 4 of this document describes the supported Escape-Services

CSTA is provided by the following OpenScape Business configurations:

- OpenScape Business X3/X5/X8 equipped with an UC Booster Card or UC Booster Server
- OpenScape Business Server

Note:
In this document one of the above described configurations is meant, when OpenScape Business is mentioned.

In this document some text is shown as crossed out (~~like this~~). These parts are specified by ECMA, but are not supported by OpenScape Business.

With OpenScape Business multiple CSTA access points can be available:

- A single node OpenScape Business offers the possibility of monitoring and controlling the local resources (see 2.18.3 List of monitorable OpenScape Business Devices).
- An OpenScape Business, that is part of a homogenous network of OpenScape Business-systems, and that is a slave-node offers the possibility of monitoring and controlling the local resources (see 2.18.3 List of monitorable OpenScape Business Devices). The networking trunks are part of the local resources. Network calls are external calls from CSTA point of view on this slave-node.
- An OpenScape Business, that is part of a homogenous network of OpenScape Business-systems, and that is the master-node offers the possibility of monitoring and controlling all resources (see 2.18.3 List of monitorable OpenScape Business Devices) of the networked OpenScape Business Systems, either equipped with UC Booster Card / UC Booster Server or not. Only the networking trunks cannot be monitored. Network calls are internal calls from CSTA point of view on the master node.

1 Establishing the CSTA Link

1.1 CSTA Link

Each CSTA access point of OpenScape Business can be connected with up to 4 TCP/IP-Links. Depending on the configuration of the OpenScape Business some of them can be used by internal CSTA-applications. If internal applications get deactivated, more links are available for applications outside the OpenScape Business.

1.1.1 Software architecture and protocol

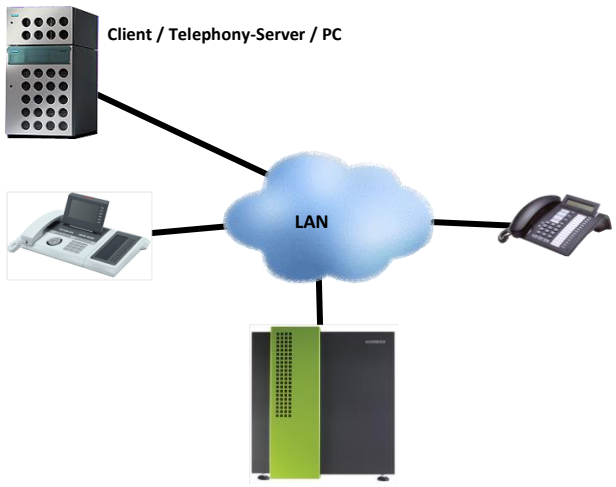
All CSTA applications are based on the OSI ISO 7-layer reference model and Use layers 1, 2, 5 and 7:

Interface	TCP/IP
Layer 7	CSTA
Layer 6	- ASN.1 -
Layer 5	Layer 5-Header
Layer 4	TCP
Layer 3	IP
Layer 2	LAN
Layer 1	LAN

Before data transfer within an application on CSTA is possible, the appropriate layers have to be established. This is described in the following sections. The initiative for session establishment can be taken only from the host computer side.

1.2 Connection via LAN (TCP/IP)

The OpenScape Business acts as TCP/IP Server that can be connected from a CSTA-client (the telephony server (PC)).



Connection parameters for the client:

- IP-address OpenScape Business
- TCP-port (default 8800)
- The data stream is framed
- Layer5 (header)
- Layer6 (ASN.1 / ECMA-323 Annex J.2 TCP without SOAP)
- Layer7 (CSTA)

1.3 Layer 5 header (OSI layer 5)

1.3.1 General information

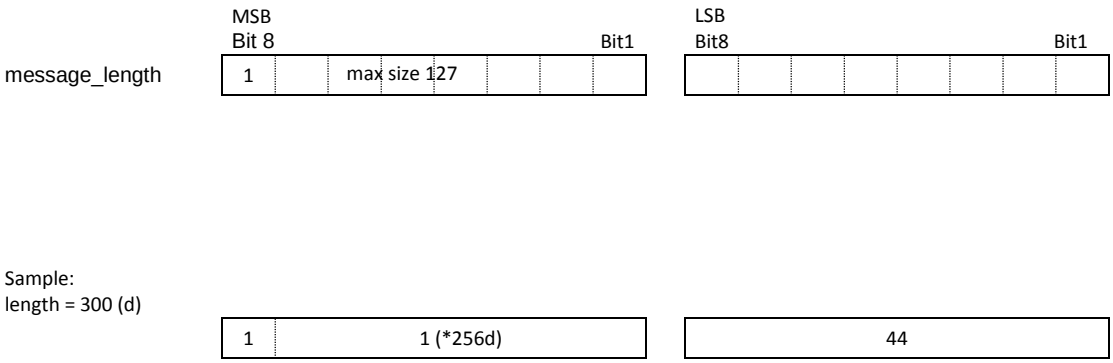
The layer 5-message begins with a header element.

Session Header	Data (ASN.1 coded)

1.3.2 Session header

The session header element has the following structure (max. length 1000 bytes):

system_application	CTI3_APPLICATION = 0x26		1 byte
message_length (MSB)	1	0 - 127	1 byte
message_length (LSB)	0 - 255		1 byte
ASN.1-Data (ACSE,ROSE)			n bytes



CSTA services can be transferred in 1 message element.

- **Bit 1-8 of LSB**
  This field contains the lower byte of message length.
- **Bit 8 of MSB** This bit is always **1**
- **Bit 1-7 of MSB** This field is used to extend the message size in LSB to values greater than 255.
  MsgLength = (MSB1-7) * 256d + LSB

For CSTA-XML the session header is according ECMA 323 Annex J.2.

1.4 Layer 7

Establishment of Layer 7 is either possible through ACSE for CSTA ASN.1 or CSTA-XML, or through StartApplicationSession according to ECMA-354 for CSTA-XML.

1.4.1 Credentials

The credentials have to be configured in OpenScape Business Assistant/Expert mode/Middleware/CSTA Service Provider (CSP):

- CSTA user id (max 24 chars)
- CSTA user password (max. 24 chars)

Note:
If the credentials are not configured, the login will be rejected. **If CSP 'Service will be started automatically' is not checked, there will be no CSTA-service available.**

1.4.2 Layer 7-Establishment/Release through ACSE

For establishing a CSTA connection the ACSE protocol has to be used. The activity starts always at the host with an ACSE-request (AARQ). That means the Explicit Association (see ISO 18051/18052 or ECMA-269/ECMA-285) is used.
The credentials for CSTA have to be transmitted in field 'calling-authentication-value.external/'authentication-name' and 'authentication-password'.

```
aCSE.aarq
{ -- SEQUENCE --
   application-context-name = {iso(1) identified-organization(3) icd-ecma(12) standard(0) csta(218)},
   sender-acse-requirements = '10'B -- NrBits = 2 --
   { -- true bits --
      authentication
   },
   calling-authentication-value.external
   { -- SEQUENCE --
      encoding.single-ASN1-type
      { -- SEQUENCE --
         sender-authentication
```

```
        { -- SEQUENCE --
            authentication-name = '41 4D 48 4F 53 54 00 4F 53 4F'H,
            authentication-password = "77777" '37 37 37 37 37'H
        }
    }
  },
  user-information
  { -- SEQUENCE OF --
    { -- SEQUENCE --
        encoding.single-ASN1-type.newDefinition
        { -- SEQUENCE --
            cSTAVersion = '0001000000000000'B -- NrBits = 16 --
            { -- true bits --
                versionFour
            }
        }
    }
  }
}
```

Alternativly the Graph-string can be used:

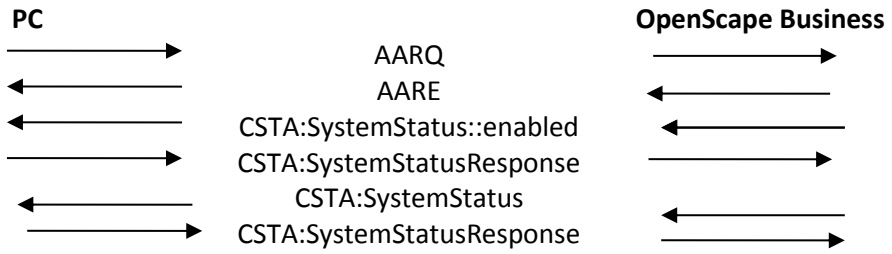
Name;login=user;passwd=password;

```
aCSE.aarq
{ -- SEQUENCE --
  application-context-name = {iso(1) identified-organization(3) icd-ecma(12) standard(0) csta(218)},
  calling-authentication-value.charstring = ";login=AMHOST;passwd=77777;native=false" '3B 6C 6F 67 69 6E 3D 41 4D 48
4F 53 54 3B 70 61 73 73 77 64 3D 37 37 37 37 37 3B 6E 61 74 69 76 65 3D 66 61 6C 73 65'H,
  user-information
  { -- SEQUENCE OF --
    { -- SEQUENCE --
        encoding.single-ASN1-type.newDefinition
        { -- SEQUENCE --
            cSTAVersion = '00000100'B -- NrBits = 8 --
            { -- true bits --
                versionSix
            }
        }
    }
  }
            }
```

The CSTA-Version ***versionFour*** selects CSTA ASN.1.

The CSTA-Version ***versionSix*** selects CSTA XML according ECMA 323 Annex J.2 'CSTA XML over TCP without SOAP'.

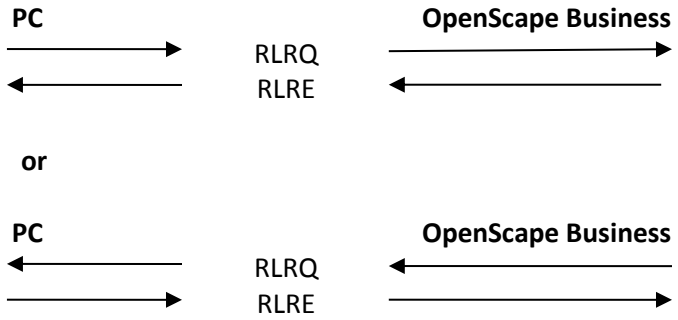
Setup:



If the request is accepted, the OpenScape Business acknowledges with an AARE message with result = 0 (accepted). From this point the CSTA link is established and the OpenScape Business sends the first CSTA message (SystemStatus::enabled) followed by a systemStatus showing the actual system status.

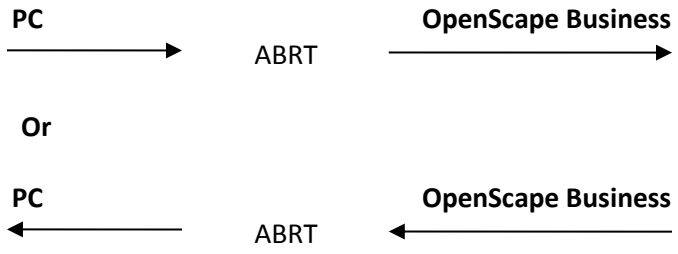
Release:

Release of the CSTA session is possible from both sides. A RLRQ has to be acknowledged.



Abort (Release without acknowledge):

In urgent cases the release of the CSTA session can be done via ABRT (ACSE Abort). That means that the session is closed directly without any acknowledge.



AARQ A-ASSOCIATE-REQUEST APDU

```
AARQ-apdu ::= [ APPLICATION 0 ] IMPLICIT SEQUENCE
    {protocol-version                    [0]   IMPLICIT BIT STRING                    { version1 (0) }
                                                                                DEFAULT {version1},
    application-context-name             [1]   Application-context-name,
    ~~called-AP-title~~                  ~~[2]~~   ~~AP-title~~                       ~~OPTIONAL~~,
    ~~called-AE-qualifier~~              ~~[3]~~   ~~AE-qualifier~~                   ~~OPTIONAL~~,
    ~~called-AP-invocation-identifier~~  ~~[4]~~   ~~AP-invocation-identifier~~       ~~OPTIONAL~~,
    ~~called-AE-invocation-identifier~~  ~~[5]~~   ~~AE-invocation-identifier~~       ~~OPTIONAL~~,
    ~~calling-AP-title~~                 ~~[6]~~   ~~AP-title~~                       ~~OPTIONAL~~,
    ~~calling-AE-qualifier~~             ~~[7]~~   ~~AE-qualifier~~                   ~~OPTIONAL~~,
    ~~calling-AP-invocation-identifier~~ ~~[8]~~   ~~AP-invocation-identifier~~       ~~OPTIONAL~~,
    ~~calling-AE-invocation-identifier~~ ~~[9]~~   ~~AE-invocation-identifier~~       ~~OPTIONAL~~,
    -- The following field shall not be present if only the Kernel is used.
    sender-acse-requirements             [10]  IMPLICIT ACSE-requirements           OPTIONAL,
    -- The following field shall only be present if the Authentication functional unit is selected.
    ~~mechanism-name~~                   ~~[11]~~  ~~IMPLICIT Mechanism-name~~       ~~OPTIONAL~~,
    -- The following field shall only be present if the Authentication functional unit is selected.
    calling-authentication-value         [12]  EXPLICIT Authentication-value        OPTIONAL,
    -- The following field shall only be present if the Application Context Negotiation functional
      unit is selected
    ~~application-context-name-list~~    ~~[13]~~  ~~IMPLICIT Application-context-name-list~~ ~~OPTIONAL~~,
    ~~implementation-information~~       ~~[29]~~  ~~IMPLICIT Implementation-data~~   ~~OPTIONAL~~,
    user-information                     [30]  IMPLICIT Association-information     OPTIONAL
    }
```

AARE A-ASSOCIATE-RESPONSE APDU

```
AARE-apdu ::= [ APPLICATION 1 ] IMPLICIT SEQUENCE
    { protocol-version                   [0]   IMPLICIT BIT STRING                    {version1 (0) }
                                                                                DEFAULT { version1 },
    application-context-name             [1]   Application-context-name,
    result                               [2]   Associate-result,
    result-source-diagnostic             [3]   Associate-source-diagnostic,
    ~~responding-AP-title~~              ~~[4]~~   ~~AP-title~~                       ~~OPTIONAL~~,
    ~~responding-AE-qualifier~~          ~~[5]~~   ~~AE-qualifier~~                   ~~OPTIONAL~~,
    ~~responding-AP-invocation-identifier~~ ~~[6]~~ ~~AP-invocation-identifier~~      ~~OPTIONAL~~,
    ~~responding-AE-invocation-identifie r~~ ~~[7]~~ ~~AE-invocation-identifier~~      ~~OPTIONAL~~,

    -- The following field shall not be present if only the Kernel is used.
    responder-acse-requirements          [8]   IMPLICIT ACSE-requirements           OPTIONAL,
    -- The following field shall only be present if the Authentication functional unit is selected.
    ~~mechanism-name~~                   ~~[9]~~   ~~IMPLICIT Mechanism-name~~       ~~OPTIONAL~~,
    -- The following field shall only be present if the Authentication functional unit is selected.
    responding-authentication-value [10] EXPLICIT Authentication-value               OPTIONAL,
    -- The following field shall only be present if the Application Context Negotiation functional
    unit is selected
    ~~application-context-name-list~~    ~~[11]~~  ~~IMPLICIT Application-context-name-list~~ ~~OPTIONAL~~,
    ~~implementation-information~~       ~~[29]~~  ~~IMPLICIT Implementation-data~~   ~~OPTIONAL~~,
    user-information                     [30]  IMPLICIT Association-information     OPTIONAL
    }
```

RLRQ A-RELEASE-REQUEST APDU

```
RLRQ-apdu ::= [ APPLICATION 2 ] IMPLICIT SEQUENCE
            { reason            [0]   IMPLICIT Release-request-reason        OPTIONAL,
              user-information  [30]  IMPLICIT Association-information        OPTIONAL
            }
```

RLRE A-RELEASE-RESPONSE APDU

```
RLRE-apdu  ::=  [ APPLICATION 3 ] IMPLICIT SEQUENCE
                { reason              [0]    IMPLICIT Release-response-reason        OPTIONAL,
                  user-information    [30]   IMPLICIT Association-information        OPTIONAL
                }
```

ABRT A-ABORT APDU

```
ABRT-apdu ::= [ APPLICATION 4 ] IMPLICIT SEQUENCE
        {       abort-source                [0]    IMPLICIT ABRT-source,
        -- The following field shall not be present if only the Kernel is used.
                abort-diagnostic            [1]    IMPLICIT ABRT-diagnostic        OPTIONAL,
                user-information            [30]   IMPLICIT Association-information OPTIONAL
        }
```

1.4.2.1 ACSE-Parameter

```
ABRT-diagnostic ::= ENUMERATED{   no-reason-given                          (1),
                  protocol-error                                           (2),
                  authentication-mechanism-name-not-recognized             (3),
                  authentication-mechanism-name-required                   (4),
                  authentication-failure                                   (5),
                  authentication-required                                  (6)  }
```

```
ABRT-source  ::=  INTEGER{ acse-service-user          (0),
                           acse-service-provider      (1)  }
```

```
ACSE-requirements  ::=  BIT STRING
{                 authentication                  (0),
                  application-context-negotiation (1)  }
```

```
Application-context-name ::= OBJECT IDENTIFIER
--  Name from CSTA III:
  CSTA-application-context-name
  { iso( 1) identified-organization( 3) icd-ecma( 12)
    standard( 0) csta( 218)}
-- Common to all CSTA protocol versions
```

```
Associate-result  ::= INTEGER{        accepted              (0),
                                      rejected-permanent    (1),
                                      rejected-transient    (2)  }
```

```
Associate-source-diagnostic  ::=  CHOICE
{   acse-service-user                    [1]  INTEGER
                 {   null                                                    (0),
                     no-reason-given                                         (1),
                     application-context-name-not-supported                  (2),
                     calling-AP-title-not-recognized                         (3),
                     calling-AP-invocation-identifier-not-recognized         (4),
                     calling-AE-qualifier-not-recognized                     (5),
                     calling-AE-invocation-identifier-not-recognized         (6),
                     called-AP-title-not-recognized                          (7),
                     called-AP-invocation-identifier-not-recognized          (8),
                     called-AE-qualifier-not-recognized                      (9),
                     called-AE-invocation-identifier-not-recognized         (10),
                     authentication-mechanism-name-not-recognized           (11),
                     authentication-mechanism-name-required                 (12),
                     authentication-failure                                 (13),
                     authentication-required                                (14)  },
acse-service-provider                    [2]  INTEGER
                 {   null                          (0),
                     no-reason-given               (1),
                     no-common-acse-version        (2)  }
}
```

```
Association-information ::=  SEQUENCE OF  ExtInformationForCSTA
```

```
ExtInformationForCSTA ::=  [UNIVERSAL 8] IMPLICIT SEQUENCE
                 {   direct-reference          OBJECT IDENTIFIER                          OPTIONAL,
                     indirect-reference        INTEGER                                    OPTIONAL,
                     data-val-descriptor       [UNIVERSAL 7] IMPLICIT OCTET  STRING       OPTIONAL,
                     encoding                  EncodingCsta
                 }
```

```
EncodingCsta ::= CHOICE
                 {   single-ASN1-type          [0]     ACSEUserInformationForCSTA,
                     octet-aligned             [1]     IMPLICIT     OCTET STRING,
                     arbitrary                 [2]     IMPLICIT     BIT STRING
                 }
```

```
Authentication-value  ::=  CHOICE
                 {   charstring          [0]    IMPLICIT      GraphicString,
                     bitstring           [1]    IMPLICIT      BIT        STRING,
                     external            [2]    IMPLICIT      PrivatExtPassword,
                     other               [3]    IMPLICIT      SEQUENCE
                      {
                          other-mechanism-name  Mechanism-name,
                          other-mechanism-value  ANY DEFINED BY
                                                      other-mechanism-name  }
                 }
```

```
Release-request-reason  ::=  INTEGER
                 {   normal              (0),
                     urgent              (1),
                     user-defined        (30)   }
```

```
Release-response-reason  ::=  INTEGER
{                    normal              (0),
                     not-finished        (1),
                     user-defined       (30)  }
```

```
PrivatExtPassword ::=  [UNIVERSAL 8] IMPLICIT SEQUENCE
{                    direct-reference          OBJECT IDENTIFIER                    OPTIONAL,
                     indirect-reference        INTEGER                              OPTIONAL,
                     data-val-descriptor       [UNIVERSAL 7] IMPLICIT OCTET  STRING OPTIONAL,
                     encoding          EncodingPassword    }
```

```
EncodingPassword ::= CHOICE
{                    single-ASN1-type   [0]      IMPLICIT  External-authentication,
                     octet-aligned      [1]      IMPLICIT  OCTET STRING,
                     arbitrary          [2]      IMPLICIT  BIT STRING    }
```

```
External-authentication  ::=  SEQUENCE
{       sender-authentication              [0]     IMPLICIT  Sender-authentication          OPTIONAL,
        responder-authentication           [1]     IMPLICIT  Responder-authentication       OPTIONAL   }
```

```
Sender-authentication  ::=  SEQUENCE
{                    authentication-name          OCTET STRING,          -- CSTA  user  id  see
```
Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.

```
                     authentication-password      OCTET STRING  }        --  CSTA  user  password  see
```
Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.

```
Responder-authentication ::=  SEQUENCE
{                    aps-stamp                          OCTET STRING,          -- e.g.: “M-IM10.00”
                     system-version                     OCTET STRING   }       -- e.g. “3.0.0.020”
```

1.4.2.2 CSTA-application-context-information

```
ACSEUserInformationForCSTA ::= CHOICE
{  oldDefinition        OldACSEUserInformationForCSTA,       -- as defined in ECMA 218
   newDefinition        [0] IMPLICIT        NewACSEUserInformationForCSTA }

OldACSEUserInformationForCSTA ::= SEQUENCE
{  cSTAVersion                                    CSTAVersion,
   cSTAFunctionsRequiredByApplication             CSTAFunctionality,
   cSTAFunctionsThatCanBeSupplied                 CSTAFunctionality       }

NewACSEUserInformationForCSTA ::= SEQUENCE
{  cSTAVersion                                              CSTAVersion,
--
--Note that the following two parameters are defined in ECMA-218
--(CSTA Phase II).©ISO/IEC ISO/IEC 18052:1999 (E)
--They are defined in the context of CSTA Phase II Services and Events for
--backward compatibility considerations.
--For CSTA Phase III (and later) implementations should use CSTA Capability Exchange
  Services
--to determine the Services and Events supported by a switching function.
--
   cSTAFunctionsRequiredByApplication                [0] IMPLICIT CSTAFunctionality                OPTIONAL,
   cSTAFunctionsThatCanBeSupplied                    [1] IMPLICIT CSTAFunctionality                OPTIONAL,
   cSTAPrivateDataVersionList                        [2] IMPLICIT CSTAPrivateDataVersionList       OPTIONAL
}
```

```
CSTAVersion ::= BIT STRING
{                   versionOne          (0),        -- CSTA protocol version defined in ECMA 180
                    versionTwo          (1),        -- CSTA protocol version defined in ECMA 218
                    versionThree        (2),        -- CSTA protocol version defined in ECMA 285 1st
                    versionFour         (3),        -- CSTA protocol version defined in ISO 18052
                    versionFive         (4),        -- CSTA protocol version defined in ECMA 285 2nd
                    versionSix          (5),        -- CSTA protocol version defined in ECMA 323
                    versionSeven        (6),        -- Reserved for future use
                    versionEight        (7),        -- Reserved for future use
                    versionNine         (8),        -- Reserved for future use
                    versionTen          (9),        -- Reserved for future use
                    versionEleven       (10),       -- Reserved for future use
                    versionTwelve       (11),       -- Reserved for future use
                    versionThirteen     (12),       -- Reserved for future use
                    versionFourteen     (13),       -- Reserved for future use
                    versionFifteen      (14),       -- Reserved for future use
                    versionSixteen      (15)  }     -- Reserved for future use

-- When requesting an application association, the requestor will set the corresponding bits for each version that it supports. When
   responding, the responder shall specify only one bit corresponding to the version selected (the highest common to both
   systems).
```

Note: some parameters of ECMA-269 9th edition are imported and referenced.

1.4.3 Layer 7- Establishment/Release through ECMA-354

For usage of CSTA XML according ECMA 323 Annex J.2 'CSTA XML over TCP' also a login via ECMA-354 StartApplicationSession can be used. OpenScape Business supports protocol version 'http://www.ecma-international.org/standards/ecma-323/csta/ed4'.

```
0001<?xml version="1.0" encoding="UTF-8"?><StartApplicationSession
xmlns="http://www.ecma-international.org/standards/ecma-354/appl_session">
  <applicationInfo>
    <applicationID>CB3</applicationID>
```

```
    <applicationSpecificInfo>
      <user>AMHOST</user>
      <password>77777</password>
      <native>false</native>
    </applicationSpecificInfo>
  </applicationInfo>
  <requestedProtocolVersions>
    <protocolVersion>http://www.ecma-international.org/standards/ecma-
323/csta/ed4</protocolVersion>
  </requestedProtocolVersions>
  <requestedSessionDuration>60</requestedSessionDuration>
</StartApplicationSession>
```


If the request is accepted, the OpenScape Business acknowledges with StartApplicationsessionPosResponse. From this point the CSTA link is established and the OpenScape Business sends the first CSTA message (SystemStatus::enabled) followed by a systemStatus showing the actual system status.


```
0001<?xml version="1.0" encoding="UTF-8"?><StartApplicationSessionPosResponse
xmlns="http://www.ecma-international.org/standards/ecma-354/appl_session">
  <sessionID>141976992</sessionID>
  <actualProtocolVersion>http://www.ecma-international.org/standards/ecma-
323/csta/ed4</actualProtocolVersion>
  <actualSessionDuration>10</actualSessionDuration>
  <extensions>
    <privateData>
      <private xmlns:scx="http://www.unify.com/schema/csta">
        <scx:ApsInfo>M-FP10.00.506:64</scx:ApsInfo>
      </private>
    </privateData>
  </extensions>
</StartApplicationSessionPosResponse>
```


On CSTA-XML StopApplicationsession can be used to release a connection.


```
0002<?xml version="1.0" encoding="UTF-8"?><StopApplicationSession
xmlns="http://www.ecma-international.org/standards/ecma-354/appl_session">
  <sessionID>60</sessionID>
  <sessionEndReason>
    <definedEndReason>normal</definedEndReason>
  </sessionEndReason>
</StopApplicationSession>


0002<?xml version="1.0" encoding="UTF-8"?><StopApplicationSessionPosResponse
xmlns="http://www.ecma-international.org/standards/ecma-354/appl_session"/>
```


1.4.3.1 Application Session Management

OpenScape Business does not support Application Session Management.
In case of an interrupted socket-connection the CSTA-Session is also terminated. All MonitorCrossRefIDs become invalid.

2 Layer 7

To have proper display of external call numbers via CSTA, CSTA requests Least Cost Routing (LCR) active in OpenScape Business. See documentation of OpenScape Business how to configure LCR.

2.1 Event Report Service

```
CSTAEventReport OPERATION ::=
{       ARGUMENT                        CSTAEventReportArgument
        ALWAYS RESPONDS                 FALSE
        CODE                            local:21
}
```

```
CSTAEventReportArgument ::= SEQUENCE
{       crossRefIdentifier              MonitorCrossRefID,
        eventSpecificInfo               EventSpecificInfo,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

```
EventSpecificInfo ::= CHOICE
{       callControlEvents               [0] CallControlServAndEvents,
        callAssociatedEvents            [1] CallAssociationEvents,
        mediaAttachmentEvents           [2] MediaAttachmentEvents,
        physicalDeviceEvents            [3] PhysicalDeviceEvents,
        logicalDeviceEvents             [4] LogicalDeviceEvents,
        deviceMaintenanceEvents         [5] DeviceMaintenanceEvents,
        voiceUnitEvents                 [6] VoiceUnitEvents,
        vendorSpecEvents                [7] VendorSpecEvents}
```

```
CallControlEvents ::= CHOICE
{       bridged                         [ 0] IMPLICIT BridgedEvent,
        callCleared                     [ 1] IMPLICIT CallClearedEvent,
        conferenced                     [ 2] IMPLICIT ConferencedEvent,
        connectionCleared               [ 3] IMPLICIT ConnectionClearedEvent,
        delivered                       [ 4] IMPLICIT DeliveredEvent,
        digitsDialed                    [ 5] IMPLICIT DigitsDialedEvent,
        diverted                        [ 6] IMPLICIT DivertedEvent,
        established                     [ 7] IMPLICIT EstablishedEvent,
        failed                          [ 8] IMPLICIT FailedEvent,
        held                            [ 9] IMPLICIT HeldEvent,
        networkCapabilitiesChanged      [10] IMPLICIT NetworkCapabilitiesChangedEvent,
        networkReached                  [11] IMPLICIT NetworkReachedEvent,
        offered                         [12] IMPLICIT OfferedEvent,
        originated                      [13] IMPLICIT OriginatedEvent,
        queued                          [14] IMPLICIT QueuedEvent,
        retrieved                       [15] IMPLICIT RetrievedEvent,
        serviceInitiated                [16] IMPLICIT ServiceInitiatedEvent,
        transferred                     [17] IMPLICIT TransferedEvent}
```

```
CallAssociatedEvents ::= CHOICE
{       callInformation                 [ 0] IMPLICIT CallInformationEvent,
        charging                        [ 1] IMPLICIT ChargingEvent,
        digitsGeneratedEvent            [ 2] IMPLICIT DigitsGeneratedEvent,
        telephonyTonesGeneratedEvent    [ 3] IMPLICIT TelephonyTonesGeneratedEvent,
        serviceCompletionFailure        [ 4] IMPLICIT ServiceCompletionFailureEvent}
```

```
MediaAttachmentEvents ::= CHOICE
{       mediaAttached                   [ 0] IMPLICIT MediaAttachedEvent,
        mediaDetached                   [ 1] IMPLICIT MediaDetachedEvent}
```

```
PhysicalDeviceFeatureEvents ::= CHOICE
{       buttonInformation               [ 0] IMPLICIT ButtonInformationEvent,
        buttonPress                     [ 1] IMPLICIT ButtonPressEvent,
        displayUpdated                  [ 2] IMPLICIT DisplayUpdatedEvent,
        hookswitch                      [ 3] IMPLICIT HookswitchEvent,
        lampMode                        [ 4] IMPLICIT LampModeEvent,
        messageWaiting                  [ 5] IMPLICIT MessageWaitingEvent,
        microphoneGain                  [ 6] IMPLICIT MicrophoneGainEvent,
        microphoneMute                  [ 7] IMPLICIT MicrophoneMuteEvent,
        ringerStatus                    [ 8] IMPLICIT RingerStatusEvent,
        speakerMute                     [ 9] IMPLICIT SpeakerMuteEvent,
        speakerVolume                   [10] IMPLICIT SpeakerVolumeEvent}
```

```
LogicalDeviceFeatureEvents ::= CHOICE
{       agentBusy                       [ 0] IMPLICIT AgentBusyEvent,
        agentLoggedOn                   [ 1] IMPLICIT AgentLoggedOnEvent,
        agentLoggedOff                  [ 2] IMPLICIT AgentLoggedOffEvent,
        agentNotReady                   [ 3] IMPLICIT AgentNotReadyEvent,
        agentReady                      [ 4] IMPLICIT AgentReadyEvent,
        agentWorkingAfterCall           [ 5] IMPLICIT AgentWorkingAfterCallEvent,
        autoAnswer                      [ 6] IMPLICIT AutoAnswerEvent,
        autoWorkMode                    [ 7] IMPLICIT AutoWorkModeEvent,
        callBack                        [ 8] IMPLICIT CallBackEvent,
        callBackMessage                 [ 9] IMPLICIT CallBackMessageEvent,
        callerIDStatus                  [10] IMPLICIT CallerIDStatusEvent,
        doNotDisturb                    [11] IMPLICIT DoNotDisturbEvent,
        forwarding                      [12] IMPLICIT ForwardingEvent,
        routeingMode                    [13] IMPLICIT RouteingModeEvent}
```

```
DeviceMaintenanceEvents ::= CHOICE
{       backInService                   [ 0] IMPLICIT BackInServiceEvent,
        deviceCapabilityChanged         [ 1] IMPLICIT DeviceCapsChangedEvent,
        outOfService                    [ 2] IMPLICIT OutOfServiceEvent}
```

```
VoiceUnitEvents ::= CHOICE
{       play                            [ 0] IMPLICIT PlayEvent,
        record                          [ 1] IMPLICIT RecordEvent,
        review                          [ 2] IMPLICIT ReviewEvent,
        stop                            [ 3] IMPLICIT StopEvent,
        suspendPlay                     [ 4] IMPLICIT SuspendPlayEvent,
        suspendRecord                   [ 5] IMPLICIT SuspendRecordEvent,
        voiceAttributesChange           [ 6] IMPLICIT VoiceAttributesChangeEvent}
```

```
VendorSpecEvents::= CHOICE
{       privateEvent                    [ 0] IMPLICIT PrivateEvent}
```

2.2 Capability Exchange Services

2.2.1 Services

2.2.1.1 Get Logical Device Information

The Get Logical Device Information service is used to obtain the current set of characteristics/ Capabilities associated with the physical element of a given device. See also 2.18.3 List of monitorable OpenScape Business Devices.

Operation

```
getLogicalDeviceInformation OPERATION ::=
{       ARGUMENT                        GetLogicalDeviceInformationArgument
        RESULT                          GetLogicalDeviceInformationResult
        ERRORS                          {universalFailure}
        CODE                            local:202
}
```

Service request

```
GetLogicalDeviceInformationArgument ::= SEQUENCE
{       device                          DeviceID,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device being queried.

Service response

```
GetLogicalDeviceInformationResult ::= SEQUENCE
{       deviceCategory                  [ 0] IMPLICIT DeviceCategory            DEFAULT station,
        groupDeviceAttributes           [ 1] IMPLICIT GroupDeviceAttributes     OPTIONAL,
        namedDeviceTypes                [ 2] IMPLICIT NamedDeviceTypes          OPTIONAL,
        shortFormDeviceID               [3] IMPLICIT DeviceID                   OPTIONAL
        hasPhyisicalElement             BOOLEAN,
        acdModels                       Bitmap,
        agentLogOnModels                Bitmap,
        appearanceAddressable           BOOLEAN,
        appearanceType                  Enumerated,
        appearanceList                  List of Characters,
        otherPhysicalDeviceList         List of DeviceIDs,
        miscMonitorCaps                 Bitmap                                  OPTIONAL,,
        associatedGroupList             List of DeviceIDs
        maxCallBacks                    Value                                   OPTIONAL,
        maxAutoAnswerRings              Value                                   OPTIONAL,
        maxActiveCalls                  Value                                   OPTIONAL,
        maxHeldCalls                    Value                                   OPTIONAL,
        maxFwdSettings                  Value                                   OPTIONAL,
        maxDevicesInConf                Value                                   OPTIONAL,
        transAndConfSetup               Bitmap                                  OPTIONAL,
        deviceOnDeviceMonitorFilter     [ 5] IMPLICIT MonitorFilter             OPTIONAL,
        deviceOnConnectionMonitorFilter [ 6] IMPLICIT MonitorFilter             OPTIONAL,
        callOnDeviceMonitorFilter       [ 7] IMPLICIT MonitorFilter             OPTIONAL,
        callOnConnectionMonitorFilter   [ 8] IMPLICIT MonitorFilter             OPTIONAL,
        mediaClassSupport               Bitmap                                  OPTIONAL,
        mediaServiceCapsList            List of Structures,
        connectionRateList              List of Values                          OPTIONAL,
        delayToleranceList              List of Values                          OPTIONAL
        numberOfChannels                Value                                   OPTIONAL
        maxChannelBind                  Value                                   OPTIONAL,
        routeingServList                RouteingServList,
        logDevServList                  Structure,
        logDevEvtsList                  Structure,
        devicemaintEvtsList             DeviceMaintEvtsList,
        security                        CSTASecurityData                        OPTIONAL,
        privateData                     CSTAPrivateData                         OPTIONAL}
```

Parameter Name	***Type***	***Description***
deviceCategory	Enumerated	Specifies the device category (station, ACD device, etc.) of the device being queried. The complete set of possible values is: - ACD - Group - Network Interface (i.e., trunk) - Park - Routeing - Station (default) - Voice Unit - Other
groupDeviceAttributes	Bitmap	Specifies the group device attributes of the device being queried. If a bit is TRUE then the specified attribute is present. The following is the list of bits (multiple bits may be set): - ACD - Hunt - Pick - Other This parameter shall be provided if the deviceCategory is Group, otherwise it shall not be provided.
hasLogicalElement	Boolean	Specifies if the device has a logical element associated with this device identifier. The complete set of possible values is: - FALSE - The device does not have a logical element. - TRUE - The device does have a logical element. The device identifier in the service request should be used with the Get Logical Device Information service to obtain the logical elements characteristics for this device.
deviceOnDeviceMonitorFilter	MonitorFilter	Specifies the complete monitorFilter parameter that the device supports with respect to device-type monitoring. This parameter shall be provided if this form of device-type monitoring is supported, otherwise the parameter shall not be provided. The information in the monitor filter parameters used in the Get Logical Device Information and the Get Physical Device Information services should be the same when the same device identifier is used (assuming that the device identifier has a logical and a physical element).
physDevServList	PhysDevServList	Specifies a list of bitmaps. Each bitmap entry represents a Physical Device service that is supported by the specified device. This includes the following categories of services: Physical Device Feature services This parameter shall be provided if the switching function supports any of these categories of services for this device. If a physical device services bitmap entry is not included in the list, then the service is not supported by the specified device.
physDevEvtsList	PhysDevEvtsList	Specifies a list of bitmaps. Each bitmap entry represents a Physical Device Event that is supported by the specified device. This includes the following categories of events: Physical Device Feature events This parameter shall be provided if the switching function supports any of these categories of events for this device. If a physical device events bitmap entry is not included in the list, then the event is not supported by the specified device.

Note:
For Mulap the getLogicalDeviceInformationResult will show the configuration. In the following sample the Mulap D2 has the three members D21, D22 and D23:

Monitored device D2	
GetLogicalDeviceInfoRequest	
• device	D2
GetLogicalDeviceInfoResult	
• deviceCategory	Other
• groupDeviceAttributes	other
• hasPhysicalElement	False
• acdModels	()
• agentlogOnModels	()
• appearanceAddressable	False
• appearanceType	Basic Bridged
• appearanceList	
• otherPhysicalDeviceList	D21 D22 D23
• miscMonitorCaps	Group inclusive
• associatedGroupList	D21 D22 D23
• deviceOnDeviceMonitorFilter	()
• logDevServList	()
• logDevEvtsList	()
• deviceMaintEvtsList	()

2.2.1.2 Get Physical Device Information

Description

The Get Physical Device Information service is used to obtain the current set of characteristics / capabilities associated with the physical element of a given device.

Note

See also 2.18.3 List of monitorable OpenScape Business Devices. The devices have the deviceOnDeviceMonitorFilter parameter set accordingly to this list.

Operation

```
getPhysicalDeviceInformation OPERATION ::=
{       ARGUMENT                        GetPhysicalDeviceInformationArgument
        RESULT                          GetPhysicalDeviceInformationResult
        ERRORS                          {universalFailure}
        CODE                            local:202
}
```

Service request

```
GetPhysicalDeviceInformationArgument ::= SEQUENCE
{       device                          DeviceID,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device being queried.

Service response

```
GetPhysicalDeviceInformationResult ::= SEQUENCE
{       deviceCategory                  [ 0] IMPLICIT DeviceCategory            DEFAULT station,
        groupDeviceAttributes           [ 1] IMPLICIT GroupDeviceAttributes             OPTIONAL,
        namedDeviceTypes                [ 2] IMPLICIT NamedDeviceTypes                  OPTIONAL,
        hasLogicalElement               BOOLEAN,
        otherLogicalDeviceList          [ 3] IMPLICIT SEQUENCE OF DeviceID              OPTIONAL,
        deviceModelName                 [ 4] IMPLICIT IA5String (SIZE(0..64))           OPTIONAL,
        deviceOnDeviceMonitorFilter     [ 5] IMPLICIT MonitorFilter                     OPTIONAL,
        deviceOnConnectionMonitorFilter [ 6] IMPLICIT MonitorFilter                     OPTIONAL,
        callOnDeviceMonitorFilter       [ 7] IMPLICIT MonitorFilter                     OPTIONAL,
        callOnConnectionMonitorFilter   [ 8] IMPLICIT MonitorFilter                     OPTIONAL,
        maxDisplays                     [ 9] IMPLICIT INTEGER                           OPTIONAL,
        maxButtons                      [10] IMPLICIT INTEGER                           OPTIONAL,
        maxLamps                        [11] IMPLICIT INTEGER                           OPTIONAL,
        maxRingPatterns                 [12] IMPLICIT INTEGER                           OPTIONAL,
        physDevServList                 [13] IMPLICIT PhysDevServList                   OPTIONAL,
        physDevEvtsList                 [14] IMPLICIT PhysDevEvtsList                   OPTIONAL,
        extensions                      CSTACommonArguments                             OPTIONAL}
```

Parameter Name	***Type***	***Description***
deviceCategory	Enumerated	Specifies the device category (station, ACD device, etc.) of the device being queried. The complete set of possible values is: - ACD - Group - Network Interface (i.e., trunk) - Park ~~- Routeing~~ - Station (default) - Voice Unit - Other
groupDeviceAttributes	Bitmap	Specifies the group device attributes of the device being queried. If a bit is TRUE then the specified attribute is present. The following is the list of bits (multiple bits may be set): - ACD - Hunt - Pick - Other This parameter shall be provided if the deviceCategory is Group, otherwise it shall not be provided.
hasLogicalElement	Boolean	Specifies if the device has a logical element associated with this device identifier. The complete set of possible values is: - FALSE - The device does not have a logical element. - TRUE - The device does have a logical element. The device identifier in the service request should be used with the Get Logical Device Information service to obtain the logical elements characteristics for this device.
deviceOnDeviceMonitorFilter	MonitorFilter	Specifies the complete monitorFilter parameter that the device supports with respect to device-type monitoring. This parameter shall be provided if this form of device-type monitoring is supported, otherwise the parameter shall not be provided. The information in the monitor filter parameters used in the Get Logical Device Information and the Get Physical Device Information services should be the same when the same device identifier is used (assuming that the device identifier has a logical and a physical element).
physDevServList	PhysDevServList	Specifies a list of bitmaps. Each bitmap entry represents a Physical Device service that is supported by the specified device. This includes the following categories of services: Physical Device Feature services This parameter shall be provided if the switching function supports any of these categories of services for this device. If a physical device services bitmap entry is not included in the list, then the service is not supported by the specified device.
physDevEvtsList	PhysDevEvtsList	Specifies a list of bitmaps. Each bitmap entry represents a Physical Device Event that is supported by the specified device. This includes the following categories of events: Physical Device Feature events This parameter shall be provided if the switching function supports any of these categories of events for this device. If a physical device events bitmap entry is not included in the list, then the event is not supported by the specified device.

2.2.1.3 Get Switching Function Capabilities (CSTA ASN.1 only)

Description

The Get Switching Function Capabilities service is used by the computing function to obtain the current set of capabilities for the entire switching function. Here only the provided structures and not the values are shown.

Operation

```
getSwitchingFunctionCapabilities OPERATION ::=
{       ARGUMENT                        GetSwitchingFunctionCapsArgument
        RESULT                          GetSwitchingFunctionCapsResult
        ERRORS                          {universalFailure}
        CODE                            local:203
}
```

Service request

```
GetSwitchingFunctionCapsArgument ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

Service response

```
GetSwitchingFunctionCapsResult ::= SEQUENCE
{       switchingSubDomainName          IA5String (SIZE(0..64)),
        manufacturerName                IA5String (SIZE(0..64)),
        profiles                        Profiles,
        deviceIDFormat                  DeviceIDFormat,
        swDomainFeatures                SwDomainFeatures,
        swAppearanceAddressability      SwAppearanceAddressability,
        swAppearanceTypes               SwAppearanceTypes,
        ignoreUnsupportedParameters     IgnoreUnsupportedParameters,
        callCharacteristicsSupported    [0] IMPLICIT CallCharacteristics                OPTIONAL,
        mediaClassSupport               [1] IMPLICIT MonitorMediaClass                  OPTIONAL,
        numberOfChannels                [2] IMPLICIT NumberOfChannels                   OPTIONAL,
        maxChannelBind                  [3] IMPLICIT MaxChannelBind                     OPTIONAL,
        miscMediaCallCharacteristics    [4] IMPLICIT MiscMediaCallCharacteristics       OPTIONAL,
        connectionRateList              [5] IMPLICIT ConnectionRateList                 OPTIONAL,
        delayToleranceList              [6] IMPLICIT DelayToleranceList                 OPTIONAL,
        pauseTime                       [7] IMPLICIT PauseTime                          OPTIONAL,
        currentTime                     [8] IMPLICIT TimeInfo                           OPTIONAL,
        messageSeqNumbers               [9] IMPLICIT MessageSeqNumbers                  OPTIONAL,
        timeStampMode                   [10] IMPLICIT TimeStampMode                     OPTIONAL,
        securityMode                    [11] IMPLICIT SecurityMode                      OPTIONAL,
        securityFormat                  [12] IMPLICIT SecurityFormat                    OPTIONAL,
        privateDataFormat               [13] IMPLICIT SecurityFormat                    OPTIONAL,
        transAndConfSetup               [14] IMPLICIT TransAndConfSetup                 OPTIONAL,
        monitorFilterItems              [15] IMPLICIT MonitorFilterItems                OPTIONAL,
        miscMonitorCaps                 [16] IMPLICIT MiscMonitorCaps                   OPTIONAL,
        correlatorDataSupported         [17] IMPLICIT BOOLEAN                           OPTIONAL,
        dynamicFeatureSupported         [18] IMPLICIT DynamicFeatureSupported           OPTIONAL,
        aCDModels                       [19] IMPLICIT ACDModels                         OPTIONAL,
        agentLogOnModels                [20] IMPLICIT AgentLogOnModels                  OPTIONAL,
        agentStateModels                [21] IMPLICIT AgentStateModels                  OPTIONAL,
        connectionView                  ConnectionView,
        maxLengthParameters             MaxLengthParameters,
        servEvtsList                    [22] IMPLICIT ServEvtsList                      OPTIONAL,
        privateDataVersionList          [23] IMPLICIT PrivateDataVersionList            OPTIONAL,
        systemStatusTimer               [24] IMPLICIT INTEGER (0..180)                  OPTIONAL,
        simpleThreshold                 [25] IMPLICIT INTEGER                           OPTIONAL,
        filterThreshold                 [26] IMPLICIT FilterThreshold                   OPTIONAL,
        mediaServiceCapsList            [27] IMPLICIT MediaServiceCapsList              OPTIONAL,
        extensions                      CSTACommonArguments                             OPTIONAL}
```

```
MonitorFilterItems ::= SEQUENCE
{       deviceOnDeviceMonitorFilter         [0] IMPLICIT MonitorFilter                  OPTIONAL,
        deviceOnConnectionMonitorFilter     [1] IMPLICIT MonitorFilter                  OPTIONAL,
        callOnDeviceMonitorFilter           [2] IMPLICIT MonitorFilter                  OPTIONAL,
        callOnConnectionMonitorFilter       [3] IMPLICIT MonitorFilter                  OPTIONAL}
```

```
ServEvtsList ::= SEQUENCE
{       capExchangeServList                 [ 0] IMPLICIT CapExchangeServList           OPTIONAL,
        systemServList                      [ 1] IMPLICIT SystemStatusServList          OPTIONAL,
        monitoringServList                  [ 2] IMPLICIT MonitoringServList            OPTIONAL,
        snapshotServList                    [ 3] IMPLICIT SnapshotServList              OPTIONAL,
        callControlServList                 [ 4] IMPLICIT CallControlServList           OPTIONAL,
        callControlEvtsList                 [ 5] IMPLICIT CallControlEvtsList           OPTIONAL,
        callAssociatedServList              [ 6] IMPLICIT CallAssociatedServList        OPTIONAL,
        callAssociatedEvtsList              [ 7] IMPLICIT CallAssociatedEvtsList        OPTIONAL,
        mediaServList                       [ 8] IMPLICIT MediaServList                 OPTIONAL,
        mediaEvtsList                       [ 9] IMPLICIT MediaEvtsList                 OPTIONAL,
        routeingServList                    [10] IMPLICIT RouteingServList              OPTIONAL,
        physDevServList                     [11] IMPLICIT PhysDevServList               OPTIONAL,
        physDevEvtsList                     [12] IMPLICIT PhysDevEvtsList               OPTIONAL,
        logicalServList                     [13] IMPLICIT LogicalServList               OPTIONAL,
        logicalEvtsList                     [14] IMPLICIT LogicalEvtsList               OPTIONAL,
        deviceMaintEvtsList                 [15] IMPLICIT DeviceMaintEvtsList           OPTIONAL,
        iOServicesServList                  [16] IMPLICIT IOServicesServList            OPTIONAL,
        dataCollectionServList              [17] IMPLICIT DataCollectionServList        OPTIONAL,
        voiceUnitServList                   [18] IMPLICIT VoiceUnitServList             OPTIONAL,
        voiceUnitEvtsList                   [19] IMPLICIT VoiceUnitEvtsList             OPTIONAL,
        cdrServList                         [20] IMPLICIT CDRServList                   OPTIONAL,
        vendorSpecificServList              [21] IMPLICIT VendorSpecificServList        OPTIONAL,
        vendorSpecificEvtsList              [22] IMPLICIT VendorSpecificEvtsList        OPTIONAL}
```

```
StatusReportingServList ::= SEQUENCE
{       monitoringServices                  [ 0] IMPLICIT MonitoringServList            OPTIONAL,
        snapshotServices                    [ 1] IMPLICIT SnapshotServList              OPTIONAL,
        systemServices                      [ 2] IMPLICIT SystemStatusServList          OPTIONAL}
```

```
PrivateDataVersionList ::= SEQUENCE OF INTEGER
```

Parameter Name	***Type***	***Description***
switchingSubDomainName	Character (64)	Specifies the name of switching sub-domain ('customer-name') which distinguishes it from other switching sub-domains. If no customer-name is configured in the OpenScape Business Assistant the value "not configured" is given as default.
manufacturerName	Characters (64)	Specifies the name of the manufacturer of the switching sub-domain. Always "Unify" is returned.
profiles	Bitmap	Specifies the CSTA Profiles supported by the switching function. The following is the list of the possible profiles (multiple bits may be set).
deviceIDFormat	Bitmap	Specifies the types of device ID formats supported by the switching function. If a bit is TRUE, then the specified format is used by the switching function. Note that the Diallable Digits format with the 0-9 characters shall be supported by the switching function.
swDomainFeatures	Bitmap	Specifies which features are supported by the switching function. If a bit is TRUE, then the specified feature is supported.
swAppearanceAddressability	Bitmap	Specifies what types of appearance addressability is available within the switching sub-domain.
swAppearanceTypes	Bitmap	Specifies what types of appearances are available within the switching sub-domain
ignoreUnsupportedParameters	Enumerated	Specifies how the switching function handles unsupported optional parameters in service requests.
mediaClassSupport	Bitmap	Specifies the media class of calls the switching sub-domain can support. If a bit is TRUE then the specified type of call can be present within the switching function.
numberOfChannels	Value	Specifies the highest number of available channels at a given device within the switching sub-domain. If the parameter is not present, the number of channels at a device is not known but is one or greater.
currentTime	TimeInfo	Specifies the current date and time of the switching function.
timeStampMode	Bitmap	Specifies when the switching function provides timestamp information (via the security parameter). If a bit is TRUE, then the mode is supported.
privateDataFormat	Bitmap	Specifies the format(s) of the privateData information supported by the switching function.
transAndConfSetup	Bitmap	Specifies the different ways that the switching function can set up for a conference and/or transfer. (Note that if this parameter is not present, then the switching function can only set up transfers and conferences through the Consultation Call service.) If the bit is TRUE, then the specified way to setup a conference or transfer is supported by at least one device in the switching sub-domain.
deviceOnDeviceMonitorFilter	MonitorFilter	Specifies the complete monitorFilter parameter that is supported by the switching function when the monitorObject is a device and the monitorType is a device. Each bitmap entry represents an event that is supported by at least one of the devices in the switching function. This parameter shall be provided if this form of device-type monitoring is supported by at least one of the devices in the switching function, otherwise it shall not be provided.
miscMonitorCaps	Bitmap	Specifies the special types of monitoring capabilities that are present within the switching sub-domain. If a bit is TRUE then the monitoring capability is present within the switching sub-domain. ACD devices).
acdModels	Bitmap	Specifies the types of ACD models that are supported by the switching function. If a bit is TRUE, then the specified ACD model is supported by the switching function. Note that if more than one type of ACD model is present in the switching function, then the Get Logical Device Information service shall be used to determine the particular ACD models supported by for each ACD device or ACD group by the switching function.
agentLogOnModels	Bitmap	Specifies the types of agent log on models that are supported by the switching function. If a bit is TRUE, then the specified agent log on model is supported by the switching function Note that if more than one type of model is present in the switching function, then the Get Logical Device Information service shall be used to determine the particular Log On models supported for each device which is or can be associated with an agent.

agentStateModels	Bitmap	Specifies the types of agent models that are supported by the switching function. If a bit is TRUE, then the specified agent model is supported by the switching function. Note that if more than one type of model is present in the switching function, then the Get Logical Device Information service shall be used to determine the particular Agent models supported for each ACD device or ACD group by the switching function.
connectionView	Enumerated	Specifies the meaning of the primary and secondary old call parameters in the Conferenced and Transferred events Refer to the descriptions of the Conferenced and the Transferred events for more information.
maxLengthParameters	List of Values	Each value is the switching function's maximum length (in octets/characters) for the corresponding parameters and parameter types. The computing function should not send larger data or the service request will be rejected. The following list provides the different parameters and parameter types for which a maximum value is provided. The number in parenthesis specifies the maximum possible length. - AccountInfo parameter type: (32) - AuthCode parameter type: (32) - AgentID parameter type: (32) - AgentPassword parameter type: (32) - callID in the ConnectionID parameter type: (8) - CorrelatorData parameter type: (32) - CSTAPrivateData parameter type: (any value) - Device Identifiers parameter type: (64) - UserData parameter type: (256) - buttonLabel parameters: (64) - lampLabel parameters: (64) - charsToBeSent parameter: (64) If any of the above values is zero, then the parameter or parameter type is not supported.
servEvtsList	List	Specifies a list of capability bitmap parameter types corresponding to categories of services. Each bitmap entry in the lists represents a service or event that is supported by the switching function. This includes the following categories of services/events: - systemStatusServList (O) SystemStatusServList - monitoringServList (O) MonitoringServList - snapshotServList (O) SnapshotServList - callControlServList (O) CallControlServList - callControlEvtsList (O) CallControlEvtsList - callAssociatedServList (O) CallAssociatedServList - callAssociatedEvtsList (O) CallAssociatedEvtsList - physServList (O) PhysServList - physEvtsList (O) PhysEvtsList - logicalServList (O) LogicalServList - logicalEvtsList (O) LogicalEvtsList - deviceMaintEvtsList (O) DeviceMaintEvtsLis - ioServicesServList (O) IOServicesServList - voiceUnitServList (O) VoiceUnitServList - voiceUnitEvtsList (O) VoiceUnitServList - cdrServList (O) CDRServList - vendorSpecificServList (O) VendorSpecificServList This parameter shall be provided if the switching function supports any of these categories of services/events. If a list entry is not included in the list, then the corresponding category of services/events is not supported by the switching function. The bitmap parameter types are described in Annex C.

2.2.1.4 GetCSTAFeatures (CSTA XML only)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed6"
            xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed6"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xsd:annotation>
        <xsd:documentation>CSTA-get-csta-features</xsd:documentation>
    </xsd:annotation>

    <xsd:include schemaLocation="status-reporting.xsd"/>
```

```
    <xsd:include schemaLocation="extension-types.xsd"/>
    <xsd:include schemaLocation="capability-exchange.xsd"/>

    <xsd:element name="GetCSTAFeatures">
        <xsd:complexType>
            <xsd:sequence>
                    <xsd:element ref="csta:extensions" minOccurs="0"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="GetCSTAFeaturesResponse">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="supportedServices">
                    <xsd:complexType>
                        <xsd:sequence>
                        <xsd:element name="capExchangeServList" type="csta:CapExchangeServList" minOccurs="0"/>
                        <xsd:element name="systemStatServList" type="csta:SystemStatusServList" minOccurs="0"/>
                        <xsd:element name="monitoringServList" type="csta:MonitoringServList" minOccurs="0"/>
                        <xsd:element name="snapshotServList" type="csta:SnapshotServList" minOccurs="0"/>
                        <xsd:element name="callControlServList" type="csta:CallControlServList" minOccurs="0"/>
                        <xsd:element name="callAssociatedServList" type="csta:CallAssociatedServList" minOccurs="0"/>
                        <xsd:element name="mediaServList" type="csta:MediaServList" minOccurs="0"/>
                        <xsd:element name="routeingServList" type="csta:RouteingServList" minOccurs="0"/>
                        <xsd:element name="physDevServList" type="csta:PhysDevServList" minOccurs="0"/>
                        <xsd:element name="logicalServList" type="csta:LogicalServList" minOccurs="0"/>
                        <xsd:element name="iOServicesServList" type="csta:IOServicesServList" minOccurs="0"/>
                        <xsd:element name="dataCollectionServList" type="csta:DataCollectionServList" minOccurs="0"/>
                        <xsd:element name="voiceUnitServList" type="csta:VoiceUnitServList" minOccurs="0"/>
                        <xsd:element name="cDRServList" type="csta:CDRServList" minOccurs="0"/>
                        <xsd:element name="vendorSpecificServList" type="csta:VendorSpecificServList" minOccurs="0"/>
                        <xsd:element name="locationServicesServList" type="csta:LocationServicesServList" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="supportedEvents" minOccurs="0">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="callControlEvtsList" type="csta:CallControlEvtsList" minOccurs="0"/>
                        <xsd:element name="callAssociatedEvtsList" type="csta:CallAssociatedEvtsList" minOccurs="0"/>
                        <xsd:element name="mediaEvtsList" type="csta:MediaEvtsList" minOccurs="0"/>
                        <xsd:element name="physDevEvtsList" type="csta:PhysDevEvtsList" minOccurs="0"/>
                    <xsd:element name="logicalEvtsList" type="csta:LogicalEvtsList" minOccurs="0"/>
                        <xsd:element name="deviceMaintEvtsList" type="csta:DeviceMaintEvtsList" minOccurs="0"/>
                        <xsd:element name="voiceUnitEvtsList" type="csta:VoiceUnitEvtsList" minOccurs="0"/>
                        <xsd:element name="vendorSpecificEvtsList" type="csta:VendorSpecificEvtsList" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element ref="csta:extensions" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

2.2.1.5 Get Switching Function Devices

The Get Switching Function Devices service is used by the computing function to obtain the current set of devices in the application working domain along with their associated device categories and associated device names.

Operation

```
getSwitchingFunctionDevices OPERATION ::=
{       ARGUMENT GetSwitchingFunctionDevicesArgument
        RESULT GetSwitchingFunctionDevicesResult
        ERRORS {universalFailure}
        CODE local:204
}
```

Service Request

```
GetSwitchingFunctionDevicesArgument ::= SEQUENCE
{       requestedDeviceID               DeviceID                        OPTIONAL,
        requestedDeviceCategory         ReqDeviceCategory               OPTIONAL,
        extensions                      CSTACommonArguments             OPTIONAL
}
ReqDeviceCategory ::= ENUMERATED
{       acd                             ( 0),
        groupACD                        ( 1),
        groupHunt                       ( 2),
        groupPick                       ( 3),
        groupOther                      ( 4),
        networkInterface                ( 5),
        park                            ( 6),
        routeingDevice                  ( 7),
        station                         ( 8),
        voiceUnit                       ( 9),
        other                           (10),
        conference                      (11),           -- import from ECMA-269 9th edition
        dtmfIV                          (12),
        genericIV                       (13),           -- import from ECMA-269 9th edition
        groupUser                       (14),
        listenerIV                      (15),
        messageIV                       (16),
        promptIV                        (17),
        promptQueue                     (18)
}
```

Note:

- The ReqDeviceCategory ACD and groupACD will ask for the configured UCD-Groups.
- The ReqDeviceCategory network interface will ask for the CO-Trunk interfaces.
- The ReqDeviceCategory station will ask for the extensions.
- The ReqDeviceCategory conference will ask for the conference-extensions of UC-Suite.
- The ReqDeviceCategory genericIV will ask for the autoattendant-extensions of UC-Suite.

Parameter Name	*Type*	*Description*
requestedDeviceID	DeviceID (O)	Specifies the device identifier of the device being queried. If this parameter is not present, the switching function returns a list of all devices (of the requestedDeviceCategory, if that parameter is provided) in the switching sub-domain.
requestedDeviceCategory	Enumerated (O)	Specifies that only devices of the requested category be provided. If this parameter is not present, the switching function will return a list of all devices (or just the requested device) in the switching function. The complete set of possible values is: • ACD • Group (ACD) • Group (Hunt) • Group (Pick) • Group (Other) • Network Interface • Park • Routeing Device • Station • Voice Unit • Other • conference -- import from ECMA-269 9th edition • ~~dtmfIV~~ • genericIV -- import from ECMA-269 9th edition • ~~groupUser~~ • ~~listenerIV (15),~~ • ~~messageIV~~ • ~~promptIV~~ • ~~promptQueue~~

Service Response

```
GetSwitchingFunctionDevicesResult ::= SEQUENCE
{       serviceCrossRefID               ServiceCrossRefID,
        extensions                      CSTACommonArguments             OPTIONAL
}
```

2.2.2 Switching Function Devices

The Switching Function Devices service is used by the switching function to provide a list of devices in the application working domain. This service is generated as a result of the Get Switching Function Devices service.

Operation

```
switchingFunctionDevices OPERATION ::=
{       ARGUMENT SwitchingFunctionDevicesArgument
        ERRORS {universalFailure}
        CODE local:205
}
```

Service Request

```
SwitchingFunctionDevicesArgument ::= SEQUENCE
{       serviceCrossRefID               ServiceCrossRefID,
        segmentID                       INTEGER                 OPTIONAL,
        lastSegment                     BOOLEAN,
        deviceList                      DeviceList,
        extensions                      CSTACommonArguments     OPTIONAL
}
```

```
DeviceList ::= SEQUENCE OF SEQUENCE
{       deviceID                    DeviceID,
        deviceCategory              [0] IMPLICIT DeviceCategory DEFAULT station,
        namedDeviceTypes            [1] IMPLICIT NamedDeviceTypes       OPTIONAL,
        deviceAttributes            DeviceAttributes                    OPTIONAL,
        deviceModelName             IA5String (SIZE(0..64))             OPTIONAL
}
```

Parameter Name	***Type***	***Description***
serviceCrossRefID	ServiceCrossRefID (M)	Specifies the cross reference used to associate the Switching Function Devices service request to the Get Switching Function Devices service request
segmentID	Value O	Specifies the segment number of this message. Each successive segment number in the sequence increments the segmentID by one.
lastSegment	Boolean M	Specifies if this segment is the last one associated with the serviceCrossRefID. The complete set of possible values is: • TRUE - Indicates that this is the last segment • FALSE - Indicates that this is not the last segment in the sequence.
deviceList	List of Structures M	Specifies the list of device Identifiers representing the devices that can be controlled and/or observed. It includes the following components for each device in the list: • deviceID (M) DeviceID - Specifies a Device Identifier associated with the entry in the list. • deviceCategory (O) Enumerated - Specifies the device category associated with the entry in the list. The complete set of possible values is: • ACD • Group • Network Interface (e.g. trunk, CO Line) • Park ~~• Routeing~~ • Station (default) • Voice Unit • Other • genericInteractiveVoice -- import from ECMA-269 9th edition • conference -- import from ECMA-269 9th edition

2.3 System Services

~~2.3.1 Registration Services~~

~~*2.3.1.1 Change System Status Filter*~~

- Not supported

~~*2.3.1.2 System Register*~~

- Not supported

~~*2.3.1.3 System Register Abort*~~

- Not supported

~~*2.3.1.4 System Register Cancel*~~

- Not supported

2.3.2 Services

2.3.2.1 Request System Status

Description

The Request System Status service is used by the computing function or switching function to obtain (i.e. query) the system status of its peer function. With this request a Client can request OpenScape Business to send its systemstatus back in the result.

Operation

```
requestSystemStatus OPERATION ::=
{       ARGUMENT                        requestSystemStatusArg
        RESULT                          requestSystemStatusRes
        ERRORS                          {universalFailure}
        CODE                            local: 211
}
```

Service request

```
requestSystemStatusArg ::= SEQUENCE
{       sysStatRegisterID               SysStatRegisterID               OPTIONAL,
        security                        CSTASecuritydata                OPTIONAL,
        privateData                     CSTAPrivateData                 OPTIONAL
}
```

Service response

```
requestSystemStatusRes ::= CHOICE
{       systemStatus                    Enumerated
        security                        CSTASecuritydata,
        privateData                     CSTAPrivateData
```

Parameter Name	*Type*	*Description*
systemStatus	Enumerated	Specifies the status of the function issuing the service request. The complete set of possible values is: - Initializing - Enabled - Normal - Messages Lost - Disabled - Partially Disabled - Overload Imminent - Overload Reached - Overload Relieved

Note:
(Compare ECMA-269, chapter 12.2.28) With OpenScape Business a systemStatus 'Partially Disabled' means, that one or multiple nodes (but not all) of a networked system of OpenScape Business are temporarily not reachable (maybe because of maintenance or issues in the IP-network).

2.3.2.2 System Status

Description

The System Status service is used by the computing function or switching function to report its system status to its peer function. The indicated status may or may not have changed since the last System Status request was issued.

Note
The service systemStatus is automatically sent after ACSE negotiation. With OpenScape Business a SystemStatus-request from switching function has to be replied by computing function, else the OpenScape Business closes the CSTA-Link.

Operation

```
systemStatus OPERATION ::=
{       ARGUMENT                        SystemStatusArg
        RESULT                          SystemStatusRes
        ERRORS                          {universalFailure}
        CODE                            local: 211
}
```

Service request

```
SystemStatusArg ::= SEQUENCE
{       sysStatRegisterID               SysStatRegisterID               OPTIONAL,
        systemStatus                    SystemStatus,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Service response

```
SystemStatusRes ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

Parameter Name	***Type***	***Description***
~~sysStatRegisterID~~	~~SysStatRegisterID~~	Specifies the system registration identifier associated with the system registration for this request. This parameter is mandatory if the switching function is issuing the request and supports system registration, and shall not be provided otherwise.
systemStatus	Enumerated	Specifies the status of the function issuing the service request. The complete set of possible values is: - Initializing - Enabled - Normal - Messages Lost - Disabled - Partially Disabled - Overload Imminent - Overload Reached - Overload Relieved

~~*2.3.2.3 Switching Function Capabilities Changed*~~
- Not supported

~~*2.3.2.4 Switching Function Devices Changed*~~
- Not supported

2.4 Monitoring Services

2.4.1 Services

Note
The maximum amount of devices that can be monitored at the same time on OpenScape Business is stated in chapter 2.18.1 OpenScape Business Parameter Constraints. There is no event report on non-voice calls. See also 2.18.3 List of monitorable OpenScape Business Devices.

With OpenScape Business the monitoring of S_0-Devices is MSN driven. That means the monitor could be started for the MSN and not the S_0-Devices itself. Because a single S_0-Device could use different MSN, a simple configuration should be used. This means, that one unique MSN should be used per S_0-Devices.

2.4.1.1 Change Monitor Filter

Description

The Change Monitor Filter service is used to modify the set of event reports that are filtered out (not sent) over an existing monitor. The new set of filtered out (not sent) event reports may be listed in the service acknowledgement.

Operation

```
changeMonitorFilter    OPERATION ::=
{       ARGUMENT                        ChangeMonitorFilterArgument
        RESULT                          ChangeMonitorFilterResult
        ERRORS                          {universalFailure}
        CODE                            local: 72
}
```

Service request

```
ChangeMonitorFilterArgument ::=
      SEQUENCE
      {crossRefIdentifier               MonitorCrossRefID,
       requestedFilterList              MonitorFilter,
       extensions                       CSTACommonArguments            OPTIONAL}
```

Parameter Name	*Type*	*Description*
crossRefIdentifer	MonitorCrossRefID	This indicates the monitor for which to change the filter.
requestedFilterList	MonitorFilter	This parameter specifies the requested set of events to be filtered out (not sent) by the server. It is a bitmap of all events defined in this Standard.

Service response

```
ChangeMonitorFilterResult ::=
      SEQUENCE
      {actualFilterList                 MonitorFilter                  OPTIONAL,
       extensions                       CSTACommonArguments            OPTIONAL}
```

Parameter Name	*Type*	*Description*
actualFilterList	MonitorFilter	This parameter specifies the actual set of events that will be filtered out (not sent) by the server. It is a bitmap of all events defined in this Standard. The actual events filtered from the requestedFilterList parameter on the service request. This parameter is optional if the actualFilterList is the same as the requestedFilterList parameter on the service request, otherwise it is mandatory.

2.4.1.2 Monitor Start

Description

The Monitor Start service initiates event reports for a device.
The server starts a monitor, allocates a Monitor Cross Reference Identifier that uniquely identifies the monitor, and then positively acknowledges the request. All activities satisfying the filter provided (for example: call, feature, agent, private) trigger events which are delivered as a stream of event reports to the server. Each event contains the Monitor Cross Reference Identifier that correlates the event back to the Monitor Start service that established the monitor.

These event reports cease after the switching function terminates the monitor. The switching function shall terminate the monitor if the monitorObject ceases to exist, or if the monitorObject leaves the switching sub-domain. There may be other conditions that cause the server to terminate the monitor. Once the monitor is terminated, the monitor cross reference ID is no longer valid.

Note

The MonitorStart request is positively acknowledged for devices currently being out-of-service. A following SnapshotDevice request on that device will be negatively acknowledged with a resourceOutOfService error.

Operation

```
monitorStart    OPERATION ::=
{       ARGUMENT                        MonitorStartArgument
        RESULT                          MonitorStartResult
        ERRORS                          {universalFailure}
        CODE                            local: 71
}
```

Service request

```
MonitorStartArgument ::= SEQUENCE
{       monitorObject                   MonitorObject,
        requestedMonitorFilter          [0] IMPLICIT MonitorFilter              OPTIONAL,
        monitorType                     MonitorType                             OPTIONAL,
        requestedMonitorMediaClass      [1] IMPLICIT MonitorMediaClass          OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
monitorObject	Choice Structure	Specifies the monitor object of a call or device to be monitored. This shall be one of the following choices: ~~- call (ConnectionID) - Specifies a call (connection) object.~~ - device (DeviceID) - Specifies a device object.
requestedMonitorFilter	MonitorFilter	This parameter specifies the requested set of events to be filtered out (not sent) by the switching function. It is a bitmap of all events defined in this standard. If this parameter is not provided (or if the parameter is not supported by the switching function), then it shall mean that no filtering of events is requested (all events are requested).

Service response

```
MonitorStartResult ::= SEQUENCE
{       crossRefIdentifier              MonitorCrossRefID,
        actualmonitorFilter             [0] IMPLICIT MonitorFilter              OPTIONAL,
        actualMonitorMediaClass         [1] IMPLICIT MonitorMediaClass          OPTIONAL,
        monitorExistingCalls            BOOLEAN                                 OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	Type	Description
crossRefIdentifer	MonitorCrossRefID	This indicates a value that is unique within the association for the duration of the monitor and that can be used to relate subsequent events to the monitor request that initiated them. It shall also allow correlating Monitor Stop and subsequent Change Monitor Filter services with the original Monitor Start service on which they act.
actualMonitorFilter	MonitorFilter	This parameter specifies the actual set of events that will be filtered (not sent) by the switching function. It is a bitmap of all events defined in this standard. The actual events filtered out may differ from the filterList parameter on the service request If this parameter is supported by the switching function, it may be omitted if the requested and actual monitor filters are the same, otherwise it shall be provided. If the parameter is not supported by the switching function, then the switching function does not filter events and all events supported (as indicated by the capability exchange services) shall be sent for this monitor.

2.4.1.3 Monitor Stop

Description

The Monitor Stop service is used to cancel a previously initiated Monitor Start service.
The Monitor Stop service can be issued by a function to terminate or signal the termination of a corresponding Monitor Start service. In OpenScape Business the system can invoke the MonitorStop, in case that the monitored device had been removed from configuration.
A positive acknowledgement to the service request indicates that the Cross Reference ID used by the Monitor Start service has become invalid.

Note:
If the monitored device does no longer exist due to configuration changes, the switching function will send a MonitorStop-service to the computing function.

Operation

```
monitorStop     OPERATION ::=
{       ARGUMENT                        MonitorStopArgument
        RESULT                          MonitorStopResult
        ERRORS                          {universalFailure}
        CODE                            local: 73
}
```

Service request

```
MonitorStopArgument ::=
      SEQUENCE
      {crossRefIdentifier               MonitorCrossRefID,
       extensions                       CSTACommonArguments              OPTIONAL}
```

Parameter Name	Type	Description
crossRefIdentifer	MonitorCrossRefID	This indicates the Cross Reference Identifier provided in the original Monitor Start service positive acknowledgement.

Service response

```
MonitorStopResult ::=
      CHOICE
      {extensions                       CSTACommonArguments,
       noData                           NULL}
```

2.5 Snapshot Services

2.5.1 Services

2.5.1.1 Snapshot Call

Description

The Snapshot Call service provides information about the devices participating in a specified call. The information provided includes device identifiers, their connections in the call, and local connection states of the devices in the call as well as call related information.
Information that applies to the entire call is provided in the Snapshot Call positive response.

Operation

```
snapshotCall  OPERATION ::=
{       ARGUMENT                        SnapshotCallArgument
        RESULT                          SnapshotCallResult
        ERRORS                          {universalFailure}
        CODE                            local: 75
}
```

Service request

```
SnapshotCallArgument ::=
      SEQUENCE
      {snapshotObject                   ConnectionID,
       extensions                       CSTACommonArguments                    OPTIONAL}
```

Parameter Name	*Type*	*Description*
snapshotObject	ConnectionID	This indicates the connectionID of the call to be snapshot.

Service response

```
SnapshotCallResult ::=
      SEQUENCE
      {CHOICE
            {serviceCrossRefID          [0] IMPLICIT ServiceCrossRefID,
             snapshotData               SnapshotCallData},
        mediaCallCharacteristics        MediaCallCharacteristics               OPTIONAL,
        callCharacteristics             CallCharacteristics                    OPTIONAL,
        callingDevice                   CallingDeviceID                        OPTIONAL,
        calledDevice                    CalledDeviceID                         OPTIONAL,
        associatedCallingDeviceID       AssociatedCallingDeviceID              OPTIONAL,
        associatedCalledDeviceID        AssociatedCalledDeviceID               OPTIONAL,
        correlatorData                  [1] IMPLICIT CorrelatorData            OPTIONAL,
        callLinkageData                 [2] IMPLICIT CallLinkageData           OPTIONAL,
        extensions                      CSTACommonArguments                    OPTIONAL}
```

Parameter Name	Type	Description
snapshotData	List of Structures	Specifies information for each endpoint in a call. This parameter is mandatory if the switching function is providing all of the response information in this message. This parameter shall not be provided if the switching function is providing the snapshot information using the Snapshot CallData Service. The supported set of possible information is: - deviceOnCall (M) DeviceID - Of a device involved with the endpoint. - connectionIdentifer (C) ConnectionID - For the endpoint. This is mandatory if the endpoint is in the switching sub-domain, otherwise it is optional. - localConnectionState (O) LocalConnectionState - For the endpoint.
callLinkageData	CallLinkageData	Specifies the global call data and thread data associated with the call. This parameter is mandatory if the switching function supports the call linkage feature otherwise it shall not be provided.

2.5.1.2 Snapshot Device

Description

The Snapshot Device service provides information about calls associated with a given device. The information provided identifies each call the device is participating in and the local connection state of the device in that call.
The switching function shall provide the response information using one of two possible mechanisms (as indicated by the capability exchange services): either in the Snapshot Device positive acknowledgement or in one or more messages using the Snapshot DeviceData Service. Note that both mechanisms cannot be used at the same time.
If the switching function supports the Dynamic Feature Availability option (as indicated through the capabilities exchange services), then for each connection at the device, this service provides the list of permitted call control services.

Note

The MonitorStart request is positively acknowledged for devices currently being out-of-service. A SnapshotDevice request on a device being out-of-service will be negatively acknowledged with a resourceOutOfService error.
For incoming calls on a busy device, the call will be not noticed before queued or alerting.

Operation

```
snapshotDevice  OPERATION ::=
{       ARGUMENT                        SnapshotDeviceArgument
        RESULT                          SnapshotDeviceResult
        ERRORS                          {universalFailure}
        CODE                            local: 74
}
```

Service request

```
SnapshotDeviceArgument ::=
      SEQUENCE
      {snapshotObject                   DeviceID,
       extensions                       CSTACommonArguments                    OPTIONAL}
```

Parameter Name	Type	Description
snapshotObject	DeviceID	This indicates the deviceID of the device to be snapshot.

Service response

```
SnapshotDeviceResult ::=
    SEQUENCE
    {CHOICE
        {serviceCrossRefID          ServiceCrossRefID,
         snapshotData               SnapshotDeviceData},
      extensions                    CSTACommonArguments                OPTIONAL}
```

Parameter Name	*Type*	*Description*
snapshotData	List of Structures	Specifies information for each call at a device. This parameter is mandatory if the switching function is providing all of the response information in this message. This parameter shall not be provided if the switching function is providing the response information using the Snapshot DeviceData Service. This complete set of information is: - connectionIdentifier (M) ConnectionID - localCallState (M) Choice Structure - This shall be one of the following choices: - compoundCallState (List of LocalConnectionStates) - This consists of a sequence of local connection states. - simpleCallState (SimpleCallState) - the simple call state. - unknown

~~2.5.1.3 Snapshot CallData~~

Not supported

~~2.5.1.4 Snapshot DeviceData~~

Not supported

2.6 Call Control Services and Events

2.6.1 Services

Note

- Call processing actions of CSTA services are leaned on the call processing capabilities of the device or the feature specifications within ETSI / ECMA. In doubt the set of features available via CSTA services is smaller than via device (e.g. speed extend is not possible via CSTA).
- It is recommended to provide in the CallControl-services connections consisting of both device- and call-IDs.

This is an overview of the supported Call Control Services for SIP-Endpoints:

Service	for SIP-Endpoint (active part)	to SIP-Endpoint (passive part)
Alternate Call	X	X
Answer Call	depends on Device capabilities	X
Clear Connection	X	X
ConferenceCall	-	X
Consultation Call	X	X
Deflect Call	X	X
Directed Pickup Call	X	X
Group Pickup Call	X	X
Hold Call	X	X
Make Call	X	X
ParkCall	X	X
Reconnect Call	X	X
Retrieve Call	X	X
Single Step Transfer	X	X
Transfer Call	X	X

This is an overview of the supported Call Control Services for S_0-Devices:

Service	for S_0-Device (active part)	to S_0-Device (passive part)
Alternate Call	-	X
Answer Call	-	X
Clear Connection	X	X
ConferenceCall	-	X
Consultation Call	-	X
Deflect Call	X	X
Directed Pickup Call	-	X
Group Pickup Call	-	X
Hold Call	-	X
Make Call	-	X
ParkCall	-	X
Reconnect Call	-	X
Retrieve Call	-	X
Single Step Transfer	X	X
Transfer Call	-	X

~~2.6.1.1 Accept Call~~

Not supported

2.6.1.2 Alternate Call

Description

The Alternate Call service places an existing active call on hold and then retrieves a previously held call. This service is also used to place an active call on hold and then connect to an alerting or queued call at the same device (i.e., to answer a call-waiting call).

Note

- The ConnectionIDs may be constructed as callID, deviceID or both, but at least one of them (activeCall, heldCall) has to include a DeviceID.
- This service is not supported to answer queued or alerting calls (e.g. camp on). For this the AnswerCall service has to be used.

Operation

```
alternateCall                   OPERATION ::=
{       ARGUMENT                AlternateCallArgument
        RESULT                  AlternateCallResult
        ERRORS                  {universalFailure}
        CODE                    local: 1
}
```

Service request

```
AlternateCallArgument ::= SEQUENCE
{       heldCall                ConnectionID,
        activeCall              ConnectionID,
        connectionReservation   BOOLEAN                 OPTIONAL,
        consultOptions          ConsultOptions          DEFAULT unrestricted,
        extensions              CSTACommonArguments     OPTIONAL
}
```

Parameter Name	*Type*	*Description*
heldCall	ConnectionID	Specifies the held connection for the alternating device.
activeCall	ConnectionID	Specifies the active connection for the alternating device.

Service response

```
AlternateCallResult ::=         CHOICE
{       extensions              CSTACommonArguments,
        noData                  NULL}
```

2.6.1.3 Answer Call

Description

The Answer Call service connects an alerting or queued call.
This service is typically associated with devices that have attached speakerphone units and headset telephones to connect to a call via hands-free operation. For example, when the call is answered, one of the following actions may occur:

- If the specified device has a speaker and a microphone, the speaker and microphone are turned on.
- If the specified device has a headset, the headset is turned on.

For example the AnswerCall-Service can be executed on an OpenStage 40, but not on an OpenStage 15, as this model has no microphone for handsfree operation. SIP devices should support the auto-answer option in the alert-info.

Note
The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID).

Operation

```
answerCall                      OPERATION ::=
{       ARGUMENT                AnswerCallArgument
        RESULT                  AnswerCallResult
        ERRORS                  {universalFailure}
        CODE                    local: 2
}
```

Service request

```
AnswerCallArgument ::=
      SEQUENCE
      {     callToBeAnswered          ConnectionID,
            correlatorData            CorrelatorData                OPTIONAL,
            userData                  UserData                      OPTIONAL,
            extensions                CSTACommonArguments           OPTIONAL}
```

Parameter Name	*Type*	*Description*
callToBeAnswered	ConnectionID	Specifies the connection to be answered.

Service response

```
AnswerCallResult ::=
      CHOICE
      {extensions                     CSTACommonArguments,
       noData                         NULL}
```

~~2.6.1.4 Call Back call-related~~
Not supported

~~2.6.1.5 Call Back message call-related~~
Not supported

~~2.6.1.6 Camp On Call~~
Not supported

~~2.6.1.7 Clear Call~~
Not supported

2.6.1.8 Clear Connection

Description

The Clear Connection service releases a specific device from a call. In the case of a two-party call, this may result in the call being torn down. In the case of a conference call, this results in the specific party being removed from the conference.
The Connection ID provided in the request is released.

Note
The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID). If a CallID is provided, only that connection will be cleared. No transfer is executed.

Using ClearConnection as a transfer might work for HFA and TDM but not for SIP. For application it is requested to use the Transfer Call service instead.

Operation

```
clearConnection OPERATION ::=
{       ARGUMENT                        ClearConnectionArgument
        RESULT                          ClearConnectionResult
        ERRORS                          {universalFailure}
        CODE                            local: 5
}
```

Service request

```
ClearConnectionArgument ::=
      SEQUENCE
      {connectionToBeCleared            ConnectionID,
       correlatorData                   CorrelatorData                  OPTIONAL,
       userData                         UserData                        OPTIONAL,
       reason                           EventCause                      OPTIONAL,
       extensions                       CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
connectionToBeCleared	ConnectionID	Specifies the connection to be cleared.

Service response

```
ClearConnectionResult ::=
      CHOICE
      {extensions                       CSTACommonArguments,
       noData                           NULL}
```

2.6.1.9 Conference Call

Description

The Conference Call service provides a conference of an existing held call and another active call at a conferencing device.
The two calls are merged into a single call and the two connections at the conferencing device are resolved into a single connection. The Connection IDs formerly associated with the conferenced connections are released and a new Connection ID for the resulting connection is created.
The existing held call may consist of two or more devices.

Note
The ConnectionIDs may be constructed as callID, deviceID or both, but at least one of them (activeCall, heldCall) has to include a DeviceID.

Operation

```
conferenceCall  OPERATION ::=
{       ARGUMENT                        ConferenceCallArgument
        RESULT                          ConferenceCallResult
        ERRORS                          {universalFailure}
        CODE                            local: 6
}
```

Service request

```
ConferenceCallArgument ::=
      SEQUENCE
      {     heldCall                    ConnectionID,
            activeCall                  ConnectionID,
            extensions                  CSTACommonArguments             OPTIONAL}
```

Parameter Name	Type	Description
heldCall	ConnectionID	Specifies the held connection.
activeCall	ConnectionID	Specifies the active connection.

Service response

```
ConferenceCallResult ::=
SEQUENCE
{       conferenceCall                  ConnectionID,
        connections                     [0] IMPLICIT ConnectionList                     OPTIONAL,
        conferenceCallInfo              [1] IMPLICIT ConnectionInformation              OPTIONAL,
        extensions                      CSTACommonArguments                             OPTIONAL
}
```

Parameter Name	Type	Description
conferenceCall	ConnectionID	Specifies the resulting connection to the new call. The ConnectionID shall have the CallID of the resulting conference call and the DeviceID of the conferencing device.

2.6.1.10 Consultation Call

Description

The Consultation Call service places an existing active call at a device on hold and initiates a new call from the same device. The existing active call may include two or more devices.

Note

The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID).
In case of consultation with an empty consultedDevice the dialling has to be done proprietarily with the MakeCall service as opposed to the ECMA mechanism via the DialDigits service[1]. This mechanism does not work for SIP.

Operation

```
consultationCall OPERATION ::=
{       ARGUMENT                        ConsultationCallArgument
        RESULT                          ConsultationCallResult
        ERRORS                          {universalFailure}
        CODE                            local: 7
}
```

Service request

```
ConsultationCallArgument ::= SEQUENCE
{       existingCall                    [0] IMPLICIT ConnectionID,
        consultedDevice                 [1] DeviceID,
        connectionReservation           [2] IMPLICIT BOOLEAN                            OPTIONAL,
        accountCode                     [3] IMPLICIT AccountInfo                        OPTIONAL,
        authCode                        [4] IMPLICIT AuthCode                           OPTIONAL,
        correlatorData                  [5] IMPLICIT CorrelatorData                     OPTIONAL,
        userData                        [6] IMPLICIT UserData                           OPTIONAL,
        callCharacteristics             [7] IMPLICIT CallCharacteristics                OPTIONAL,
        mediaCallCharacteristics        [8] IMPLICIT MediaCallCharacteristics           OPTIONAL,
        callingConnectionInfo           [9] IMPLICIT ConnectionInformation              OPTIONAL,
        consultOptions                  [10] IMPLICIT ConsultOptions            DEFAULT unrestricted,
        subjectOfCall                   [11]IMPLICIT SubjectOfCall                      OPTIONAL,
        languagePreferences             [12] IMPLICIT LanguagePreferences               OPTIONAL,
        extensions                      CSTACommonArguments                             OPTIONAL
}
```

[1] ECMA-269, chap. 17.1.10.3.4 #6

Parameter Name	Type	Description
existingCall	ConnectionID	Specifies the active connection.
consultedDevice	DeviceID	Specifies the device to be consulted.

Service response

```
ConsultationCallResult ::= SEQUENCE
{       initiatedCall                   ConnectionID,
        mediaCallCharacteristics        [0] IMPLICIT MediaCallCharacteristics       OPTIONAL,
                                        -- shall be provided, if the MediaCallCharacteristics have been
                                        -- adjusted
        initiatedCallInfo               [1] IMPLICIT ConnectionInformation          OPTIONAL,
        extensions                      CSTACommonArguments                         OPTIONAL}
```

Parameter Name	Type	Description
initiatedCall	ConnectionID	Specifies the initial connection to the new call. The ConnectionID shall have the CallID of the resulting new call and the DeviceID of the consulting device.

2.6.1.11 Deflect Call

Description

The Deflect Call service allows the computing function to divert an alerting, queued or failed call to another destination that may be inside or outside the switching sub-domain.

Note

The ConnectionID has to be constructed as both (including CallID and DeviceID).
For an UCD call the deflect service has to be invoked on behalf of the agent device if the call has been delivered to an agent. Only if the UCD call is still available in the UCD itself (e.g. no available agents) the deflect service may be invoked on behalf of the UCD group.

Operation

```
deflectCall     OPERATION ::=
{       ARGUMENT                        DeflectCallArgument
        RESULT                          DeflectCallResult
        ERRORS                          {universalFailure}
        CODE                            local: 218
}
```

Service request

```
DeflectCallArgument ::=
      SEQUENCE
      {     callToBeDiverted                ConnectionID,
            newDestination                  DeviceID,
            addNewDestinationsList DeviceIDList                         OPTIONAL,
                                    -- List of additional new Destinations
            correlatorData                  CorrelatorData              OPTIONAL,
            userData                        UserData                    OPTIONAL,
            callCharacteristics             [0] IMPLICIT CallCharacteristics    OPTIONAL,
            subjectOfCall                   [1] IMPLICIT SubjectOfCall          OPTIONAL,
            languagePreferences             [2] IMPLICIT LanguagePreferences    OPTIONAL,
            reason                          [3] IMPLICIT EventCause             OPTIONAL,
            extensions                      CSTACommonArguments         OPTIONAL}
```

Parameter Name	Type	Description
callToBeDiverted	ConnectionID	Specifies the connection to be diverted.
newDestination	DeviceID	Specifies the device to which the call is to be diverted (newDestination device).

Service response

```
DeflectCallResult ::=
    CHOICE
    {extensions                    CSTACommonArguments,
     noData                        NULL}
```

~~2.6.1.12 Dial Digits~~

Not supported

2.6.1.13 Directed Pickup Call

Description

The Directed Pickup Call service moves a specified call and connects it at a new specified destination. This results in the connection being diverted to a new destination inside the switching sub-domain.

Note

The ConnectionID may to be constructed as callID, deviceID or both.

Operation

```
directedPickupCall    OPERATION ::=
{     ARGUMENT                     DirectedPickupCallArgument
      RESULT                       DirectedPickupCallResult
      ERRORS                       {universalFailure}
      CODE                         local: 220
}
```

Service request

```
DirectedPickupCallArgument ::=
    SEQUENCE
    {   callToBePickedUp           ConnectionID,
        requestingDevice           DeviceID,
        correlatorData             CorrelatorData                OPTIONAL,
        userData                   UserData                      OPTIONAL,
        extensions                 CSTACommonArguments           OPTIONAL}
```

Parameter Name	*Type*	*Description*
callToBePickedUp	ConnectionID	Specifies the connection to be picked up.
requestingDevice	DeviceID	Specifies the device which is picking up the call.

Service response

```
DirectedPickupCallResult ::=
    SEQUENCE
    {   pickedCall                 ConnectionID                  OPTIONAL,
        pickedCallInfo             ConnectionInformation         OPTIONAL,
        extensions                 CSTACommonArguments           OPTIONAL}
```

2.6.1.14 Group Pickup Call

Description

The Group Pickup Call service moves a call that is a member of a specified or default pickup group to a new specified destination.

- This results in a connection in a pickup group to be connected to a new specified destination inside the switching sub-domain.

- Note that the difference between this service and the Directed Pickup Call service is that Directed Pickup Call service specifies the actual connection to be picked up whereby the Group Pickup Call service does not.

Operation

```
groupPickupCall OPERATION ::=
{       ARGUMENT                        GroupPickupCallArgument
        RESULT                          GroupPickupCallResult
        ERRORS                          {universalFailure}
        CODE                            local: 221
}
```

Service request

```
GroupPickupCallArgument ::=
      SEQUENCE
      {     newDestination              DeviceID,
            pickGroup                   DeviceID                    OPTIONAL,
            correlatorData              CorrelatorData              OPTIONAL,
            userData                    UserData                    OPTIONAL,
            extensions                  CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
newDestination	DeviceID	Specifies the device which is picking up the call.

Service response

```
GroupPickupCallResult ::=
      SEQUENCE
      {     pickedCall                  ConnectionID                OPTIONAL,
            pickedCallInfo              ConnectionInformation       OPTIONAL,
            extensions                  CSTACommonArguments         OPTIONAL}
```

2.6.1.15 Hold Call

Description

The Hold Call service places an external call on hold at the same device. This service interrupts communication for an existing call at a device.

Note

The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID).

Operation

```
holdCall      OPERATION ::=
{       ARGUMENT                        HoldCallArgument
        RESULT                          HoldCallResult
        ERRORS                          {universalFailure}
        CODE                            local: 9
}
```

Service request

```
HoldCallArgument ::=
      SEQUENCE
      {callToBeHeld                     ConnectionID,
       connectionReservation            BOOLEAN                     OPTIONAL,
       extensions                       CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
callToBeHeld	ConnectionID	Specifies the active connection to be held.

Service response

```
HoldCallResult ::=
      CHOICE
      {extensions                    CSTACommonArguments,
       noData                        NULL}
```

~~2.6.1.16 Intrude Call~~

Not supported

~~2.6.1.17 Join Call~~

Not supported

2.6.1.18 Make Call

Description

The Make Call service allows the computing function to set up a call between a calling device and a called device. The service creates a new call and establishes an initiated or connected connection with the calling device. The Make Call service assigns a ConnectionID to the calling device and returns it in the positive acknowledgement.
In the process of establishing the connection with the calling device, the calling device may be prompted to go off-hook (if necessary) and when that device does so, a call to the called device is originated or the calling device is still in the process of dialling the called device.

Note

- The make call is executed immediately for Calling Device with handsfree or Calling Device offhook listening to dial tone.
- However, for hotel applications, the automatic make call with handsfree is administratable so that the handsfree is ignored and ringing is applied to the Calling Device till answering (or time out).
  The MakeCallResult is given after answering of Calling Device. This can be immediately with handsfree or after time when answering upon ringing.
- If CalledDeviceId isn't provided in this message (length=0), Music on Hold is connected to the CallingDeviceId. This special feature is also created for hotel applications (wake up call).
- The parameter "autoOriginate" is not provided. The system provides always DoNotPrompt at system phones (HFA) and Prompt at analogue phones. SIP depends on endpoint capabilities.
- The MakeCall service is implemented for subsribers and for MULAPs. It is not useful for other groups e.g. UCD.
- The usage of features / service codes via MakeCall is not released.
- If the max. BHCA (busy hour call attempts) on a CSTA controlled OpenScape Business node (s. 2.18.4 CSTA Services Overload-Rejection) is reached, and then the MakeCall service will fail with error cause performanceLimitedExceeded.

Operation

```
makeCall OPERATION ::=
{       ARGUMENT                MakeCallArgument
        RESULT                  MakeCallResult
        ERRORS                  {universalFailure}
        CODE                    local: 10
}
```

Service request

```
MakeCallArgument ::= SEQUENCE
{       callingDevice                   DeviceID,
        calledDirectoryNumber           DeviceID,
        accountCode                     [0] IMPLICIT AccountInfo                OPTIONAL,
        authCode                        [1] IMPLICIT AuthCode                   OPTIONAL,
        autoOriginate                   [3] IMPLICIT AutoOriginate              DEFAULT prompt,
        correlatorData                  [2] IMPLICIT CorrelatorData             OPTIONAL,
        userData                        UserData                                OPTIONAL,
        callCharacteristics             CallCharacteristics                     OPTIONAL,
        mediaCallCharacteristics        [4] IMPLICIT MediaCallCharacteristics   OPTIONAL,
        callingConnectionInfo           [5] IMPLICIT ConnectionInformation      OPTIONAL,
        subjectOfCall                   [6] IMPLICIT SubjectOfCall              OPTIONAL,
        languagePreferences             [7] IMPLICIT LanguagePreferences        OPTIONAL,
        reason                          [8] IMPLICIT EventCause                 OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	***Type***	***Description***
callingDevice	DeviceID	Specifies the calling/originating device. ~~Note that this device may be a device that represents a group of devices. In this case the callingDevice in the service request is different from the actual calling device.~~
calledDirectoryNumber	DeviceID	Specifies the called device.

Service response

```
MakeCallResult ::= SEQUENCE
{       callingDevice                   ConnectionID,
        mediaCallCharacteristics        [0] IMPLICIT MediaCallCharacteristics   OPTIONAL,
                                        -- mandatory, if the call charachteristics have been adjusted.
        initiatedCallInfo               [1] IMPLICIT ConnectionInformation      OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

~~2.6.1.19 Make Predictive Call~~

Not supported

2.6.1.20 Park Call

Description

The Park Call service moves a specified call at a device to a specified (parked-to) destination.
The device on whose behalf Park Call was invoked (the parking device) is no longer associated with the call.

Note

The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID).

Operation

```
answerCall                      OPERATION ::=
{       ARGUMENT                ParkCallArgument
        RESULT                  ParkCallResult
        ERRORS                  {universalFailure}
        CODE                    local: 2
}
```

Service request

```
ParkCallArgument ::=
    SEQUENCE
    {   parking             ConnectionID,
        parkTo              SubjectDeviceID
        correlatorData      CorrelatorData          OPTIONAL,
        subjectOfCall       SubjectofCall           OPTIONAL
        languagePreferences LanguagePreferences     OPTIONAL
        extensions          CSTACommonArguments     OPTIONAL}
```

Parameter Name	*Type*	*Description*
parkedTo	ConnectionID	Specifies the connection to be answered.
~~security~~	~~CSTASecurityData~~	~~Specifies timestamp information, message sequence number, and security information.~~
~~privateData~~	~~CSTAPrivateData~~	~~Specifies non-standardized information.~~

Service response

```
ParkCallResult ::=
    CHOICE
    {extensions         CSTACommonArguments,
     parkPosition       privateDeviceID}
```

2.6.1.21 Reconnect Call

Description

The Reconnect Call service will clear a specified connection at the reconnecting device and retrieve a specified held connection at the same device. This works even for cascaded consultation calls.

Note

The ConnectionIDs may be constructed as callID, deviceID or both, but at least one of them (activeCall, heldCall) has to include a DeviceID.

Operation

```
reconnectCall OPERATION ::=
{   ARGUMENT            ReconnectCallArgument
    RESULT              ReconnectCallResult
    ERRORS              {universalFailure}
    CODE                local: 13
}
```

Service request

```
ReconnectCallArgument ::= SEQUENCE
{   activeCall          ConnectionID,
    heldCall            ConnectionID,
    extensions          CSTACommonArguments     OPTIONAL}
```

Parameter Name	*Type*	*Description*
activeCall	ConnectionID	Specifies the connection to be cleared.
heldCall	ConnectionID	Specifies the held connection.

Service response

```
ReconnectCallResult ::=
    CHOICE
    {extensions         CSTACommonArguments,
     noData             NULL}
```

2.6.1.22 Retrieve Call

Description

The Retrieve Call service connects a specified held connection.

Note
The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID).

Operation

```
retrieveCall        OPERATION ::=
{       ARGUMENT                        RetrieveCallArgument
        RESULT                          RetrieveCallResult
        ERRORS                          {universalFailure}
        CODE                            local: 14
}
```

Service request

```
RetrieveCallArgument ::=
      SEQUENCE
      {     callToBeRetrieved           ConnectionID,
            extensions                  CSTACommonArguments                 OPTIONAL}
```

Parameter Name	*Type*	*Description*
callToBeRetrieved	ConnectionID	Specifies the held connection to be retrieved.

Service response

RetrieveCallResult ::=

```
      CHOICE
      {extensions                       CSTACommonArguments,
       noData                           NULL}
```

~~2.6.1.23 Single Step Conference Call~~

Not supported

2.6.1.24 Single Step Transfer Call

Description

The Single Step Transfer Call service transfers an existing connected connection at a device to another device. This transfer is performed in a single-step that is the device doing the transfer does not have to place the existing call on hold before issuing the Single Step Transfer Call service.

Note
The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID).

Operation

```
singleStepTrans OPERATION ::=
{       ARGUMENT                        SingleStepTransArgument
        RESULT                          SingleStepTransResult
        ERRORS                          {universalFailure}
        CODE                            local: 50
}
```

Service request

SingleStepTransArgument ::=

```
SEQUENCE
{activeCall          ConnectionID,
 tranferredTo        DeviceID,
 accountCode         [0] IMPLICIT AccountInfo          OPTIONAL,
 authCode            [1] IMPLICIT AuthCode             OPTIONAL,
 correlatorData      [2] IMPLICIT CorrelatorData       OPTIONAL,
 userData            UserData                          OPTIONAL,
 callCharacteristics [3] IMPLICIT CallCharacteristics  OPTIONAL,
 subjectOfCall       [4] IMPLICIT SubjectOfCall        OPTIONAL,
 languagePreferences [5] IMPLICIT LanguagePreferences  OPTIONAL,
 reason              [6] IMPLICIT EventCause           OPTIONAL,
 extensions          CSTACommonArguments               OPTIONAL}
```

Parameter Name	*Type*	*Description*
activeCall	ConnectionID	Specifies the connected connection in the call to be replaced.
transferredTo	DeviceID	Specifies the new called (transferred-to) device.

Service response

SingleStepTransResult ::= SEQUENCE

```
{transferredCall     ConnectionID,
 connections         [0] IMPLICIT ConnectionList          OPTIONAL,
 transferredCallInfo [1] IMPLICIT ConnectionInformation   OPTIONAL,
 extensions          CSTACommonArguments                  OPTIONAL}
```

2.6.1.25 Transfer Call

Description

The Transfer Call service transfers a call held at a device to an active call at the same device. The held and active calls at the transferring device shall be merged into a new call. Also, the Connections of the held and active calls at the transferring device shall become Null and their ConnectionIDs shall be released (i.e., the transferring device is no longer involved with the call).

Note

- The ConnectionIDs may be constructed as callID, deviceID or both, but at least one of them (activeCall, heldCall) has to include a DeviceID.
- The heldCall of the TransferCallArgument has to be the last held Call. Other held Calls are not supported.
- The held call must not be in the common hold queue (i.e. being held via holdCall service). Such a call in the common hold queue has to be retrieved via retrieveCall service first.

Operation

```
transferCall OPERATION ::=
{    ARGUMENT    TransferCallArgument
     RESULT      TransferCallResult
     ERRORS      {universalFailure}
     CODE        local: 16
}
```

Service request

TransferCallArgument ::= SEQUENCE

```
{    heldCall    ConnectionID,
     activeCall  ConnectionID,
     extensions  CSTACommonArguments   OPTIONAL}
```

Parameter Name	*Type*	*Description*
heldCall	ConnectionID	Specifies the held connection.
activeCall	ConnectionID	Specifies the active connection.

Service response

```
TransferCallResult ::=  SEQUENCE
{       transferredCall                 ConnectionID,
        connections                     [0] IMPLICIT ConnectionList                 OPTIONAL,
        transferredCallInfo             [1] IMPLICIT ConnectionInformation          OPTIONAL,
        extensions                      CSTACommonArguments                         OPTIONAL}
```

Parameter Name	*Type*	*Description*
transferredCall	ConnectionID	Specifies the ConnectionID of the transferred-to device in the resulting call.

2.6.2 Events

Note

- The max. amount of devices that can be monitored at the same time is stated in chapter 2.18.1 OpenScape Business parameter constraints. See also 2.18.3 List of monitorable OpenScape Business devices.
  There is no event report on non-voice calls.
- The OpenScape Business handles several trunks with their different capabilities and harmonizes these different capabilities as good as it is possible on the CSTA Interface.
  That means, for Outbound calls you will see a Network Reached Event but the following Events cannot be guaranteed due to the signaling type of the network. This is because most of the Networking Messages are optional.

~~*2.6.2.1 Bridged*~~

Not supported

~~*2.6.2.2 Call Cleared*~~

Not supported

2.6.2.3 Conferenced

Description

The Conferenced event indicates that the conferencing device has conferenced itself or another device with an existing call and that no devices have been removed from the resulting call.
Common situations that generate this event include:

- Two step conferencing situations (manual and service initiated)
- Single step conferencing situations (manual and service initiated)

Note

- The conferenced event provides the global view of connections and a new CallID is used.
- In case of feature Silent-Monitoring the old called is re-used, no new called is used.

Operation

```
ConferencedEvent ::= SEQUENCE
{       primaryOldCall              ConnectionID,
        secondaryOldCall            ConnectionID                        OPTIONAL,
        conferencingDevice          SubjectDeviceID,
        addedParty                  SubjectDeviceID,
        conferenceConnections       ConnectionList,
        localConnectionInfo         LocalConnectionState                OPTIONAL,
        correlatorData              [1] IMPLICIT CorrelatorData         OPTIONAL,
        userData                    UserData                            OPTIONAL,
        cause                       EventCause,
        servicesPermitted           [2] IMPLICIT ServicesPermitted      OPTIONAL,
        mediaCallCharacteristics    [3] IMPLICIT MediaCallCharacteristics OPTIONAL,
        callCharacteristics         [4] IMPLICIT CallCharacteristics    OPTIONAL,
        callLinkageDataList         [6] IMPLICIT CallLinkageDataList    OPTIONAL,
        languagePreferences         [7] IMPLICIT LanguagePreferences    OPTIONAL,
        deviceHistory               [8] IMPLICIT DeviceHistory          OPTIONAL,
        locationInfo                [9] IMPLICIT LocationInfoList       OPTIONAL,
        extensions                  [5] IMPLICIT CSTACommonArguments    OPTIONAL}
```

Parameter Name	***Type***	***Description***
primaryOldCall	ConnectionID	Specifies the connection of the primary call. See Functional Requirement.
secondaryOldCall	ConnectionID	Specifies the connection of the secondary call. If the switching function supports the „fixed-view“ option (as indicated by the capability exchange services), this parameter is mandatory. If the switching function supports the „local-view“ option, this parameter is mandatory if there are two known calls involved with the conference (before the conference is created) from the perspective of the monitored device, otherwise it shall not be provided.
conferencingDevice	SubjectDeviceID	Specifies the device ID of the conferencing device.
addedParty	SubjectDeviceID	Specifies the device ID of the last device added to the call.
conferenceConnections	ConnectionList	Specifies information on each device/ConnectionID in the resulting conference call. See Functional Requirement.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices that are in the conference call: - Conferencing device - Connected (or Unaffected for Case B). - Other devices - (See above figures for specific cases of the Conferenced event) This parameter is mandatory for events generated for device-type monitors and otherwise shall not be provided.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Active Participation - Conference - Silent Participation
~~callLinkageDataList~~	~~CallLinkageDataList~~	
security	CSTASecurityData	Specifies timestamp information

2.6.2.4 Connection Cleared

Description

The Connection Cleared event indicates that a single device has disconnected or dropped out of a call. Common situations that generate this event include:

- A user manually terminates the call (by going on-hook, for example).
- The Clear Connection service is successfully invoked.
- Connection clears as a result of some other services operation.

Operation

ConnectionClearedEvent ::=

```
SEQUENCE
{ droppedConnection        ConnectionID,
  releasingDevice          SubjectDeviceID,
  localConnectionInfo      LocalConnectionState          OPTIONAL,
  correlatorData           [0] IMPLICIT CorrelatorData   OPTIONAL,
  userData                 UserData                      OPTIONAL,
  chargingInfo             [1] IMPLICIT ChargingInfo     OPTIONAL,
  cause                    EventCause,
  servicesPermitted        [2] IMPLICIT ServicesPermitted OPTIONAL,
  mediaCallCharacteristics [3] IMPLICIT MediaCallCharacteristics OPTIONAL,
  callCharacteristics      [4] IMPLICIT CallCharacteristics OPTIONAL,
  droppedConnectionInfo    [5] IMPLICIT ConnectionInformation OPTIONAL,
  callLinkageData          [6] IMPLICIT CallLinkageData  OPTIONAL,
  languagePreferences      [7] IMPLICIT LanguagePreferences OPTIONAL,
  deviceHistory            [8] IMPLICIT DeviceHistory    OPTIONAL,
  locationInfo             [9] IMPLICIT LocationInfoList OPTIONAL,
  extensions               CSTACommonArguments           OPTIONAL}
```

Parameter Name	***Type***	***Description***
droppedConnection	ConnectionID	Specifies the connection of the device that was dropped from the call.
releasingDevice	SubjectDeviceID	Specifies the device that dropped from the call.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the clearing device: Null - For the other devices left in the call: (unaffected) This parameter is mandatory for events generated for device-type monitors and otherwise shall not be provided.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Call Back - Normal Clearing - Reorder Tone
~~callLinkageData~~	~~CallLinkageData~~	
security	CSTASecurityData	Specifies timestamp information

2.6.2.5 Delivered

Description

The Delivered event indicates that a call is being presented to a device in either the Ringing or Entering Distribution modes of the alerting state. Common situations that generate this event include:

- A call has been assigned to a device and that device is alerting.
- A call has been assigned to a distribution device such as an ACD.
- A call has been assigned to a distribution device such as a hunt group. Note that in this case only the hunt group members receive this event.

Note

- The originatingNIDConnection contains the Network Interface Device for external connections.

Operation

DeliveredEvent ::=

```
SEQUENCE
{   connection              ConnectionID,
    alertingDevice          SubjectDeviceID,
    callingDevice           CallingDeviceID,
    calledDevice            CalledDeviceID,
    lastRedirectionDevice   RedirectionDeviceID,
```

originatingNIDConnection	ConnectionID	OPTIONAL,
localConnectionInfo	LocalConnectionState	OPTIONAL,
~~correlatorData~~	~~[0] IMPLICIT CorrelatorData~~	~~OPTIONAL~~,
~~userData~~	~~UserData~~	~~OPTIONAL~~,
cause	EventCause,	
~~servicesPermitted~~	~~[1] IMPLICIT ServicesPermitted~~	~~OPTIONAL~~,
networkCallingDevice	NetworkCallingDeviceID	OPTIONAL,
networkCalledDevice	NetworkCalledDeviceID	OPTIONAL,
associatedCallingDevice	AssociatedCallingDeviceID	OPTIONAL,
associatedCalledDevice	AssociatedCalledDeviceID	OPTIONAL,
~~mediaCallCharacteristics~~	~~[2] IMPLICIT MediaCallCharacteristics~~	~~OPTIONAL~~,
~~callCharacteristics~~	~~[3] IMPLICIT CallCharacteristics~~	~~OPTIONAL~~,
~~connectionInfo~~	~~[4] IMPLICIT ConnectionInformation~~	~~OPTIONAL~~,
~~callLinkageData~~	~~[5] IMPLICIT CallLinkageData~~	~~OPTIONAL~~,
~~subjectOfCall~~	~~[6] IMPLICIT SubjectOfCall~~	~~OPTIONAL~~,
~~messageInfo~~	~~[7] IMPLICIT MessageInfo~~	~~OPTIONAL~~,
~~languagePreferences~~	~~[8] IMPLICIT LanguagePreferences~~	~~OPTIONAL~~,
~~deviceHistory~~	~~[9] IMPLICIT DeviceHistory~~	~~OPTIONAL~~,
~~locationInfo~~	~~[10] IMPLICIT LocationInfoList~~	~~OPTIONAL~~,
extensions	CSTACommonArguments	OPTIONAL}

Parameter Name	***Type***	***Description***
connection	ConnectionID	Specifies the connection that is alerting.
alertingDevice	SubjectDeviceID	Specifies the device that is alerting.
callingDevice	CallingDeviceID	Specifies the calling device.
calledDevice	CalledDeviceID	Specifies the originally called device.
lastRedirectionDevice	RedirectionDeviceID	Specifies the previously known redirected from device.
originatingNIDConnection	ConnectionID	Specifies the connection of the Network Interface Device (NID) that the call originated from. If this parameter is supported, it shall be provided if there is an originating NID associated with the call, otherwise it shall not be provided.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the alerting device: Alerting - For the calling device: (unaffected) This parameter is mandatory for events generated for device-type monitors and otherwise shall not be provided.
Cause	EventCause	Specifies the reason for the event. Supported causes are: • Call Back • Call Forward • Call Forward Busy • Call Forward Immediate • Call Forward No Answer • Distributed • Entering Distribution • New Call • Recall • Redirected • Single Step Transfer • callInterception – 1) • callInterceptionBusy – 1) • callInterceptionForwarded – 1) • callInterceptionNoAnswer – 1) • callInterceptionResourcesNotAvailable – 1) • pathReplacement – 1) • recallBusy – 1) • recallForwarded – 1) • recallNoAnswer – 1) • recallResourcesNotAvailable – 1)
networkCallingDevice	NetworkCallingDeviceID	Specifies the original calling device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
networkCalledDevice	NetworkCalledDeviceID	Specifies the original called device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
associatedCallingDevice	AssociatedCallingDeviceID	Specifies the Network Interface Device associated with the calling device if the call is an external incoming call. This parameter is mandatory for all external incoming calls and shall not be provided otherwise.
associatedCalledDevice	AssociatedCalledDeviceID	For outgoing external calls, this parameter specifies the Network Interface Device associated with the originally called device. For incoming external calls, this parameter specifies a device within the switching sub-domain associated with the originally called device. This parameter is mandatory for all external outgoing calls and it is optional for external incoming calls.
~~callLinkageData~~	~~CallLinkageData~~	
security	CSTASecurityData	Specifies timestamp information

1) import from ECMA-269 9th edition

~~2.6.2.6 Digits dialed~~

Not supported

2.6.2.7 Diverted

Description

The Diverted event indicates that a call has been diverted from a device and that the call is no longer present at the device. Common situations that generate this event include:

- A call leaves a device that has some type of forwarding feature activated. Examples are Ring No Answer, (depends upon the forwarding feature model supported by the switching function).
- A call leaves an ACD group device to be redirected to an agent, an extension, another ACD Group device, or to an offsite destination.
- A call leaves an ACD queue and is redirected to either an agent or an extension.
- A divert (Deflect, Pick, etc.) is successfully invoked.

Operation

DivertedEvent ::= SEQUENCE

```
{   connection                  ConnectionID,
    divertingDevice             SubjectDeviceID,
    newDestination              SubjectDeviceID,
    callingDevice               CallingDeviceID                 OPTIONAL,
    calledDevice                CalledDeviceID                  OPTIONAL,
    lastRedirectionDevice       RedirectionDeviceID,
    localConnectionInfo         LocalConnectionState            OPTIONAL,
    correlatorData              [0] IMPLICIT CorrelatorData     OPTIONAL,
    userData                    UserData                        OPTIONAL,
    cause                       EventCause,
    servicesPermitted           [1] IMPLICIT ServicesPermitted  OPTIONAL,
    mediaCallCharacteristics [2] IMPLICIT MediaCallCharacteristics OPTIONAL,
    callCharacteristics         [3] IMPLICIT CallCharacteristics OPTIONAL,
    connectionInfo              [4] IMPLICIT ConnectionInformation OPTIONAL,
    networkCallingDevice        NetworkCallingDeviceID          OPTIONAL,
    networkCalledDevice         NetworkCalledDeviceID           OPTIONAL,
    associatedCallingDevice     AssociatedCallingDeviceID       OPTIONAL,
    associatedCalledDevice      AssociatedCalledDeviceID        OPTIONAL,
    callLinkageData             [5] IMPLICIT CallLinkageData    OPTIONAL,
    subjectOfCall               [6] IMPLICIT SubjectOfCall      OPTIONAL,
    messageInfo                 [7] IMPLICIT MessageInfo        OPTIONAL,
    languagePreferences         [8] IMPLICIT LanguagePreferences OPTIONAL,
    deviceHistory               [9] IMPLICIT DeviceHistory      OPTIONAL,
    locationInfo                [10] IMPLICIT LocationInfoList  OPTIONAL,
    extensions                  CSTACommonArguments             OPTIONAL}
```

Parameter Name	***Type***	***Description***
connection	ConnectionID	Specifies the connection that was diverted.
divertingDevice	SubjectDeviceID	Specifies the device from which the call was diverted.
newDestination	SubjectDeviceID	Specifies the device to which the call was diverted.
lastRedirectionDevice	RedirectionDeviceID	Specifies the previously known redirected from device. Note that this is not the divertingDevice that caused this event, but the device that performed a redirection towards the current divertingDevice just before the current redirection.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the diverting device: Null - For the other devices left in the call: (unaffected) This parameter is mandatory for events generated for device-type monitors and otherwise shall not be provided.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Call Pickup - Distributed - Park - Recall - Redirected
networkCallingDevice	NetworkCallingDeviceID	Specifies the original calling device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.

networkCalledDevice	NetworkCalledDeviceID	Specifies the original called device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
associatedCallingDevice	AssociatedCallingDeviceID	Specifies the Network Interface Device associated with the calling device if the call is an external incoming call. This parameter is mandatory for all external incoming calls and shall not be provided otherwise.
associatedCalledDevice	AssociatedCalledDeviceID	For outgoing external calls, this parameter specifies the Network Interface Device associated with the originally called device. For incoming external calls, this parameter specifies a device within the switching sub-domain associated with the originally called device. This parameter is mandatory for all external outgoing calls and it is optional for external incoming calls.
~~callLinkageData~~	~~CallLinkageData~~	
security	CSTASecurityData	Specifies timestamp information

2.6.2.8 Established

Description

The Established event indicates that a call has been answered at a device or that a call has been connected to a device.
Common situations that generate this event include:

- A call has been answered at a device (e.g., a user has manually gone off-hook).
- The Answer Call service has been successfully invoked.
- A call has been picked by another device.

Note

The originatingNIDConnection contains the Network Interface Device for external.

Operation

```
EstablishedEvent ::=
    SEQUENCE
    { establishedConnection          ConnectionID,
      answeringDevice                SubjectDeviceID,
      callingDevice                  CallingDeviceID,
      calledDevice                   CalledDeviceID,
      lastRedirectionDevice          RedirectionDeviceID,
      originatingNIDConnection       ConnectionID                          OPTIONAL,
      localConnectionInfo            LocalConnectionState                  OPTIONAL,
      correlatorData                 [1] IMPLICIT CorrelatorData           OPTIONAL,
      userData                       UserData                              OPTIONAL,
      cause                          EventCause,
      servicesPermitted              [2] IMPLICIT ServicesPermitted        OPTIONAL,
      networkCallingDevice           NetworkCallingDeviceID                OPTIONAL,
      networkCalledDevice            NetworkCalledDeviceID                 OPTIONAL,
      associatedCallingDevice        AssociatedCallingDeviceID             OPTIONAL,
      associatedCalledDevice         AssociatedCalledDeviceID              OPTIONAL,
      mediaCallCharacteristics       [3] IMPLICIT MediaCallCharacteristics OPTIONAL,
      callCharacteristics            [4] IMPLICIT CallCharacteristics      OPTIONAL,
      establishedConnectionInfo      [5] IMPLICIT ConnectionInformation    OPTIONAL,
      callLinkageData                [6] IMPLICIT CallLinkageData          OPTIONAL,
      subjectOfCall                  [7] IMPLICIT SubjectOfCall            OPTIONAL,
      messageInfo                    [8] IMPLICIT MessageInfo              OPTIONAL,
      languagePreferences            [9] IMPLICIT LanguagePreferences      OPTIONAL,
      deviceHistory                  [10] IMPLICIT DeviceHistory           OPTIONAL,
      locationInfo                   [11] IMPLICIT LocationInfoList        OPTIONAL,
      extensions                     CSTACommonArguments                   OPTIONAL}
```

Parameter Name	***Type***	***Description***
establishedConnection	ConnectionID	Specifies the connection that was connected.
answeringDevice	SubjectDeviceID	Specifies the device that connected into the call.
callingDevice	CallingDeviceID	Specifies the calling device.
calledDevice	CalledDeviceID	Specifies the originally called device.
lastRedirectionDevice	RedirectionDeviceID	Specifies the previously known redirected from device.
originatingNIDConnection	ConnectionID	Specifies the connection of the Network Interface Device (NID) that the call originated from. If this parameter is supported, it shall be provided if there is an originating NID associated with the call, otherwise it shall not be provided.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the answering device: Connected - For the calling device: (unaffected - never Null) This parameter is mandatory for events generated for device-type monitors and otherwise shall not be provided.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Call Pickup - Intrude - New Call - Single Step Transfer
networkCallingDevice	NetworkCallingDeviceID	Specifies the original calling device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
networkCalledDevice	NetworkCalledDeviceID	Specifies the original called device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
associatedCallingDevice	AssociatedCallingDeviceID	Specifies the Network Interface Device associated with the calling device if the call is an external incoming call. This parameter is mandatory for all external incoming calls and shall not be provided otherwise.
associatedCalledDevice	AssociatedCalledDeviceID	For outgoing external calls, this parameter specifies the Network Interface Device associated with the originally called device. For incoming external calls, this parameter specifies a device within the switching sub-domain associated with the originally called device. This parameter is mandatory for all external outgoing calls and it is optional for external incoming calls.
callLinkageData	CallLinkageData	
security	CSTASecurityData	Specifies timestamp information

2.6.2.9 Failed

Description

The Failed event indicates that a call cannot be completed or a connection has entered the Fail state.

Note

The originatingNIDConnection contains the Network Interface Device for external.

Operation

FailedEvent ::= SEQUENCE

```
{ failedConnection            ConnectionID,
  failingDevice               SubjectDeviceID,
  callingDevice               CallingDeviceID,
  calledDevice                CalledDeviceID,
  lastRedirectionDevice       RedirectionDeviceID,
  originatingNIDConnection    ConnectionID                          OPTIONAL,
  localConnectionInfo         LocalConnectionState                  OPTIONAL,
  correlatorData              [0] IMPLICIT CorrelatorData           OPTIONAL,
  userData                    UserData                              OPTIONAL,
  cause                       EventCause,
  servicesPermitted           [1] IMPLICIT ServicesPermitted        OPTIONAL,
  networkCallingDevice        NetworkCallingDeviceID                OPTIONAL,
  networkCalledDevice         NetworkCalledDeviceID                 OPTIONAL,
  associatedCallingDevice     AssociatedCallingDeviceID             OPTIONAL,
  associatedCalledDevice      AssociatedCalledDeviceID              OPTIONAL,
  mediaCallCharacteristics    [2] IMPLICIT MediaCallCharacteristics OPTIONAL,
  callCharacteristics         [3] IMPLICIT CallCharacteristics      OPTIONAL,
  failedConnectionInfo        [4] IMPLICIT ConnectionInformation    OPTIONAL,
  callLinkageData             [5] IMPLICIT CallLinkageData          OPTIONAL,
  subjectOfCall               [6] IMPLICIT SubjectOfCall            OPTIONAL,
  messageInfo                 [7] IMPLICIT MessageInfo              OPTIONAL,
  languagePreferences         [8] IMPLICIT LanguagePreferences      OPTIONAL,
  deviceHistory               [9] IMPLICIT DeviceHistory            OPTIONAL,
  locationInfo                [10] IMPLICIT LocationInfoList        OPTIONAL,
  extensions                  CSTACommonArguments                   OPTIONAL}
```

Parameter Name	***Type***	***Description***
failedConnection	ConnectionID	Specifies the connection that failed.
failingDevice	SubjectDeviceID	Specifies the device that failed.
callingDevice	CallingDeviceID	Specifies the calling device.
calledDevice	CalledDeviceID	Specifies the originally called device.
lastRedirectionDevice	RedirectionDeviceID	Specifies the previously known redirected-from device.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the failing device: Fail - For the other devices left in the call: (unaffected) This parameter is mandatory for events generated for device-type monitors and otherwise shall not be provided.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Blocked - Busy - Destination not Obtainable - Do not Disturb - Reorder Tone - Trunks Busy
networkCallingDevice	NetworkCallingDeviceID	Specifies the original calling device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
networkCalledDevice	NetworkCalledDeviceID	Specifies the original called device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
associatedCallingDevice	AssociatedCallingDeviceID	Specifies the Network Interface Device associated with the otherDevice if the call is an external incoming call. This parameter is mandatory for all external incoming calls and shall not be provided otherwise.
associatedCalledDevice	AssociatedCalledDeviceID	For outgoing external calls, this parameter specifies the Network Interface Device associated with the originally called device. For incoming external calls, this parameter specifies a device within the switching sub-domain associated with the originally called device. This parameter is mandatory for all external outgoing calls and it is optional for external incoming calls.
~~callLinkageData~~	~~CallLinkageData~~	
security	CSTASecurityData	Specifies timestamp information

2.6.2.10 Held

Description

The Held event indicates that a call has been placed on hold. Common situations that generate this event include:

- Consultation situations (manual and service initiated).
- Hold situations (manual and service initiated).

Operation

HeldEvent ::=

```
SEQUENCE
{   heldConnection              ConnectionID,
    holdingDevice               SubjectDeviceID,
    localConnectionInfo         LocalConnectionState                OPTIONAL,
    correlatorData              [0] IMPLICIT CorrelatorData         OPTIONAL,
    cause                       EventCause,
    servicesPermitted           [1] IMPLICIT ServicesPermitted      OPTIONAL,
    mediaCallCharacteristics    [2] IMPLICIT MediaCallCharacteristics OPTIONAL,
    callCharacteristics         [3] IMPLICIT CallCharacteristics    OPTIONAL,
    heldConnectionInfo          [4] IMPLICIT ConnectionInformation  OPTIONAL,
    callLinkageData             [5] IMPLICIT CallLinkageData        OPTIONAL,
    languagePreferences         [6] IMPLICIT LanguagePreferences    OPTIONAL,
    locationInfo                [7] IMPLICIT LocationInfoList       OPTIONAL,
    extensions                  CSTACommonArguments                 OPTIONAL}
```

Parameter Name	***Type***	***Description***
heldConnection	ConnectionID	Specifies the connection at which the hold was activated.
holdingDevice	SubjectDeviceID	Specifies the device at which hold was activated.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the holding device: Hold - For the other devices left in the call: (unaffected) This parameter is mandatory for events generated for device-type monitors.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Alternate - Consultation - Normal
~~callLinkageData~~	~~CallLinkageData~~	
security	CSTASecurityData	Specifies timestamp information

~~2.6.2.11 Network Capabilities Changed~~

Not supported

2.6.2.12 Network Reached

Description

The Network Reached event indicates that a call has cut through the switching sub-domain boundary to another network; that is, has reached and engaged a Network Interface Device (e.g., trunk, CO Line). This event indicates that there may be a reduced level of event reporting and possibly no additional device feedback, except connection/call clearing, provided for this device in the call due to a lack of network signalling. The level of signalling provided by the network may be indicated by the networkCapability parameter.

Additionally, the computing function should assume that it cannot directly manipulate the far-end device associated with the Network Interface Device.

This event is never sent for calls made to devices that are within the switching sub-domain. This event indicates that a connection with a Network Interface Device has reached the connected state, and that

further events for that connection refer to the state of the endpoint which the Network Interface Device is associated.
A common situation that generates this event includes:

- An outgoing call has cut-through at a network interface device and further call progress information, such as the Delivered and Established events, may not be available.

Operation

NetworkReachedEvent ::=

```
SEQUENCE
{   outboundConnection          ConnectionID,
    networkInterfaceUsed        SubjectDeviceID,
    callingDevice               CallingDeviceID,
    calledDevice                CalledDeviceID,
    lastRedirectionDevice       RedirectionDeviceID,
    originatingNID              Conneciton ConnectionID          OPTIONAL,
    localConnectionInfo         LocalConnectionState             OPTIONAL,
    correlatorData              [0] IMPLICIT CorrelatorData      OPTIONAL,
    userData                    UserData                         OPTIONAL,
    networkCapability           [1] IMPLICIT NetworkCapability   OPTIONAL,
    cause                       EventCause,
    servicesPermitted           [2] IMPLICIT ServicesPermitted   OPTIONAL,
    mediaCallCharacteristics    [3] IMPLICIT MediaCallCharacteristics OPTIONAL,
    callCharacteristics         [4] IMPLICIT CallCharacteristics OPTIONAL,
    outboundConnectionInfo      [5] IMPLICIT ConnectionInformation OPTIONAL,
    networkCallingDevice        NetworkCallingDeviceID           OPTIONAL,
    networkCalledDevice         NetworkCalledDeviceID            OPTIONAL,
    associatedCallingDevice     AssociatedCallingDeviceID        OPTIONAL,
    callLinkageData             [6] IMPLICIT CallLinkageData     OPTIONAL,
    languagePreferences         [7] IMPLICIT LanguagePreferences OPTIONAL,
    deviceHistory               [8] IMPLICIT DeviceHistory       OPTIONAL,
    locationInfo                [9] IMPLICIT LocationInfoList    OPTIONAL,
    extensions                  CSTACommonArguments              OPTIONAL}
```

Parameter Name	***Type***	***Description***
outboundConnection	ConnectionID	Specifies the outbound connection associated with the call that is leaving the switching sub-domain.
networkInterfaceUsed	SubjectDeviceID	Specifies the Network Interface Device that was selected.
callingDevice	CallingDeviceID	Specifies the calling device.
calledDevice	CalledDeviceID	Specifies the originally called device.
lastRedirectionDevice	RedirectionDeviceID	Specifies the previously known redirected from device.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the Network Interface Device: Connected - For the other devices left in the call: (unaffected) This parameter is mandatory for events generated for device-type monitors.
networkCapability	Structure	Specifies the type of network reached and the Call Control events supported by the network. It includes the following components: 1. networkType (M) - The complete set of possible values is: • ISDN public • Non-ISDN public • ISDN private • Non-ISDN private •Other 2. eventsProvided (O) - This is a bitmap of the Call Control - events defined in this Standard.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Normal
associatedCallingDevice	AssociatedCallingDeviceID	Specifies the Network Interface Device associated with the calling device if the call is an external incoming call (that, in this case is also an outgoing). This parameter is mandatory for all external incoming calls and shall not be provided otherwise.

~~callLinkageData~~	~~CallLinkageData~~	
security	CSTASecurityData	Specifies timestamp information

~~2.6.2.13 Offered~~

Not supported

2.6.2.14 Originated

Description

The Originated event indicates that a call is being attempted from a device. It implies that input activity for the call is complete and that a call (rather than a feature) has been requested.
Common situations that generate this event when the switch has originated a call at the originating device are:

- Due to the execution of the Make Call service.
- Due to the execution of the Consultation Call service.
- After a user has completed manually dialling a number.
- When an external incoming call originates from a Network Interface Device (e.g., trunk, CO line)

Operation

OriginatedEvent ::= SEQUENCE

```
{   originatedConnection        ConnectionID,
    callingDevice               SubjectDeviceID,
    calledDevice                CalledDeviceID,
    originatingDevice           DeviceID                        OPTIONAL,
    localConnectionInfo         LocalConnectionState            OPTIONAL,
    correlatorData              [2] IMPLICIT CorrelatorData     OPTIONAL,
    cause                       EventCause,
    servicesPermitted           [3] IMPLICIT ServicesPermitted  OPTIONAL,
    networkCallingDevice        NetworkCallingDeviceID          OPTIONAL,
    networkCalledDevice         NetworkCalledDeviceID           OPTIONAL,
    associatedCallingDevice     AssociatedCallingDeviceID       OPTIONAL,
    associatedCalledDevice      AssociatedCalledDeviceID        OPTIONAL,
    mediaCallCharacteristics    [4] IMPLICIT MediaCallCharacteristics OPTIONAL,
    callCharacteristics         [5] IMPLICIT CallCharacteristics OPTIONAL,
    originatedConnectionInfo    [6] IMPLICIT ConnectionInformation OPTIONAL,
    callLinkageData             [7] IMPLICIT CallLinkageData    OPTIONAL,
    subjectOfCall               [8] IMPLICIT SubjectOfCall      OPTIONAL,
    messageInfo                 [9] IMPLICIT MessageInfo        OPTIONAL,
    languagePreferences         [10] IMPLICIT LanguagePreferences OPTIONAL,
    locationInfo                [11] IMPLICIT LocationInfoList  OPTIONAL,
    extensions                  CSTACommonArguments             OPTIONAL}
```

Parameter Name	***Type***	***Description***
originatedConnection	ConnectionID	Specifies the connection at which the call originated.
callingDevice	SubjectDeviceID	Specifies the calling device.
calledDevice	CalledDeviceID	Specifies the originally called device.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the device initiating the call: Connected This parameter is mandatory for events generated for device-type monitors.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Call Back - Consultation - New Call
networkCallingDevice	NetworkCallingDeviceID	Specifies the original calling device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
networkCalledDevice	NetworkCalledDeviceID	Specifies the original called device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
associatedCallingDevice	AssociatedCallingDeviceID	Specifies the Network Interface Device associated with the calling device if the call is an external incoming call. This parameter is mandatory for all external incoming calls and shall not be provided otherwise.
associatedCalledDevice	AssociatedCalledDeviceID	For outgoing external calls, this parameter specifies the Network Interface Device associated with the originally called device. For incoming external calls, this parameter specifies a device within the switching sub-domain associated with the originally called device. This parameter is mandatory for all external outgoing calls and it is optional for external incoming calls.
~~callLinkageData~~	~~CallLinkageData~~	
security	CSTASecurityData	Specifies timestamp information

2.6.2.15 Queued

Description

The Queued event indicates that a call has been queued.
Common situations that generate this event include:

- A call is queued at an UCD device.
- A call is queued at a hunt group device. Note that in this case only the calling device (e.g. trunk) receives this event.
- A call is queued to a park-Device. In this case the extensions contain a privateDevice containing the parkPosition.
- A call is queued (camped on or parked, for example) at a device.

Operation

QueuedEvent ::=

```
SEQUENCE
{ queuedConnection          ConnectionID,
  queue                     SubjectDeviceID,
  callingDevice             CallingDeviceID,
  calledDevice              CalledDeviceID,
  lastRedirectionDevice     RedirectionDeviceID,
  numberQueued              [0] IMPLICIT INTEGER                    OPTIONAL,  -- only UCD
  callsInFront              [1] IMPLICIT INTEGER                    OPTIONAL,
  localConnectionInfo       LocalConnectionState                    OPTIONAL,
  correlatorData            [2] IMPLICIT CorrelatorData             OPTIONAL,
  userData                  UserData                                OPTIONAL,
  cause                     EventCause,
  servicesPermitted         [3] IMPLICIT ServicesPermitted          OPTIONAL,
  networkCallingDevice      NetworkCallingDeviceID                  OPTIONAL,
  networkCalledDevice       NetworkCalledDeviceID                   OPTIONAL,
  associatedCallingDevice   AssociatedCallingDeviceID               OPTIONAL,
  associatedCalledDevice    AssociatedCalledDeviceID                OPTIONAL,
  mediaCallCharacteristics  [4] IMPLICIT MediaCallCharacteristics   OPTIONAL,
  callCharacteristics       [5] IMPLICIT CallCharacteristics        OPTIONAL,
  queuedConnectionInfo      [6] IMPLICIT ConnectionInformation      OPTIONAL,
  callLinkageData           [7] IMPLICIT CallLinkageData            OPTIONAL,
  subjectOfCall             [8] IMPLICIT SubjectOfCall              OPTIONAL,
  messageInfo               [9] IMPLICIT MessageInfo                OPTIONAL,
  languagePreferences       [10] IMPLICIT LanguagePreferences       OPTIONAL,
  deviceHistory             [11] IMPLICIT DeviceHistory             OPTIONAL,
  locationInfo              [12] IMPLICIT LocationInfoList          OPTIONAL,
  extensions                CSTACommonArguments                     OPTIONAL}
```

Parameter Name	***Type***	***Description***
queuedConnection	ConnectionID	Specifies the queued connection.
Queue	SubjectDeviceID	Specifies the queued device.
callingDevice	CallingDeviceID	Specifies the calling device.
calledDevice	CalledDeviceID	Specifies the originally called device.
lastRedirectionDevice	RedirectionDeviceID	Specifies the previously known redirected from device.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the queueing device: Queued - For the other devices left in the call: (unaffected) This parameter is mandatory for events generated for device-type monitors.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Camp On - No Available Agents - Park
networkCallingDevice	NetworkCallingDeviceID	Specifies the original calling device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
networkCalledDevice	NetworkCalledDeviceID	Specifies the original called device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
associatedCallingDevice	AssociatedCallingDeviceID	Specifies the Network Interface Device associated with the calling device if the call is an external incoming call. This parameter is mandatory for all external incoming calls and shall not be provided otherwise.
associatedCalledDevice	AssociatedCalledDeviceID	For outgoing external calls, this parameter specifies the Network Interface Device associated with the originally called device. For incoming external calls, this parameter specifies a device within the switching sub-domain associated with the originally called device. This parameter is mandatory for all external outgoing calls and it is optional for external incoming calls.
~~callLinkageData~~	~~CallLinkageData~~	
Security	CSTASecurityData	Specifies timestamp information

2.6.2.16 Retrieved

Description

The Retrieved event indicates that a previously held call has been retrieved.
Common situations that generate this event include:

- When a held call is retrieved through the phone using features such as Retrieve, Alternate, etc.
- When a held call is retrieved during the successful execution of the Alternate Call, Reconnect Call, or the Retrieve Call service.

Operation

RetrievedEvent ::=

```
SEQUENCE
{ retrievedConnection        ConnectionID,
  retrievingDevice           SubjectDeviceID,
  localConnectionInfo        LocalConnectionState                OPTIONAL,
  correlatorData             [0] IMPLICIT CorrelatorData         OPTIONAL,
  cause                      EventCause,
  servicesPermitted          [1] IMPLICIT ServicesPermitted      OPTIONAL,
  mediaCallCharacteristics   [2] IMPLICIT MediaCallCharacteristics OPTIONAL,
  callCharacteristics        [3] IMPLICIT CallCharacteristics    OPTIONAL,
  retrievedConnectionInfo    [4] IMPLICIT ConnectionInformation  OPTIONAL,
  callLinkageData            [5] IMPLICIT CallLinkageData        OPTIONAL,
  languagePreferences        [6] IMPLICIT LanguagePreferences    OPTIONAL,
  locationInfo               [7] IMPLICIT LocationInfoList       OPTIONAL,
  extensions                 CSTACommonArguments                 OPTIONAL}
```

Parameter Name	***Type***	***Description***
retrievedConnection	ConnectionID	Specifies the connection at which hold was deactivated.
retrievingDevice	SubjectDeviceID	Specifies the device at which hold was deactivated.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the retrieving device: Connected - For the other devices left in the call: (unaffected) This parameter is mandatory for events generated for device-type monitors.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Alternate - Normal - Recall
~~callLinkageData~~	~~CallLinkageData~~	
security	CSTASecurityData	Specifies timestamp information

2.6.2.17 Service Initiated

Description

The Service Initiated event indicates that a telephony service has been initiated at a monitored device. The switching function typically generates this event when „dial-tone“ is being provided. This event indicates that either a call may be originated or a feature may be invoked. This event also may indicate that a device is prompting a user. Common situations that generate this event include:

- When a service or feature prompts a user to take a phone off-hook and that phone is not able to do so without manual intervention.
- A Make Call or Consultation Call service has been invoked and the originating device is initiating a new call that is associated with the originating device.
- When manually invoking any feature at a device for an existing call that requires a new call to be created to input the feature.
- When a device is taken off-hook manually.
- When an incoming call arrives on a monitored Network Interface Device (e.g. trunk, CO line).

Operation

ServiceInitiatedEvent ::=

```
SEQUENCE
{ initiatedConnection        ConnectionID,
  initiatingDevice           SubjectDeviceID,
  calledDevice CalledDeviceID                                      OPTIONAL,
  localConnectionInfo        LocalConnectionState                OPTIONAL,
  correlatorData             [0] IMPLICIT CorrelatorData         OPTIONAL,
  cause                      EventCause,
```

~~servicesPermitted~~	~~[1] IMPLICIT ServicesPermitted~~	~~OPTIONAL,~~
~~mediaCallCharacteristics~~	~~[2] IMPLICIT MediaCallCharacteristics~~	~~OPTIONAL,~~
~~callCharacteristics~~	~~[3] IMPLICIT CallCharacteristics~~	~~OPTIONAL,~~
~~initiatedConnectionInfo~~	~~[4] IMPLICIT ConnectionInformation~~	~~OPTIONAL,~~
networkCallingDevice	NetworkCallingDeviceID	OPTIONAL,
networkCalledDevice	NetworkCalledDeviceID	OPTIONAL,
associatedCallingDevice	AssociatedCallingDeviceID	OPTIONAL,
~~callLinkageData~~	~~[5] IMPLICIT CallLinkageData~~	~~OPTIONAL,~~
~~subjectOfCall~~	~~[7] IMPLICIT SubjectOfCall~~	~~OPTIONAL,~~
~~messageInfo~~	~~[8] IMPLICIT MessageInfo~~	~~OPTIONAL,~~
~~languagePreferences~~	~~[9] IMPLICIT LanguagePreferences~~	~~OPTIONAL,~~
~~locationInfo~~	~~[10] IMPLICIT LocationInfoList~~	~~OPTIONAL,~~
extensions	CSTACommonArguments	OPTIONAL}

Parameter Name	***Type***	***Description***
initiatedConnection	ConnectionID	Specifies the connection at which service was initiated.
initiatingDevice	SubjectDeviceID	Specifies the initiating device.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - When the device is initiating a service of some form: Initiated This parameter is mandatory for events generated for device-type monitors.
Cause	EventCause	Specifies the reason for the event. Supported causes are: - Call Back - Consultation - Make Call - New Call
networkCallingDevice	NetworkCallingDeviceID	Specifies the original calling device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
networkCalledDevice	NetworkCalledDeviceID	Specifies the original called device information provided by the network for external incoming calls. This parameter is provided only for external incoming calls.
associatedCallingDevice	AssociatedCallingDeviceID	Specifies the Network Interface Device associated with the calling device if the call is an external incoming call. This parameter is mandatory for all external incoming calls and shall not be provided otherwise.
~~callLinkageData~~	~~CallLinkageData~~	
security	CSTASecurityData	Specifies timestamp information

2.6.2.18 Transferred

Description

The Transferred event indicates that an existing call has been transferred to another device and the transferring device has been dropped from the call. The transferring device does not appear in any future events for the call.
Common situations that generate this event include:

- Two step transferring situations (manual and service initiated).
- Single step transferring situations (manual and service initiated).

Note
The transferred event might provide a new Call-ID depending on the capabilities of the executed transfer type (see examples in part 3).

Operation

TransferredEvent ::=

```
SEQUENCE
{   primaryOldCall          ConnectionID,
    secondaryOldCall        [0] IMPLICIT ConnectionID          OPTIONAL,
    transferringDevice      SubjectDeviceID,
    transferredToDevice     SubjectDeviceID,
    transferredConnections  [1] IMPLICIT ConnectionList,
    localConnectionInfo     LocalConnectionState               OPTIONAL,
    correlatorData          [2] IMPLICIT CorrelatorData        OPTIONAL,
    userData                UserData                           OPTIONAL,
    chargingInfo            [3] IMPLICIT ChargingInfo          OPTIONAL,
    cause                   EventCause,
    servicesPermitted       [4] IMPLICIT ServicesPermitted     OPTIONAL,
    mediaCallCharacteristics [5] IMPLICIT MediaCallCharacteristics OPTIONAL,
    callCharacteristics     [6] IMPLICIT CallCharacteristics   OPTIONAL,
    callLinkageDataList     [7] IMPLICIT CallLinkageDataList   OPTIONAL,
    languagePreferences     [ 8] IMPLICIT LanguagePreferences  OPTIONAL,
    deviceHistory           [ 9] IMPLICIT DeviceHistory        OPTIONAL,
    locationInfo            [10] IMPLICIT LocationInfoList     OPTIONAL,
    extensions              CSTACommonArguments                OPTIONAL}
```

Parameter Name	***Type***	***Description***
primaryOldCall	ConnectionID	Specifies the connection of the primary call.
secondaryOldCall	ConnectionID	Because the switching function supports the „local-view“ option, this parameter is mandatory if there are two known calls involved with the transfer (before the transfer is created) from the perspective of the monitored device, otherwise it shall not be provided.
transferringDevice	SubjectDeviceID	Specifies the device that transferred the call.
transferredToDevice	SubjectDeviceID	Specifies the transferred to device.
transferredConnections	ConnectionList	Specifies information on each device/ConnectionID in the resulting transferred call that are known by the switching function.
localConnectionInfo	LocalConnectionState	Specifies the local connection state of the device associated with the Monitor Cross Reference ID. The following are the states of the devices in the call. - For the transferring device (any Connection IDs associated with the transfer, i.e., this event should be used for both single and multi-step transfers.): Null - For the other devices associated with the transfer: (unaffected) This parameter is mandatory for events generated for device-type monitors.
cause	EventCause	Specifies the reason for the event. Supported causes are: - Single Step Transfer - Transfer
~~callLinkageDataList~~	~~CallLinkageDataList~~	
security	CSTASecurityData	Specifies timestamp information

2.7 Call Associated Features

Note

The Call Associated Features with the exception of the Charging Events are not supported for S_0-Devices and SIP-Endpoints.

2.7.1 Services

~~*2.7.1.1 Associate Data*~~

Not supported

~~*2.7.1.2 Cancel Telephony Tones*~~

Not supported

2.7.1.3 Generate Digits

Description

The Generate Digits service causes a series of digits to be sent on behalf of a connection in a call. The digits may be sent in the form of DTMF tones or rotary pulses. This service also supports optional parameters to control digit generation. This service is used for generating end-to-end information that is to be sent to a device in a call (i.e., not to address/select a device). This service does not affect the state or progress of a call.

Note

The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID).

Operation

```
generateDigits                  OPERATION ::=
{       ARGUMENT                GenerateDigitsArgument
        RESULT                  GenerateDigitsResult
        ERRORS                  {universalFailure}
        CODE                    local: 232
}
```

Service request

GenerateDigitsArgument ::= SEQUENCE

```
{       connectionToSendDigits  ConnectionID,
        digitMode               DigitMode                       OPTIONAL,
        charactersToSend        IA5String,
        toneDuration            [0] IMPLICIT INTEGER            OPTIONAL,
        pulseRate               [1] IMPLICIT INTEGER            OPTIONAL,
        pauseDuration           [2] IMPLICIT INTEGER            OPTIONAL,
        extensions              CSTACommonArguments             OPTIONAL}
```

```
DigitMode ::= ENUMERATED
{       rotaryPulse     (0),
        dTMF    (1)}
```

Parameter Name	*Type*	*Description*
connectionToSendDigits	ConnectionID	Connection of the device which is generating the digits for the call.
digitMode	Enumerated	Specifies the signalling format. The complete set of possible values is: ~~- rotaryPulse - rotary pulse signalling.~~ - DTMF - DTMF signalling (default).
charactersToSend	Characters (26)	Specifies the string of characters to send. Shall consist of the following set: - For DTMF digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #, A, B, C, D This parameter type is a character string. The maximum length supported by the switching function is provided via the capabilities exchange services. The maximum length is limited to 26.

Service response

```
GenerateDigitsResult ::=
        CHOICE
        {extensions             CSTACommonArguments,
         noData                 NULL}
```

2.7.1.4 Generate Telephony Tones

Description

The Generate Telephony Tones service causes a telephony tone such as a beep, busy, or ringback to be sent on behalf of a connection in a call. This service also supports optional parameters to control tone generation. This service does not affect the state or progress of a call.

Note

The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID).

Operation

```
generateTelephonyTones          OPERATION ::=
{       ARGUMENT                GenerateTelephonyTonesArgument
        RESULT                  GenerateTelephonyTonesResult
        ERRORS                  {universalFailure}
        CODE                    local: 232
}
```

Service request

GenerateTelephonyTonesArgument ::= SEQUENCE

```
{       connectionToSendTone    ConnectionID,
        toneToSend              Enumeration,
        toneDuration            [0] IMPLICIT INTEGER            OPTIONAL,
        security                CSTASecuritydata                OPTIONAL,
        privateParameter        CSTAPrivatedata                 OPTIONAL}
```

```
TonetoSend ::= ENUMERATED
{     beep, confirmation, ringback, silence)}
```

Parameter Name	*Type*	*Description*
connectionToSendDigits	ConnectionID	Connection of the device which is generating the digits for the call.
toneToSend	Enumerated	Specifies the signalling format. The complete set of possible values is: beep confirmation ringback silence.
toneDuration	Value	Specifies the total length in milliseconds of the tones to be provided. A value of 0 indicates that the tone should be continuous..
~~security~~	~~CSTASecurityData~~	~~Specifies timestamp information, message sequence number, and security information.~~
~~privateData~~	~~CSTAPrivateData~~	~~Specifies non-standardized information.~~

Service response

```
GenerateTelephonyTonesResult ::=
        CHOICE
        {extensions             CSTACommonArguments,
         noData                 NULL}
```

~~2.7.1.5 Send User Information~~

Not supported

2.7.2 Events

Note
The maximum amount of devices that can be monitored at the same time is stated in chapter 2.18.1 OpenScape Business parameter contraints.
There is no event report on non-voice calls.
See also 2.18.3 List of monitorable OpenScape Business Devices.

~~*2.7.2.1 Call Information*~~
Not supported

2.7.2.2 Charging
The Charging event indicates that call charging information has been detected for a device involved in a call. This event may occur after the call has been released, to report arrival of additional call charge information.

ChargingEvent ::= SEQUENCE

```
{   connection          ConnectionID,
    chargedDevice       DeviceID,
    chargingInfo        ChargingInfo,
    cause               EventCause                  OPTIONAL,
    extensions          CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
connection	ConnectionID	Indicates the call for which the device is being charged.
chargedDevice	DeviceID	Indicates the device that is being charged. If this device is not specified then this parameter indicates Unknown or Not Required.
chargingInfo	ChargingInfo	Indicates an intermediate or total value of charging or currency units for the device being charged.
security	CSTASecurityData	Indicates timestamp information

~~*2.7.2.3 Digits Generated*~~
Not supported

~~*2.7.2.4 Telephony Tones Generated*~~
Not supported

~~*2.7.2.5 Service Completion Failure*~~
Not supported

~~2.8 Media Attachment Services and Events~~
The Media Attachment Services and Events are not supported.

~~2.8.1 Services~~

~~*2.8.1.1 Attach Media Service*~~
Not supported

~~*2.8.1.2 Detach Media Service*~~
Not supported

~~2.8.2 Events~~

~~*2.8.2.1 Media Attached*~~
Not supported

~~*2.8.2.2 Media Detached*~~
Not supported

~~2.9 Routeing Services~~
The Routing services are not provided.

~~2.9.1 Registration Services~~
Not supported

~~*2.9.1.1 Route Register Abort*~~
Not supported

~~*2.9.1.2 Route Register Cancel*~~
Not supported

~~2.9.2 Services~~

~~*2.9.2.1 Re-Route*~~
Not supported

~~*2.9.2.2 Route End*~~
Not supported

~~*2.9.2.3 Route Reject*~~
Not supported

~~*2.9.2.4 Route Request*~~
Not supported

~~*2.9.2.5 Route Select*~~
Not supported

~~*2.9.2.6 Route Used*~~
Not supported

2.10 Physical Device Features

Note
The Physical Device Features are not supported for S_0-Devices and SIP-Endpoints.

2.10.1 Services
For the mapping of buttons and lamps (LED) see chapter 2.17.4 Button and Lamp map.

2.10.1.1 Button Press

Description

The Button Press service allows a computing function to simulate the activation of a specified button at a specified device.

Operation

```
buttonPress OPERATION ::=
{       ARGUMENT                        ButtonPressArgument
        RESULT                          ButtonPressResult
        ERRORS                          {universalFailure}
        CODE                            local: 260
}
```

Service request

```
ButtonPressArgument ::= SEQUENCE
{       device                          DeviceID,
        button                          ButtonID,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device's physical element.
Button	ButtonID	Specifies the button on the device.

Service response

```
ButtonPressResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

~~*2.10.1.2 Get Auditory Apparatus Information*~~

Not supported

2.10.1.3 Get Button Information

Description

The Get Button Information service provides information about the specified button or buttons on a device.

Note

- The Get Button Information response is only generated for Level 1 of the programmable keys. Level 2 (accessed via the Shift key, contains only external callnumbers) is not reported.
- The Get Button Information service is not supported for virtual and analog Devices. In these cases a buttonNotSupported error is returned.

Operation

```
getButtonInformation OPERATION ::=
{       ARGUMENT                        GetButtonInformationArgument
        RESULT                          GetButtonInformationResult
        ERRORS                          {universalFailure}
        CODE                            local: 262
}
```

Service request

```
GetButtonInformationArgument ::= SEQUENCE
{       device                          DeviceID,
        button                          ButtonID                                OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	Type	Description
device	DeviceID	Specifies the device's physical element.
Button	ButtonID	Specifies which button to query. If not provided, then information is obtained on all the buttons associated with the device.

Service response

```
GetButtonInformationResult ::= SEQUENCE
{       buttonList                          ButtonList,
        ~~extensions~~                      ~~CSTACommonArguments~~                  ~~OPTIONAL}~~

ButtonList ::= SEQUENCE OF SEQUENCE
{       button                              ButtonID,
        ~~buttonLabel~~                     ~~[0] IMPLICIT IA5String~~               ~~OPTIONAL,~~
        ~~buttonLabelSettable~~             ~~[1] IMPLICIT BOOLEAN~~                 ~~OPTIONAL,~~
        buttonFunction                      [2] IMPLICIT IA5String                  OPTIONAL,
        buttonAssociatedNumber              [3] IMPLICIT DeviceID                   OPTIONAL *)
        ~~buttonAssociatedNumberSettable~~  ~~[4] IMPLICIT BOOLEAN~~                 ~~OPTIONAL,~~
        ~~buttonPressIndicator~~            ~~[5] IMPLICIT BOOLEAN~~                 ~~OPTIONAL,~~
        ~~lampList~~                        ~~SEQUENCE OF LampID~~                   ~~OPTIONAL}~~
        *) , --only when the request belongs to a single button
```

Parameter Name	Type	Description
buttonList	List of Structures	Specifies information about the specified button or about all buttons associated with the device if no button was provided in the request. Each entry contains the following: - button (M) ButtonID - Specifies the buttonID of the button described by this entry. - buttonFunction (O) Characters - The function assigned to the button. The buttonFunction is represented as an digit string (size 3). For the mapping to the feature see the Sample Keyfeatures (see 2.17.2). - buttonAssociatedNumber (O) DeviceID - This indicates a number (in Diallable Digits format) associated with this button.

~~2.10.1.4 Get Display~~

Not supported

2.10.1.5 Get Hookswitch Status

Description

This Get Hookswitch Status service provides the hookswitch (handset) status associated with a specified device. The hookswitch status indicates which hookswitches are present on a device and which hookswitches are currently off-hook.

Operation

```
getHookswitchStatus OPERATION ::=
{       ARGUMENT                            GetHookswitchStatusArgument
        RESULT                              GetHookswitchStatusResult
        ERRORS                              {universalFailure}
        CODE                                local: 264
}
```

Service request

```
GetHookswitchStatusArgument ::= SEQUENCE
{       device                              DeviceID,
        ~~hookswitch~~                      ~~HookswitchID~~                         ~~OPTIONAL,~~
        ~~extensions~~                      ~~CSTACommonArguments~~                  ~~OPTIONAL}~~
```

Parameter Name	Type	Description
device	DeviceID	Specifies the device's physical element.

Service response

```
GetHookswitchStatusResult ::= SEQUENCE
{       hookswitchStatusList            HookswitchStatusList,
        extensions                      CSTACommonArguments                     OPTIONAL}

HookswitchStatusList ::= SEQUENCE OF SEQUENCE
{       hookswitch                      HookswitchID,                           -- always: 1
        hookswitchOnHook                BOOLEAN}
```

Parameter Name	Type	Description
hookswitchStatusList	List of Structures	Specifies information about the specified hookswitch or about all hookswitches associated with the device if no hookswitch was provided in the request. Each entry contains the following: - hookswitch (M) HookswitchID - always 1 (handset). - hookswitchOnHook (M) Boolean - Indicates the state of the hookswitch. The complete set of possible values is: - TRUE - Indicates that the switch is onhook. - FALSE - Indicates that the switch is offhook.

~~2.10.1.6 Get Lamp Information~~

Not supported

~~2.10.1.7 Get Lamp Mode~~

Not supported

2.10.1.8 Get Message Waiting Indicator

Description

The Get Message Waiting Indicator service provides the message waiting feature status at a specified device. The message waiting feature is typically used to notify a user (typically via a dedicated lamp on a phone device) when messages are available.

Operation

```
getMessageWaitingIndicator OPERATION ::=
{       ARGUMENT                        GetMessageWaitingIndicatorArgument
        RESULT                          GetMessageWaitingIndicatorResult
        ERRORS                          {universalFailure}
        CODE                            local: 267
}
```

Service request

```
GetMessageWaitingIndicatorArgument ::= SEQUENCE
{       device                          DeviceID,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	Type	Description
device	DeviceID	Specifies the device's physical element.

Service response

```
GetMessageWaitingIndicatorResult ::= SEQUENCE
{       messageWaitingOn                BOOLEAN,
        deviceForMessage                DeviceID                                OPTIONAL,
        lampIsPresent                   BOOLEAN                                 OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
messageWaitingOn	Boolean	Specifies the value of the requested feature. The complete set of possible values is: - FALSE - Message waiting off. - TRUE - Message waiting on.

~~2.10.1.9 Get Microphone Gain~~

Not supported

2.10.1.10 Get Microphone Mute

Description

The Get Microphone Mute service provides the microphone mute feature status of each microphone actual used for a call. While a devices microphone is muted, no audio information is transmitted over the device microphone. This feature is used when it is desired to prevent the other party(s) in a call from hearing a conversation through the device.

Operation

```
getMicrophoneMute OPERATION ::=
{       ARGUMENT                        GetMicrophoneMuteArgument
        RESULT                          GetMicrophoneMuteResult
        ERRORS                          {universalFailure}
        CODE                            local: 269
}
```

Service request

```
GetMicrophoneMuteArgument ::= SEQUENCE
{       device                          DeviceID,
        auditoryApparatus               AuditoryApparatusID                     OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device's physical element.

Service response

```
GetMicrophoneMuteResult ::= SEQUENCE
{       microphoneMuteList              MicrophoneMuteList,
        extensions                      CSTACommonArguments                     OPTIONAL}

MicrophoneMuteList ::= SEQUENCE OF SEQUENCE
{       auditoryApparatus               AuditoryApparatusID,                    -- always 1
        microphoneMuteOn                BOOLEAN}
```

Parameter Name	Type	Description
microphoneMuteList	List of Structures	Specifies information about the specified auditory apparatus or about all auditory apparatuses associated with the device if no auditoryApparatus was provided in the request. Each entry contains the following: - auditoryApparatus (M) AuditoryApparatusID - ~~Specifies the auditory apparatus that the microphone belongs to.~~ always 1 - microphoneMuteOn (M) Boolean - Specifies whether the microphone is muted or not. The complete set of possible values is: - FALSE - Microphone is activated. - TRUE - Microphone is muted.

~~2.10.1.11 Get Ringer Status~~

Not supported

2.10.1.12 Get Speaker Mute

Description

The Get Speaker Mute service provides the information if the speaker of the phone is activ. While a device's speaker is muted, no audio information is transmitted over the speaker.

Operation

```
getSpeakerMute OPERATION ::=
{       ARGUMENT                        GetSpeakerMuteArgument
        RESULT                          GetSpeakerMuteResult
        ERRORS                          {universalFailure}
        CODE                            local: 271
}
```

Service request

```
GetSpeakerMuteArgument ::= SEQUENCE
{       device                          DeviceID,
        auditoryApparatus               AuditoryApparatusID                     OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	Type	Description
device	DeviceID	Specifies the devices physical element.

Service response

```
GetSpeakerMuteResult ::= SEQUENCE
{       speakerMuteList                 SpeakerMuteList,
        extensions                      CSTACommonArguments             OPTIONAL}

SpeakerMuteList ::= SEQUENCE OF SEQUENCE
{       auditoryApparatus               AuditoryApparatusID,        -- always 1
        speakerMuteOn                   BOOLEAN}
```

Parameter Name	Type	Description
speakerMuteList	List of Structures	Specifies information about the specified auditory apparatus or about all auditory apparatuses associated with the device if no auditoryApparatus was provided in the request. Each entry contains the following: - auditoryApparatus (M) AuditoryApparatusID - Specifies the auditory apparatus to which this speaker belongs. - speakerMuteOn (M) Boolean - Specifies whether the speaker mute setting is on or not. The complete set of possible values is: - FALSE - Mute is off (i.e., speaker is activated). - TRUE - Mute is on (i.e., speaker is muted).

2.10.1.13 Get Speaker Volume

Description

The Get Speaker Volume service provides the speaker volume setting for the speakers associated with one or more auditory apparatuses at a specified device.

Operation

```
getSpeakerVolume OPERATION ::=
{       ARGUMENT                GetSpeakerVolumeArgument
        RESULT                  GetSpeakerVolumeResult
        ERRORS                  {universalFailure}
        CODE                    local: 272
}
```

Service request

```
GetSpeakerVolumeArgument ::= SEQUENCE
{       device                  DeviceID,
        auditoryApparatus       AuditoryApparatusID         OPTIONAL,
        extensions              CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device's physical element.
auditoryApparatus	AuditoryApparatusID	0 = Handset (default volume) 1 = Handset (actual volume) 2 = Speaker (default volume) 3 = Speaker (actual volume)

Service response

```
GetSpeakerVolumeResult ::= SEQUENCE
{       speakerVolumeList       SpeakerVolumeList,
        extensions              CSTACommonArguments         OPTIONAL}

SpeakerVolumeList ::= SEQUENCE OF SEQUENCE
{       auditoryApparatus       AuditoryApparatusID,
        speakerVolAbs           VolAbs                      OPTIONAL}
```

Parameter Name	*Type*	*Description*
speakerVolumeList	List of Structures	Specifies information about the specified auditory apparatus or about all auditory apparatuses associated with the device if no auditoryApparatus was provided in the request. Each entry contains the following: - auditoryApparatus (M) AuditoryApparatusID - Specifies the auditory apparatus to which this speaker belongs. - speakerVolAbs (O) Value - Specifies the absolute speaker volume. A value of 0 indicates silence, and 100 indicates maximum volume. The granularity and quantization of the values 1 though 99 are device specific. If this component is not provided, the absolute speaker volume is unknown.

~~2.10.1.14 Set Button Information~~

Not supported

2.10.1.15 Set Display

Description

The Set Display service allows the computing function to set a display associated with a device. The capabilities exchange services can be used to determine the number of displays associated with a device.

Operation

```
setDisplay OPERATION ::=
{       ARGUMENT                        SetDisplayArgument
        RESULT                          SetDisplayResult
        ERRORS                          {universalFailure}
        CODE                            local: 274
}
```

Service request

```
SetDisplayArgument ::= SEQUENCE
{       device                          DeviceID,
        display                         DisplayID                       OPTIONAL,
        physicalBaseRowNumber           [0] IMPLICIT INTEGER            OPTIONAL,
        physicalBaseColumnNumber        [1] IMPLICIT INTEGER            OPTIONAL,
        contentsOfDisplay               IA5String (SIZE(0..24)),
        offset                          [2] IMPLICIT INTEGER            OPTIONAL,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Parameter Name	***Type***	***Description***
device	DeviceID	Specifies the devices physical element.
contentsOfDisplay	Characters (24)	Specifies the 7bit-ASCII (IA5) text to place on the display as a string of characters consisting of the text on each row of the display (including spaces) concatenated together. If a null string is sent, the display will be cleared. Sending of a 0x7f value will also clear the display. Starting a String with a BEL will cause a notification ring at the device.

Service response

```
SetDisplayResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

~~2.10.1.16 Set Hookswitch Status~~

Not supported

2.10.1.17 Set Lamp Mode

Description

The Set Lamp Mode service allows a computing function to control how a specified lamp is lit at a specified device.

Operation

```
setLampMode OPERATION ::=
{       ARGUMENT                        SetLampModeArgument
        RESULT                          SetLampModeResult
        ERRORS                          {universalFailure}
        CODE                            local: 276
}
```

Service request

```
SetLampModeArgument ::= SEQUENCE
{       device                          DeviceID,
        lamp                            LampID,
        lampMode                        LampMode,
        lampBrightness                  LampBrightness                  OPTIONAL,
        lampColor                       LampColor                       OPTIONAL,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device's physical element.
Lamp	LampID	Specifies the LampID of the lamp on the device.
lampMode	Integer -- ISO	Specifies how the lamp associated with the specified device should be lit. The complete set of possible values is: - flutter - Fast on and off. - off - Lamp is off. - steady - Lamp is continuously lit. - wink - Lamp is winking (slowly on/off).

Service response

```
SetLampModeResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

2.10.1.18 Set Message Waiting Indicator

Description

The Set Message Waiting Indicator service allows a computing function to control the status of the message waiting feature at a specified device. The message waiting feature is typically used to notify a user (typically via a dedicated lamp on a phone device) when messages are available.

Note

Using the feature "setMessageWaitingIndicator" the receiver gets a message related to the attendant position, although it has been sent via CSTA. The default text that is taken by this message is no. 9. It can be administrated via OpenScape Business Assistant.

Operation

```
setMessageWaitingIndicator OPERATION ::=
{       ARGUMENT                        SetMessageWaitingIndicatorArgument
        RESULT                          SetMessageWaitingIndicatorResult
        ERRORS                          {universalFailure}
        CODE                            local: 277
}
```

Service request

```
SetMessageWaitingIndicatorArgument ::= SEQUENCE
{       device                          DeviceID,
        messageWaitingOn                BOOLEAN,
        deviceForMessage                DeviceID OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device's physical element.
messageWaitingOn	Boolean	Specifies the setting of the message waiting feature. The complete set of possible values is: - OFF - Message waiting off. - ON - Message waiting on.

Service response

```
SetMessageWaitingIndicatorResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

~~2.10.1.19 Set Microphone Gain~~

Not supported

2.10.1.20 Set Microphone Mute

Description

The Set Microphone Mute service allows the computing function to control the microphone mute status of the microphone associated with one auditory apparatus at a specified device. While a device's microphone is muted, no audio information is transmitted over the microphone. This is used when it is desired to prevent the other device(s) in a call from hearing a conversation.

Note

State transition table:

	Handset on-hook **Micro off**	**Handset off-hook** **Micro on**	**Handset off-hook** **Micro off**	**Handsfree active** **Micro on**	**Handsfree active** **Micro off**
SetMicrophoneMute microphoneMuteOn =**TRUE**	positive Acknowledgement to Application; Micro remains off	positive Acknowledgement to Application; Micro switched off	positive Acknowledgement to Application; Micro remains off	positive Acknowledgement to Application; Micro switched off	positive Acknowledgement to Application; Micro remains off
SetMicrophoneMute microphoneMuteOn = **FALSE**	*negative* Acknowledgement to Application; Micro remains off	positive Acknowledgement to Application; Micro remains on	positive Acknowledgement to Application; Micro switched on	positive Acknowledgement to Application; Micro remains on	positive Acknowledgement to Application; Micro switched on

Operation

```
setMicrophoneMute OPERATION ::=
{       ARGUMENT                        SetMicrophoneMuteArgument
        RESULT                          SetMicrophoneMuteResult
        ERRORS                          {universalFailure}
        CODE                            local: 279
}
```

Service request

```
SetMicrophoneMuteArgument ::= SEQUENCE
{       device                          DeviceID,
        auditoryApparatus               AuditoryApparatusID,
        microphoneMuteOn                BOOLEAN,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	***Type***	***Description***
device	DeviceID	Specifies the device's physical element.
auditoryApparatus	AuditoryApparatusID	This parameter is ignored, the service belongs to the actual used microphone
microphoneMuteOn	Boolean	Specifies the microphone mute setting of a particular microphone. The complete set of possible values is: - OFF - Microphone is activated. - ON - Microphone is muted.

Service response

```
SetMicrophoneMuteResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

~~2.10.1.21 Set Ringer Status~~

Not supported

2.10.1.22 Set Speaker Mute

Description

The Set Speaker Mute service allows a computing function to control the speaker mute status of the hookswitch components at a specified device. While a devices speaker is muted, no audio information is transmitted over the speaker.

Note

State transition table:

	Handset off-hook **Speaker off**	**Handset on-hook** **Speaker off**	**Handset off-hook** **Speaker on**	**Handset on-hook** **Speaker on**
SetSpeakerMute speakerMuteOn =**TRUE**	positive Acknowledgement to Application; Speaker remains off	positive Acknowledgement to Application; Speaker remains off	positive Acknowledgement to Application; Speaker switched off	*negative* Acknowledgement to Application; Speaker remains on
SetSpeakerMute speakerMuteOn = **FALSE**	positive Acknowledgement to Application; Speaker switched on	*negative* Acknowledgement to Application; Speaker remains off	positive Acknowledgement to Application; Speaker remains on	positive Acknowledgement to Application; Speaker remains on

Operation

```
setSpeakerMute OPERATION ::=
{       ARGUMENT                        SetSpeakerMuteArgument
        RESULT                          SetSpeakerMuteResult
        ERRORS                          {universalFailure}
        CODE                            local: 281
}
```

Service request

```
SetSpeakerMuteArgument ::= SEQUENCE
{       device                          DeviceID,
        auditoryApparatus               AuditoryApparatusID,
        speakerMuteOn                   BOOLEAN,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device's physical element.
auditoryApparatus	AuditoryApparatusID	This parameter is ignored,
speakerMuteOn	Boolean	Specifies the speaker mute setting of a particular speaker at a device. The complete set of possible values is: - FALSE - Mute is off (i.e., speaker is activated). - TRUE - Mute is on (i.e., speaker is muted).

Service response

```
SetSpeakerMuteResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

2.10.1.23 Set Speaker Volume

Description

The Set Speaker Volume service allows the computing function to control the speaker volume of the speaker associated with one auditory apparatus at a specified device.

Operation

```
setSpeakerVolume OPERATION ::=
{       ARGUMENT                        SetSpeakerVolumeArgument
        RESULT                          SetSpeakerVolumeResult
        ERRORS                          {universalFailure}
        CODE                            local: 282
}
```

Service request

```
SetSpeakerVolumeArgument ::= SEQUENCE
{       device                          DeviceID,
        auditoryApparatus               AuditoryApparatusID,
        speakerVolume                   Volume,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device's physical element.
auditoryApparatus	AuditoryApparatusID	0 = Handset (default volume) 1 = Handset (actual volume) 2 = Speaker (default volume) 3 = Speaker (actual volume)
speakerVolume	Choice Structure	Specifies the speaker volume as an absolute value or that the volume should be incremented or decremented by a switch specified increment. It may be one of the following possible choices: - 12 - 25 - 37 - 50 - 62 - 75 - 87 - 100 other choices are set to next higher value

Service response

```
SetSpeakerVolumeResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

2.10.2 Events

Note
The maximum amount of devices that can be monitored at the same time is stated in chapter OpenScape Business Parameter Constraints. There is no event report on non-voice calls. See also 2.18.3 List of monitorable OpenScape Business Devices.

~~*2.10.2.1 Button Information*~~
Not supported

2.10.2.2 Button Press

Description

The Button Press event indicates that a button has been pressed. This event may be generated in any one of the following ways:

- The button has been pressed manually.
- A computing function, on behalf of a user, has invoked the Button Press service.

Note
The Button Press event is only generated for Level 1 of the programmable keys. Level 2 (accessed via the Shift key, contains only external callnumbers) is not reported.

- Event report is only generated for ButtonIDs starting with 0x12, that means that events for e.g. dialpad keys are not generated.

Operation

```
ButtonPressEvent ::= SEQUENCE
{       device                          SubjectDeviceID,
        button                          ButtonID,
        buttonLabel                     IA5String (SIZE(0..64))         OPTIONAL,
        buttonAssociatedNumber          DeviceID                        OPTIONAL,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Parameter Name	***Type***	***Description***
device	SubjectDeviceID	Specifies the device where the button press feature was changed.
Button	ButtonID	Specifies the button that was pressed.
buttonAssociatedNumber	DeviceID	Indicates a number (in diallable string format) associated with the button.
security	CSTASecurityData	Specifies timestamp information

~~*2.10.2.3 Display Updated*~~
Not supported

2.10.2.4 Hookswitch

Description

The Hookswitch event indicates that the hookswitch status (on-hook/off-hook) has changed. This event may be generated in any one of the following ways:

- The computing function has invoked the Set Hookswitch Status service.
- The hookswitch status has been changed manually.
- The computing function has invoked a call control service that indirectly affected the hookswitch.

Operation

```
HookswitchEvent ::= SEQUENCE
{       device                  SubjectDeviceID,
        hookswitch              HookswitchID,
        hookswitchOnHook        BOOLEAN,
        extensions              CSTACommonArguments             OPTIONAL}
```

Parameter Name	***Type***	***Description***
Device	SubjectDeviceID	Specifies the device where the feature was changed.
hookswitch	HookswitchID	Always 1 (handset)
hookswitchOnHook	Boolean	Specifies the state of the hookswitch. The complete set of possible values is: - On-hook - The switch is open and the hookswitch is inactive. - Off-hook - The switch is closed and the hookswitch is active.
Security	CSTASecurityData	Specifies timestamp information

~~2.10.2.5 Lamp Mode~~

Not supported

2.10.2.6 Message Waiting

Description

The Message Waiting event indicates that the message waiting status has been changed for a device. This event may be generated in any one of the following ways:

- The message waiting feature has been changed on the telephone.
- A computing function, on behalf of a user, has invoked the Set Message Waiting service.

Operation

```
MessageWaitingEvent ::= SEQUENCE
{       targetDevice            SubjectDeviceID,
        deviceForMessage        DeviceID                        OPTIONAL,
        messageWaitingOn        BOOLEAN,
        extensions              CSTACommonArguments             OPTIONAL}
```

Parameter Name	Type	Description
targetDevice	SubjectDeviceID	Specifies the device where the message waiting feature has changed.
deviceForMessage	DeviceID	Specifies the device where the message is waiting.
messageWaitingOn	Boolean	Specifies the setting of the message waiting feature. The complete set of possible values is: - FALSE - Message waiting off. - TRUE - Message waiting on.
security	CSTASecurityData	Specifies timestamp information

~~2.10.2.7 Microphone Gain~~
Not supported

~~2.10.2.8 Microphone Mute~~
Not supported

~~2.10.2.9 Ringer Status~~
Not supported

~~2.10.2.10 Speaker Mute~~
Not supported

~~2.10.2.11 Speaker Volume~~
Not supported

2.11 Logical Device Features

Note
The Logical Device Features are not supported for S_0-Devices and SIP-Endpoints.

2.11.1 Services

~~2.11.1.1 Call Back Non-Call-Related~~
Not supported

2.11.1.2 Call Back Message Non-Call-Related

Description

The Call Back Message Non-Call-Related service allows a computing function to instruct the switching function to leave a pre-defined message requesting that the target device call the originating device.

Operation

```
callBackMessageNonCallRelated OPERATION ::=
{       ARGUMENT                        CallBackMessageNonCallRelatedArgument
        RESULT                          CallBackMessageNonCallRelatedResult
        ERRORS                          {universalFailure}
        CODE                            local: 301
}
```

Service request

```
CallBackMessageNonCallRelatedArgument ::= SEQUENCE
{       originatingDevice               DeviceID,
        targetDevice                    DeviceID,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
originatingDevice	deviceID	Specifies the originating device for the call back message.
targetDevice	deviceID	Specifies the target device for the call back message.

Service response

```
CallBackMessageNonCallRelatedResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

~~2.11.1.3 Cancel Call Back~~

Not supported

2.11.1.4 Cancel Call Back Message

Description

The Cancel Call Back Message service allows the computing function to cancel a previous (or all) Call Back Message feature at a device.

Note:

- that this service cancels call back messages that were created with either call related or non-call related Call Back Message features.
- The service clears all messages of the originatingDevice to the targetDevice.

Operation

```
cancelCallBackMessage OPERATION ::=
{       ARGUMENT                        CancelCallBackMessageArgument
        RESULT                          CancelCallBackMessageResult
        ERRORS                          {universalFailure}
        CODE                            local: 303
}
```

Service request

```
CancelCallBackMessageArgument ::= SEQUENCE
{       originatingDevice               DeviceID,
        targetDevice                    DeviceID,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
originatingDevice	DeviceID	The DeviceID of the party who initiated the original Call Back Message service.
targetDevice	DeviceID	The DeviceID of the target of the original Call Back Message service. If the switching function supports clearing of all Call Back Message features (as indicated by the capability exchange services) and a null format DeviceID (a DeviceID with 0 characters) is provided, all of the Call Back Message features at the originatingDevice are cancelled.

Service response

```
CancelCallBackMessageResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

2.11.1.5 Get Agent State

Description

The Get Agent State service provides the agent state at a specified device.

Operation

```
getAgentState OPERATION ::=
{       ARGUMENT                        GetAgentStateArgument
        RESULT                          GetAgentStateResult
        ERRORS                          {universalFailure}
```

```
        CODE                            local: 304
}
```

Service request

```
GetAgentStateArgument ::= SEQUENCE
{       device                          DeviceID,
        acdGroup                        DeviceID                        OPTIONAL,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the DeviceID of the device on which the agent state is being queried.

Service response

```
GetAgentStateResult ::= SEQUENCE
{       agentStateList                  AgentStateList,
        extensions                      CSTACommonArguments             OPTIONAL}

AgentStateList ::= SEQUENCE SIZE (1..32) OF AgentStateEntry

AgentStateEntry ::= SEQUENCE
{       agentID                         AgentID                         OPTIONAL,
        loggedOnState                   BOOLEAN,
        agentInfo                       SEQUENCE OF AgentInfo           OPTIONAL}


      AgentInfo ::= SEQUENCE
{       acdGroup                        DeviceID                        OPTIONAL,
        agentState                      AgentState,
        pendingAgentState               [0] IMPLICIT PendingAgentState  OPTIONAL,
        agentStateCondition             [1] IMPLICIT AgentStateCondition OPTIONAL}


AgentStateCondition ::= ENUMERATED
{       forcedPause                     (0),
        other                           (1)}
```

Parameter Name	*Type*	*Description*

agentStateList	List of Structures	This parameter specifies a list of agent identifiers, and/or their corresponding agent states and ACD groups for a given device. ~~This list parameter has a maximum list size of 32 entries.~~ This list parameter has 1 entry~~. Each entry~~ containing the following components:
		- agentID (C) AgentID - Indicates the agentID of the agent with respect to the associated ACD device or ACD group. This component shall be provided if there are multiple agentIDs associated with the agent device.
		- loggedOnState (M) Boolean - Indicates the logged on state of the agent. The complete set of possible values is:
		  - TRUE - Agent is logged on.
		  - FALSE - Agent is not logged on.
		- agentInfo (O) List of Structures - A specific agent may be associated with one or more agent states. The following components are associated with each agent state:
		~~- acdGroup (C) DeviceID - This component is mandatory in an entry when there is more than one entry in the list.~~
		- agentState (M) Enumerated- The complete set of possible values is Busy, Not Ready, Null, Ready, and Working After Call.

2.11.1.6 Get Auto Answer

Description

The Get Auto Answer service provides the auto-answer feature status at a specified device. The auto-answer feature is used to automatically connect to (answer) a call when it arrives at a device, without

manual intervention (AICC [1]headset mode). The service specifies the number of rings at the device before the device is auto-answered.

Operation

```
getAutoAnswer OPERATION ::=
{       ARGUMENT                        GetAutoAnswerArgument
        RESULT                          GetAutoAnswerResult
        ERRORS                          {universalFailure}
        CODE                            local: 305
}
```

Service request

```
GetAutoAnswerArgument ::= SEQUENCE
{       device                          DeviceID,
        extensions                      CSTACommonArguments OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the DeviceID of the device on which the auto-answer status is being queried.

Service response

```
GetAutoAnswerResult ::= SEQUENCE
{       autoAnswerOn                    BOOLEAN,
        numberOfRings                   INTEGER                 OPTIONAL,
        extensions                      CSTACommonArguments     OPTIONAL}
```

Parameter Name	*Type*	*Description*
autoAnswerOn	Boolean	Specifies the value of the requested feature. The complete set of possible values is: - FALSE - auto answer is not enabled at the device - TRUE - auto answer is enabled at the device

2.11.1.7 Get Auto Work Mode

Description

The Get Auto Work Mode service provides the auto-work mode feature status at a specified device. The auto-work feature is used to automatically transition an agent state to the WorkingAfterCall state (from the Busy state) after an agent is finished with a call. The feature may also automatically transition the agent state from WorkingAfterCall (to Ready, for example) after a certain amount of time.

Operation

```
getAutoWorkMode OPERATION ::=
{       ARGUMENT                        GetAutoWorkModeArgument
        RESULT                          GetAutoWorkModeResult
        ERRORS                          {universalFailure}
        CODE                            local: 306
}
```

Service request

```
GetAutoWorkModeArgument ::= SEQUENCE
{       device                          DeviceID,
        extensions                      CSTACommonArguments             OPTIONAL}
```

[1] **A**utomatic **I**ncoming UCD **C**all **C**onnection

Parameter Name	Type	Description
device	DeviceID	Specifies the DeviceID of the device on which the auto-work mode status is being queried. The DeviceID may refer to an ACD device, ACD group, or to an Agent.

Service response

```
GetAutoWorkModeResult ::= SEQUENCE
{       autoWorkOn                  BOOLEAN,
        autoWorkInterval            INTEGER                     OPTIONAL,
        extensions                  CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
autoWorkOn	Boolean	Specifies the value of the requested feature. The complete set of possible values is: - FALSE - Auto-work mode off. - TRUE - Auto-work mode on.
autoWorkInterval	Value	Indicates the number of seconds that the agent state remains in the AfterCallWork state. The value of zero indicates that the auto work mode feature does not automatically cause the transition of the agent state from WorkingAfterCall (i.e., the transition must be done manually or via the Set Agent State service)

~~2.11.1.8 Get Call Back~~

Not supported

2.11.1.9 Get Caller ID Status

Description

The Get Caller ID Status service provides the Caller ID status at the specified device. When the status is set to TRUE, the device's Caller ID will be provided on all calls originating from it. When the status is set to FALSE, the switching function will not provide the device's Caller ID on calls, which originate from it, to the called device.

Operation

```
getCallerIDStatus OPERATION ::=
{       ARGUMENT                    GetCallerIDStatusArgument
        RESULT                      GetCallerIDStatusResult
        ERRORS                      {universalFailure}
        CODE                        local: 307
}
```

Service request

```
GetCallerIDStatusArgument ::= SEQUENCE
{       device                      DeviceID,
        extensions                  CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the DeviceID of the device on which the Caller ID status feature is being queried.

Service response

```
GetCallerIDStatusResult ::= SEQUENCE
{       callerIDProvided                BOOLEAN,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
callerIDProvided	Boolean	Specifies the value of the requested feature. The complete set of possible values is: - FALSE - Caller ID is not provided on calls originating from this device. - TRUE - Caller ID is provided on calls originating from this device.

2.11.1.10 Get Do Not Disturb

Description

The Get Do Not Disturb service provides the do not disturb feature status at a specified device. The do not disturb feature is used to prevent incoming calls at a device.

Operation

```
getDoNotDisturb OPERATION ::=
{       ARGUMENT                        GetDoNotDisturbArgument
        RESULT                          GetDoNotDisturbResult
        ERRORS                          {universalFailure}
        CODE                            local: 308
}
```

Service request

```
GetDoNotDisturbArgument ::= SEQUENCE
{       device                          DeviceID,
        extensions                      CSTACommonArguments OPTIONAL}
```

Parameter Name	Type	Description
device	DeviceID	Specifies the DeviceID of the device on which the do not disturb feature is being queried.

Service response

```
GetDoNotDisturbResult ::= SEQUENCE
{       doNotDisturbOn                  BOOLEAN,
        callOrigination                 CallOrigination                 OPTIONAL,
        callingDeviceList               SEQUENCE OF DeviceID            OPTIONAL,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Parameter Name	Type	Description
doNotDisturbOn	Boolean	Specifies the value of the requested feature. The complete set of possible values is: - FALSE - Do not disturb feature is not enabled. - TRUE - Do not disturb feature is enabled.

2.11.1.11 Get Forwarding

Description

The Get Forwarding service provides the forwarding feature status at a specified device. The status returned may consist of one or more forwarding types that are active at the specified device based on user defined conditions.
The forwarding feature is used to redirect calls that arrive at a specified device to an alternate destination.

Operation

```
getForwarding OPERATION ::=
{       ARGUMENT                        GetForwardingArgument
        RESULT                          GetForwardingResult
```

```
        ERRORS                          {universalFailure}
        CODE                            local: 309
}
```

Service request

```
GetForwardingArgument ::= SEQUENCE
{       device                          DeviceID,
        extensions                      CSTACommonArguments     OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	DeviceID	Specifies the device on which to query.

Service response

```
GetForwardingResult ::= SEQUENCE
{       forwardingList                  ForwardList,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
forwardList	List of Structures	The list contains one structure per forwardingType/forwardDN combination. The structure has the following components: - forwardingType (C) Enumerated. Specifies the type of forwarding. It shall be provided for user specified settings and it is optional for switching function default settings. The „internal“ and „external“ types refer to the type of call origination (for example an external call) that will be forwarded if it matches a forwarding type (for example forwardImmExt) enabled at the device. The complete set of possible values is: - forwardImmediate - forwardImmInt - forwardImmExt forwardStatus (M) Boolean. Indicates the status of the forwarding type. The complete set of possible values is: - FALSE - the forwarding type is deactivated. - TRUE - the forwarding type is active. - forwardDN (C) DeviceID. Specifies the destination to which calls are forwarded. It shall be provided for user specified settings and it is optional for switching function default settings.

~~*2.11.1.12 Get Last Number Dialed*~~

Not supported

~~*2.11.1.13 Get Routeing Mode*~~

Not supported

2.11.1.14 Set Agent State

Description

The Set Agent State service requests a new agent state at a specified device. In the case where an ACD agent is involved with an ACD call, the transition to the requested state may or may not occur until the current connection transitions to the null state.

Operation

```
setAgentState OPERATION ::=
{       ARGUMENT                        SetAgentStateArgument
        RESULT                          SetAgentStateResult
        ERRORS                          {universalFailure}
        CODE                            local: 312
}
```

Service request

```
SetAgentStateArgument ::= SEQUENCE
{       device                      DeviceID,
        requestedAgentState         ReqAgentState,
        agentID                     [2] IMPLICIT AgentID                OPTIONAL,
        password                    [3] IMPLICIT AgentPassword          OPTIONAL,
        group                       DeviceID                            OPTIONAL,
        extensions                  CSTACommonArguments                 OPTIONAL}
```

```
ReqAgentState ::= ENUMERATED
{       loggedOn                    (0),
        loggedOff                   (1),
        notReady                    (2),
        ready                       (3),
        workingAfterCall            (4)}
```

Parameter Name	***Type***	***Description***
device	DeviceID	Specifies the DeviceID for the ACD agent for which the state is to be changed. The device may also be an ACD device or an ACD group device if allowed by the switching function, as indicated by the capability exchange services.
requestedAgentState	Enumerated	Specifies the requested agent state. The complete set of possible values is: - loggedOn - Requests that the agent be logged on. - loggedOff - Requests that the agent be logged off. - notReady - Requests that the agent be placed into the notReady agent state. - ready - Requests that the agent be placed into the ready state. - workingAfterCall - Requests that the agent be placed into the workingAfterCall state. Note that the list of values in this parameter is different from the list of values in the agentState parameter in the Get Agent State service.
agentID	AgentID	Specifies the agent identifier. Used for agent login

Service response

```
SetAgentStateResult ::= SEQUENCE
{       pendingAgentState           PendingAgentState                   OPTIONAL,
        extensions                  CSTACommonArguments                 OPTIONAL}
```

~~2.11.1.15 Set Auto Answer~~

Not supported

~~2.11.1.16 Set Auto Work Mode~~

Not supported

2.11.1.17 Set Caller ID Status

Description

This Set Caller ID Status service sets the Caller ID Status at the specified device. When the status is set to TRUE, the device's Caller ID will be provided on all calls originating from it. When the status is set to FALSE, the switching function will not provide the device's Caller ID on calls, which originate from it, to the called device.

Note

This service has effect for external CO calls only.

Operation

```
setCallerIDStatus OPERATION ::=
{       ARGUMENT                        SetCallerIDStatusArgument
        RESULT                          SetCallerIDStatusResult
        ERRORS                          {universalFailure}
        CODE                            local: 315
}
```

Service request

```
SetCallerIDStatusArgument ::= SEQUENCE
{       device                          DeviceID,
        callerIDProvided                BOOLEAN,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	Type	Description
device	DeviceID	Specifies the device on which to query.
callerIDProvided	Boolean	Specifies whether this device's caller ID value is being provided on outbound calls. The complete set of possible values is: - FALSE - Caller ID is not to be provided on call(s) originating from this device. - TRUE - Caller ID is to be provided on call(s) originating from this device.

Service response

```
SetCallerIDStatusResult  ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

2.11.1.18 Set Do Not Disturb

Description

The Set Do Not Disturb service allows the computing function to control the do not disturb feature at a specified device. The do not disturb feature is typically used to prevent a specified device from being alerted.

Note

This service is not possible for the intercept positions.

Operation

```
setDoNotDisturb OPERATION ::=
{       ARGUMENT                        SetDoNotDisturbArgument
        RESULT                          SetDoNotDisturbResult
        ERRORS                          {universalFailure}
        CODE                            local: 316
}
```

Service request

```
SetDoNotDisturbArgument ::= SEQUENCE
{       device                          DeviceID,
        doNotDisturbOn                  BOOLEAN,
        callOrigination                 CallOrigination                         OPTIONAL,
        callingDeviceList               SEQUENCE OF DeviceID                    OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	Type	Description
device	DeviceID	Specifies the device on which to set the feature.
doNotDisturbOn	Boolean	Specifies whether the do not disturb feature is enabled. The complete set of possible values is: - FALSE- Do not disturb feature is not enabled. - TRUE- Do not disturb feature is enabled.

Service response

```
SetDoNotDisturbResult  ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

2.11.1.19 Set Forwarding

Description

The Set Forwarding service allows the computing function to control the forwarding feature at a specified device based on user defined conditions. The forwarding feature is used to redirect calls that arrive at a specified device to an alternate destination.
This service allows only one user-specified setting (forwarding type/forward-destination combination) to be changed per service invocation.

Operation

```
setForwarding OPERATION ::=
{       ARGUMENT                        SetForwardingArgument
        RESULT                          SetForwardingResult
        ERRORS                          {universalFailure}
        CODE                            local: 317
}
```

Service request

```
SetForwardingArgument ::= SEQUENCE
{       device                          DeviceID,
        forwardingType                  ForwardingType                  OPTIONAL,
        activateForward                 BOOLEAN,
        forwardDN                       DeviceID                        OPTIONAL,
        ringCount                       INTEGER (1..100)                OPTIONAL,
        ringDuration                    [3] IMPLICIT INTEGER            OPTIONAL,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Parameter Name	Type	Description
device	DeviceID	Specifies the device on which to set the feature.
forwardingType	Enumerated	Specifies the type of forwarding. The „internal“ and „external“ types refer to the type of call origination (for example an external call) that will be forwarded if it matches a forwarding type (for example, forwardImmExt) enabled at the device. The complete set of possible values is: - forwardImmediate - forwardNoAns - forwardImmInt - forwardImmExt This parameter shall be provided for user specified forwarding settings and shall not be provided for switching function default settings.
activateForward	Boolean	Indicates the status of the forwarding type. The complete set of possible values is: - FALSE - Deactivate forwarding - TRUE - Activate forwarding.
forwardDN	DeviceID	Specifies the device to which new calls are forwarded. This parameter shall be provided for user specified forwarding settings when activateForward is TRUE (it is optional when FALSE). It shall not be provided for switching function default settings.

Service response

```
SetForwardingResult  ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

Note:

- The Set Forwarding service for MULAP is based on the forwarding of a single extension. The first configured non virtual MULAP Master is addressed by the Feature Processing as a representative for the MULAP. That enables the system to visible the activation of Call Forwarding on the display.
- with standard feature code *1 Call Forwarding is set for the MULAP only from the master of the MULAP and not from all members. Using standard feature code *501 Call Forwarding can be activated from every member that belongs in the MULAP group but this is not displayed.

~~2.11.1.20 Set Routeing Mode~~

Not supported

2.11.2 Events

Note

- The maximum amount of devices that can be monitored at the same time is stated in 2.18.1 OpenScape Business Parameter Constraints.
- There is no event report on non-voice calls.
- See also 2.18.3 List of monitorable OpenScape Business Devices.

2.11.2.1 Agent Busy

Description

The Agent Busy event indicates that an agent has entered the Busy state. In this state an agent is involved with an existing ACD call at a device, even if that call is on hold at the device. It also implies that the agent may be able to accept non-ACD calls. Calls between agents, calls between supervisors and agents and private calls may or may not cause this transition.

Operation

```
AgentBusyEvent ::= SEQUENCE
{       agentDevice                     SubjectDeviceID,
        agentID                         AgentID                         OPTIONAL,
        acdGroup                        DeviceID                        OPTIONAL,
        pendingAgentState               [2] IMPLICIT PendingAgentState  OPTIONAL,
        cause                           [3] IMPLICIT EventCause         OPTIONAL,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
agentDevice	SubjectDeviceID	Indicates the device at which the agent entered the Agent Busy state.
agentID	AgentID	Indicates the agent identifier. This parameter shall be provided if there are multiple agentIDs associated with the agent device.
acdGroup	DeviceID	Indicates the acd group.
security	CSTASecurityData	Specifies timestamp information

2.11.2.2 Agent Logged Off

Description

The Agent Logged Off event indicates that an agent has logged off an ACD device or an ACD group. Typical examples of when this event may be generated are:

- An agent logs off using the telephone.
- An agent is logged off via the Set Agent State service.
- A supervisor logs off an agent on behalf of the agent.
- A logged on device leaves a switching sub-domain or becomes an invalid device in a switching sub-domain (is deconfigured, for example).

Operation

```
AgentLoggedOffEvent ::= SEQUENCE
{       agentDevice                 SubjectDeviceID,
        agentID                     [2] IMPLICIT AgentID                OPTIONAL,
        acdGroup                    DeviceID                            OPTIONAL,
        agentPassword               [3] IMPLICIT AgentPassword          OPTIONAL,
        cause                       EventCause                          OPTIONAL,
        extensions                  CSTACommonArguments                 OPTIONAL}
```

Parameter Name	*Type*	*Description*
agentDevice	SubjectDeviceID	Indicates the device where the agent logged off.
agentID	AgentID	Indicates the agent identifier. This parameter shall be provided if there are multiple agentIDs associated with the agent device.
acdGroup	DeviceID	Indicates the acd group.
security	CSTASecurityData	Specifies timestamp information

2.11.2.3 Agent Logged On

Description

The Agent Logged On event indicates that an agent is logged-on at a particular device to an ACD device or ACD group and is ready to contribute to the activities of the ACD device or ACD group. It does not indicate that the agent is ready to accept ACD calls. Typical examples of when this event may be generated are:

- The agent logged on using the telephone.
- The agent logged on using the Set Agent State service.
- During system start-up, if the agent is configured for logging on automatically.

Operation

```
AgentLoggedOnEvent ::= SEQUENCE
{       agentDevice                 SubjectDeviceID,
        agentID                     [2] IMPLICIT AgentID                OPTIONAL,
        acdGroup                    DeviceID                            OPTIONAL,
        agentPassword               [3] IMPLICIT AgentPassword          OPTIONAL,
        cause                       EventCause                          OPTIONAL,
        extensions                  CSTACommonArguments                 OPTIONAL}
```

Parameter Name	*Type*	*Description*
agentDevice	SubjectDeviceID	Indicates the device where the agent logged on.
agentID	AgentID	Indicates the agent identifier. This parameter shall be provided if there are multiple agentIDs associated with the agent device.
acdGroup	DeviceID	Indicates the acd group.
security	CSTASecurityData	Specifies timestamp information

2.11.2.4 Agent Not Ready

Description

The Agent Not Ready event indicates that an agent has entered the Agent Not Ready state. In this state an agent is logged-on at a particular device to an ACD device or ACD group but is not prepared to handle calls that the ACD distributes. While in this state an agent may receive calls that are not ACD calls. Typical examples of when this event may be generated are:

- An agent logs on using the telephone and is placed into the Not Ready agent state.
- An agent invokes the Agent Not Ready feature on the telephone.
- The agent invoked Agent Not Ready by using the Set Agent State service.
- A supervisor invokes the Agent Not Ready feature on behalf of the agent.

Operation

```
AgentNotReadyEvent ::= SEQUENCE
{       agentDevice                 SubjectDeviceID,
        agentID                     AgentID                     OPTIONAL,
        acdGroup                    DeviceID                    OPTIONAL,
        cause                       EventCause                  OPTIONAL,
        extensions                  CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
agentDevice	SubjectDeviceID	Indicates the device at which the agent entered the Agent Not Ready state.
agentID	AgentID	Indicates the agent identifier. This parameter shall be provided if there are multiple agentIDs associated with the agent device.
acdGroup	DeviceID	Indicates the acd group.
security	CSTASecurityData	Specifies timestamp information

2.11.2.5 Agent Ready

Description

The Agent Ready event indicates that an agent has entered the Ready state. In this state, an agent is logged-on at a particular device to an ACD device or ACD group and is prepared to handle ACD calls even though it may be involved with non-ACD calls. Typical examples of when this event may be generated are:

- An agent auto-work timer expires. (This is a configuration option.)
- An agent invokes the Agent Ready feature on the telephone.
- The agent invoked Agent Ready by using the Set Agent State service.
- A supervisor invokes the Agent Ready feature on behalf of the agent.

Operation

```
AgentReadyEvent ::= SEQUENCE
{       agentDevice                 SubjectDeviceID,
        agentID                     AgentID                     OPTIONAL,
        acdGroup                    DeviceID                    OPTIONAL,
        cause                       EventCause                  OPTIONAL,
        extensions                  CSTACommonArguments         OPTIONAL}
```

Parameter Name	Type	Description
agentDevice	SubjectDeviceID	Indicates the device at which the agent entered the Agent Ready state.
agentID	AgentID	Indicates the agent identifier. This parameter shall be provided if there are multiple agentIDs associated with the agent device.
acdGroup	DeviceID	Indicates the acd group.
security	CSTASecurityData	Specifies timestamp information

2.11.2.6 Agent Working After Call

Description

The Agent Working After Call event indicates that an agent has entered the Working After Call state. In this state an agent is no longer connected to an ACD call but is still occupied with work related to a previous ACD call. In this state, an agent cannot receive ACD calls but may be able to receive non-ACD calls. The agent may be performing administrative duties (e.g., updating a business order form) for a previous call, or may be involved with a non-ACD call. Typical examples of when this event may be generated are:

- An agent completes an ACD call and goes into the workingAfterCall state. (This is a configuration option.)
- An agent invokes the Working After Call feature on the telephone.
- An agent invoked workingAfterCall state by using the Set Agent State service.

Operation

```
AgentWorkingAfterCallEvent ::= SEQUENCE
{       agentDevice             SubjectDeviceID,
        agentID                 AgentID                         OPTIONAL,
        acdGroup                DeviceID                        OPTIONAL,
        pendingAgentState       [2] IMPLICIT ENUMERATED
                                {notReady       (0),
                                ready           (1),
                                null            (2)}            OPTIONAL,
        cause                   [3] IMPLICIT EventCause         OPTIONAL,
        extensions              CSTACommonArguments             OPTIONAL}
```

Parameter Name	Type	Description
agentDevice	SubjectDeviceID	Indicates the device at which the agent entered the workingAfterCall state.
agentID	AgentID	Indicates the agent identifier. This parameter shall be provided if there are multiple agentIDs associated with the agent device.
acdGroup	DeviceID	Indicates the acd group.
security	CSTASecurityData	Specifies timestamp information

~~2.11.2.7 Auto Answer~~

Not supported

~~2.11.2.8 Auto Work Mode~~

Not supported

~~2.11.2.9 Call Back~~

Not supported

~~2.11.2.10 Call Back Message~~

Not supported

2.11.2.11 Caller ID Status

Description

The CallerID Status event indicates that the caller ID status has been changed for a device.
Typical examples of when this event may be generated are:

- The Caller ID status change has been invoked on the telephone.
- The computing function, on behalf of a user, has invoked the Set Caller ID Status service.

Operation

```
CallerIDStatusEvent ::= SEQUENCE
{       device                  SubjectDeviceID,
        callerIDProvided        BOOLEAN,
        extensions              CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	SubjectDeviceID	Indicates the device where the do not disturb feature was changed.
callerIDProvided	Boolean	Indicates the status of the feature. The complete set of possible values is: - FALSE - caller ID status is not enabled. - TRUE - caller ID status is enabled.
security	CSTASecurityData	Specifies timestamp information

2.11.2.12 Do Not Disturb

Description

The Do Not Disturb event indicates that the do not disturb feature has been changed for a device. Typical examples of when this event may be generated are:

- The do not disturb feature has been changed on the telephone.
- The computing function, on behalf of a user, has invoked the Set Do Not Disturb service.

Operation

```
DoNotDisturbEvent ::= SEQUENCE
{       device                  SubjectDeviceID,
        doNotDisturbOn          BOOLEAN,
        callOrigination         CallOrigination                 OPTIONAL,
        callingDeviceList       SEQUENCE OF DeviceID            OPTIONAL,
        extensions              CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	SubjectDeviceID	Indicates the device where the do not disturb feature was changed.
doNotDisturbOn	Boolean	Indicates the status of the feature. The complete set of possible values is: - FALSE - Do not disturb feature is not enabled. - TRUE - Do not disturb feature is enabled.
security	CSTASecurityData	Specifies timestamp information

2.11.2.13 Forwarding

Description

The Forwarding event indicates that the forwarding feature has been changed for a device. Typical examples of when this event may be generated are:

- The forwarding feature has been changed on the telephone.
- The computing function, on behalf of a user, has invoked the Set Forwarding service.

Operation

```
ForwardingEvent ::= SEQUENCE
{       device                  SubjectDeviceID,
        forwardingType          ForwardingType                  OPTIONAL,
        forwardStatus           BOOLEAN,
```

```
    forwardTo                   DeviceID                    OPTIONAL,
    forwardDefault              ForwardDefault              OPTIONAL,
    ringCount                   INTEGER (1..100)            OPTIONAL,
    ringDuration                [4] IMPLICIT INTEGER        OPTIONAL,
    extensions                  CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	SubjectDeviceID	Indicates the device where the forwarding feature was changed.
forwardingType	Enumerated	Indicates the type of forwarding. It shall be provided for user specified settings and it is optional for switching function default settings.The internal and external types refer to the type of call origination (for example an external call) that will be forwarded if it matches a forwarding type (for example forwardImmExt) enabled at the device. The complete set of possible values is: - forwardImmediate - forwardImmInt - forwardImmExt
forwardStatus	Boolean	Indicates the status of the forwarding type. The complete set of possible values is: - FALSE - the forwarding type is deactivated - TRUE - the forwarding type is active
forwardTo	DeviceID	Specifies the destination to which calls are forwarded. It shall be provided for user specified settings and it is optional for switching function default settings
security	CSTASecurityData	Specifies timestamp information

~~*2.11.2.14 Routeing Mode*~~

Not supported

2.12 Device Maintenance Events

Note

The Device Maintenance Events are supported for S_0-Devices and SIP-Endpoints.

2.12.1 Events

2.12.1.1 Back In Service

Description

The Back In Service event indicates that the device has been returned to service and is operating normally.

Operation

```
BackInServiceEvent ::= SEQUENCE
{   device                      SubjectDeviceID,
    cause                       EventCause                  OPTIONAL,
    extensions                  CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	SubjectDeviceID	Indicates the device that is back in service.
security	CSTASecurityData	Specifies timestamp information

~~*2.12.1.2 Device Capabilities Changed*~~

Not supported

2.12.1.3 Out Of Service

Description

The Out Of Service event indicates that the device has entered a maintenance state (i.e., has been taken out of service) and can no longer accept calls and some categories of CSTA service requests (Call Control services, for example).

Operation

```
OutOfServiceEvent ::= SEQUENCE
{     device                    SubjectDeviceID,
      cause                     EventCause                    OPTIONAL,
      extensions                CSTACommonArguments           OPTIONAL}
```

Parameter Name	*Type*	*Description*
device	SubjectDeviceID	Indicates the device that has been taken out of service.
security	CSTASecurityData	Specifies timestamp information.

~~2.12.1.4 Partially In Service~~

Not Supported.

~~2.13 Data Collection Services~~

The Data Collection Services and events are not supported.

~~2.13.1 Services~~

~~2.13.1.1 Data Collected~~

Not supported

~~2.13.1.2 Data Collection Resumed~~

Not supported

~~2.13.1.3 Data Collection Suspended~~

Not supported

~~2.13.1.4 Resume Data Collection~~

Not supported

~~2.13.1.5 Start Data Collection~~

Not supported

~~2.13.1.6 Stop Data Collection~~

Not supported

~~2.13.1.7 Suspend Data Collection~~

Not supported

2.14 Voice Unit Services & Events

The voice unit services and events are used to attach recorded anouncements to a connection.

2.14.1 Services

2.14.1.1 Play Message

Description

The Play Message service plays a voice message on a particular connection.

Note

- The ConnectionID has be constructed as both (including CallID and DeviceID).
- The MessageID has to be filled with the index of the rcda-device. Valid indices are 0x01 .. 0x10 coded as 1 byte integer or “1” .. “16” coded as ASCII (MessageID length has to be >1 for ASCII).
- For start of internal Music On Hold an empty MessageID (0x00 coded as 1 byte integer or “0” coded as ASCII) has to be used.
- For playMessage the device has to be in a valid connection (ringback state).

Operation

```
PlayMessage OPERATION ::=
{       ARGUMENT                        PlayMessageArgument
        RESULT                          PlayMessageResult
        ERRORS                          {universalFailure}
        CODE                            local: 502
}
```

Service request

```
PlayMessageArgument ::= SEQUENCE
{       messageToBePlayed               MessageID,
        overConnection                  ConnectionID,
        duration                        INTEGER                         OPTIONAL,
        termination                     TerminatingConditions           OPTIONAL,
        extensions                      CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
messageToBePlayed	MessageID	Specifies the message to be played.
overConnection	ConnectionID	Specifies the connection on which the message is to be played.

Service response

```
PlayMessageResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

2.14.1.2 Stop

Description

The Stop service stops playing of a message and resets the position pointer to the beginning of the message.

Note

The ConnectionID may be constructed as deviceID or both (including CallID and DeviceID).

Operation

```
Stop OPERATION ::=
{       ARGUMENT                StopArgument
        RESULT                  StopResult
        ERRORS                  {universalFailure}
        CODE                    local: 508
}
```

Service request

```
StopArgument ::= SEQUENCE
{       connection              ConnectionID,
        messageToBeStopped      MessageID                   OPTIONAL,
        extensions              CSTACommonArguments         OPTIONAL}
```

Parameter Name	*Type*	*Description*
connection	ConnectionID	Specifies the connection on which the message is to be stopped
messageToBeStopped	MessageID	Specifies the message to be stopped

Service response

```
StopResult ::= CHOICE
{       extensions              CSTACommonArguments,
        noData                  NULL}
```

~~*2.14.1.3 Concatenate Message*~~

- Not supported

~~*2.14.1.4 Delete Message*~~

- Not supported

~~*2.14.1.5 Query Voice Attribute*~~

- Not supported

~~*2.14.1.6 Record Message*~~

- Not supported

~~*2.14.1.7 Reposition*~~

- Not supported

~~*2.14.1.8 Resume*~~

- Not supported

~~*2.14.1.9 Review*~~

- Not supported

~~*2.14.1.10 Set Voice Attribute*~~

- Not supported

~~*2.14.1.11 Suspend*~~

- Not supported

~~*2.14.1.12 Synthesize Message*~~

- Not supported

2.14.2 Events

2.14.2.1 Play

Description

The Play event indicates that a message is being played.

Operation

```
PlayEvent ::=SEQUENCE
{       connection              ConnectionID,
        message                 MessageID,
        length                  [0] IMPLICIT INTEGER            OPTIONAL,
        currentPosition         [1] IMPLICIT INTEGER            OPTIONAL,
        speed                   [2] IMPLICIT INTEGER            OPTIONAL,
        cause                   EventCause                      OPTIONAL,
        servicesPermitted       ServicesPermitted               OPTIONAL,
        extensions              CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
connection	ConnectionID	Indicates the connection on which the message is being played.
message	MessageID	Indicates the message that is being played.
security	CSTASecurityData	Specifies timestamp information

~~2.14.2.2 Record~~

- Not supported

~~2.14.2.3 Review~~

- Not supported

2.14.2.4 Stop

Description

The Stop event indicates that a play operation for a message on a connection has been stopped.

Operation

```
StopEvent ::= SEQUENCE
{       connection              ConnectionID,
        message                 MessageID,
        length                  [0] IMPLICIT INTEGER            OPTIONAL,
        currentPosition         [1] IMPLICIT INTEGER            OPTIONAL,
        cause                   EventCause                      OPTIONAL,
        servicesPermitted       ServicesPermitted               OPTIONAL,
        extensions              CSTACommonArguments             OPTIONAL}
```

Parameter Name	*Type*	*Description*
connection	ConnectionID	Indicates the connection of the stopped message.
message	MessageID	Indicates the message that is being stopped.
security	CSTASecurityData	Specifies timestamp information

~~2.14.2.5 Suspend Play~~

- Not supported

~~2.14.2.6 Suspend Record~~

- Not supported

~~2.14.2.7 Voice Attribute Changed~~

- Not supported

2.15 Call Detail Record Services

Note
The Call Detail Record Services have no impact on S_0-Devices and SIP-Endpoints.

2.15.1 Services

~~*2.15.1.1 Call Detail Records Notification*~~
Not supported

2.15.1.2 Call Detail Records Report

Description

The Call Detail Records Report service provides call detail information for one or more calls.This information may be used to derive charging information, statistic purposes, etc.

Operation

```
cDRReport      OPERATION ::=
{      ARGUMENT                    CDRReportArgument
       RESULT                      CDRReportResult
       ERRORS                      {universalFailure}
       CODE                        local: 361
}
```

Service request

```
CDRReportArgument ::= SEQUENCE
{      cdrRrossRefID               CDRCrossRefID,
       numberOfRecordsSent         INTEGER (1..128),                -- always 1
       cdrInfo                     CDRInfo,
       lastStoredCDRReportSent     BOOLEAN                          OPTIONAL,
       extensions                  CSTACommonArguments              OPTIONAL}
```

Parameter Name	*Type*	*Description*
cdrCrossRefID	CDRCrossRefID	Specifies the CDR cross reference identifier.
numberOfRecordsSent	Value (1..128)	Specifies the number of call detail records that are provided in this service.
cdrInfo	List of Structures	Specifies a list CDR information. Each entry in the list consists of the following components: **recordNumber** (O) Value (1..128) - Indicates the record number, if more that one record is sent. **recordCreationTime** (M) TimeInfo - Indicates the time that the CDR record was created. **callingDevice** (O) CallingDeviceID - Indicates the calling device. **calledDevice** (O) CalledDeviceID - Indicates the called device. **associatedCallingDevice** (O) AssociatedCallingDeviceID Indicates the Network Interface Device (trunk, for example) associated with the calling device. **associatedCalledDevice** (O) AssociatedCalledDeviceID - Indicates the Network Interface Device (trunk, for example) associated with the called device. **networkCallingDevice** (O) NetworkCallingDeviceID - Indicates the calling device that was provided by the network. **networkCalledDevice** (O) NetworkCalledDeviceID - Indicates the called device that was provided by the network. **mediaCallCharacteristics** (O) MediaCallCharacteristics - Specifies the media class (voice, digital data, etc.), connection rate, etc. of the call. **tariffTable** (O) Value - Indicates the table to be used according to the network type characteristics (PSTN, ISDN, etc.) **connectionDuration** (O) Value - Indicates the duration of the connection, in tenths of seconds. **accessCode** (O) Value - Indicates the dialled access number (e.g. to differentiate between business or private external calls) **carrier** (O) Value - Indicates the network that was used. **selectedRoute** (O) Value - Indicates the route that was used. **chargingInfo** (O) ChargingInfo - Indicates the value of charging or currency units charged to a device in a call. **supplServiceInfo** (O) Bitmap - Indicates the features (supplementary services) used. Multiple bits may be set. The possible values are: - normalCall - transferCall - callForwarding - conferencing - other **accountInfo** (O) AccountInfo - Indicates the account code used for the call

Service response

```
CDRReportResult ::= CHOICE
{       extensions                    CSTACommonArguments,
        noData                        NULL}
```

~~2.15.1.3 Send Stored Call Detail Records~~

Not supported

2.15.1.4 Start Call Detail Records Transmission

Description

The Start Call Detail Records Transmission service starts the transmission of Call Detail Records. The switching function starts transmitting call detail information at the end of each call if transmission of Call Detail Records was activated. If transmission of Call Detail Records was not activated the Call Detail Records are buffered in the PBX and transmitted when transmission gets activated.

Note:

- that the buffer size is limited and therefore Call Detail Records will be lost if the buffer is full.
- that only 1 CDR session is provided.

Operation

```
startCDRTransmission    OPERATION ::=
{       ARGUMENT                        StartCDRTransmissionArgument
        RESULT                          StartCDRTransmissionResult
        ERRORS                          {universalFailure}
        CODE                            local: 363
}
```

Service request

```
StartCDRTransmissionArgument ::= SEQUENCE
{       transferMode                    CDRTransferMode,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
transferMode	Enumerated	M Specifies the mode of transmission. The complete set of possible modes is: - transferAtEndOfCall - specifies that the requested information shall be sent after the end of each call or call segment.

Service response

```
StartCDRTransmissionResult ::= SEQUENCE
{       cdrRrossRefID                   CDRCrossRefID,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
cdrCrossRefID	CDRCrossRefID	Specifies the CDR cross reference identifier.

2.15.1.5 Stop Call Detail Records Transmission

Description

The Stop Call Detail Records Transmission service is used to cancel a previously initiated Start Call Detail Records Transmission service. A positive acknowledgement to the service request indicates that the CDR cross reference identifier used in the Start Call Detail Records Transmission service has become invalid. Either the computing function or the switching function may issue the service.

Operation

```
stopCDRTransmission     OPERATION ::=
{       ARGUMENT                        StopCDRTransmissionArgument
        RESULT                          StopCDRTransmissionResult
        ERRORS                          {universalFailure}
        CODE                            local: 364
}
```

Service request

```
StopCDRTransmissionArgument ::= SEQUENCE
{       cdrRrossRefID                   CDRCrossRefID,
        cdrTermReason                   CDRTermReason                           OPTIONAL,
        extensions                      CSTACommonArguments                     OPTIONAL}
```

Parameter Name	*Type*	*Description*
cdrCrossRefID	CDRCrossRefID	Specifies the CDR cross reference identifier.

Service response

```
StopCDRTransmissionResult ::= CHOICE
{       extensions                      CSTACommonArguments,
        noData                          NULL}
```

2.16 Vendor Specific Extensions Services & Events

Not supported

2.17 Samples

2.17.1 Makecall

2.17.1.1 Invoke

S: A1 16 02 01 02 02 01 0A 30 0E 30 05 80 03 31 30 30 30 05 80 03 31 30 31

```
rOSE.roiv-apdu
{ -- SEQUENCE --
   invokeID = 2,
   operation-value = 10 (makeCall),
   argument
   { -- SEQUENCE --
      callingDevice
      { -- SEQUENCE --
         deviceIdentifier.dialingNumber = "100" '31 30 30'H
      },
      calledDirectoryNumber
      { -- SEQUENCE --
         deviceIdentifier.dialingNumber = "101" '31 30 31'H
      }
   }
}
```

2.17.1.2 Result

R: A2 1B 02 01 02 30 16 02 01 0A 30 11 6B 0F 30 0D 80 02 00 16 A1 07 30 05 80 03 31 30 30

```
rOSE.rors-apdu
{ -- SEQUENCE --
   invokeID = 2,
   { -- SEQUENCE --
      operation-value = 10 (makeCall),
      result
      { -- SEQUENCE --
         callingDevice.both
         { -- SEQUENCE --
            callID = '00 00 00 16'H,
            deviceID.staticID
            { -- SEQUENCE --
               deviceIdentifier.dialingNumber = "100" '31 30 30'H
            }
         }
      }
   }
}
```

2.17.2 Keyfeatures

Note

If nothing is indicated in the column ‘comment’ then the key number is sent to the application.

buttonFunction	Feature	comment
000	**Free**	
001	**Internal station**	
002	**S – service**	open servicemenu in all state and stop session
003	**Repeat last number**	
004	**Open listening/hands free talking**	in case of idle state the device goes to dial and the session goes susspend, in case of talk state open listening and session stays active

005	**DTMF sending**	
006	**Door opener (relay)**	
007	**Toggle**	in case of idle state comes a negative tone and the session stays active, in case of talk state change the partner and the session goes to susspend
008	Call forward, no destination	
009	**Conference**	in case of idle state comes a negative tone and the session stays active, in case of cons state make a conference and the session goes to susspend
010	**Speaker mute**	in case of idle state comes a negative tone and the session stays active, , in case of talk state microphone mute on/off and the session stays active
011	**Code lock**	
012	**Release**	in case of idle state stop the session, in case of talk state release the call and stop the session
013	**Night service**	
014	**Override**	in case of idle state comes a negative tone and the session stays active, in case of ringback state make an override and the session goes to susspend
015	**Waiting calls**	
016	**Call key**	in case of idle state incoming call will be picked up, and session will be suspended, otherwise comes a negative tone and the session stays active, in case of talk state comes a negative tone and the session stays active
017	**Voice calling**	
018	**Account code**	
019	**Speed dial individual**	
020	**Speed dial central/individual**	
021	**Int. calls diverted, int. dest.**	
022	**Int. calls diverted, ext. dest.**	
023	**Ext. calls diverted, int. dest.**	
024	**Ext. calls diverted, ext. dest.**	
025	**All calls diverted, int. dest.**	
026	**All calls diverted, ext. dest.**	
027	**Do Not Disturb**	
028	**Direct Station Select**	
029	**Message leave**	
030	**Send textmessage**	
031	**Send textmessage, Text 0**	
032	**Send textmessage, Text 1**	
033	**Send textmessage, Text 2**	
034	**Send textmessage, Text 3**	
035	**Send textmessage, Text 4**	
036	**Send textmessage, Text 5**	
037	**Send textmessage, Text 6**	
038	**Send textmessage, Text 7**	
039	**Send textmessage, Text 8**	
040	**Send textmessage, Text 9**	
041	**Callback**	
042	**not used**	
043	**Seizure**	
044	**Actors**	
045	**Trunk**	
046	**Call pickup**	
047	**Park**	
048	**Flash to analogue trunk line**	
049	**Retrieve call charge data**	
050	**not used**	
051	**not used**	
052	**Call pickup group**	in case of idle state incoming call will be picked up, and session will be suspended, otherwise comes a negative tone and the session stays active, in case of talk state incoming call will be picked up, and session will be suspended, otherwise comes a negative tone and the session stays active
053	**CFW for ext line**	
054	**Answer camp on**	in case of idle state comes a negative tone and the session stays active, in case of talk state incoming call will be picked up, and session will be suspended, otherwise comes a negative tone and

		the session stays active
055	**Direct answer**	
056	**Babyphone**	
057	**Incomming call assignment**	
058	**Call forward unconditional**	
059	**Call forward on no reply**	
060	**Call forward busy**	
061	**not used**	
062	**Entry to speed dial central**	
063	**Incomming call log**	
064	**Telephonebook**	
065	**Start/Stop linehunting**	
066	**Catch call**	
067	**Associated service**	
068	**Associated dialing**	
069	**Silence**	
070	**Silent call waiting**	
071	**Calling line identification restriction**	
072	**Fax received**	
073	**Dooropen via DTMF**	
074	**Common Hold**	
075	**Retrieve hold call**	
076	**MUSAP-key**	
077	**Seizure key**	
078	**Release of trunk**	
079	**Codelock central**	
080	**Message waiting receive**	
081	**Telefone Data Service**	
082	**Alarm clock**	
083	**Charge call**	
084	**Paging**	
085	**Not used**	
086	**Send textmessage, ext.,**	
087	**Send textmessage, ext., Text 0**	
088	**Send textmessage, ext., Text 1**	
089	**Send textmessage, ext., Text 2**	
090	**Send textmessage, ext., Text 3**	
091	**Send textmessage, ext., Text 4**	
092	**Send textmessage, ext., Text 5**	
093	**Send textmessage, ext., Text 6**	
094	**Send textmessage, ext., Text 7**	
095	**Send textmessage, ext., Text 8**	
096	**Send textmessage, ext., Text 9**	
097	**Call transfer**	in case of idle state comes a negative tone and the session stays active, in call initiated state stop session
098	**UCD Log On**	
099	**UCD available**	
100	**UCD wrap-up**	
101	**UCD show-calls**	
102	**UCD Night service internal**	
103	**UCD Night service external**	
104	**Internal callback**	
105	**Telefone Data Service**	
106	**Common call key**	in case of idle state incoming call will be picked up, and session will be suspended, otherwise comes a negative tone and the session stays active, in case of talk state comes a negative tone and the session stays active
107	**Call Waiting Terminating**	
108	**Clearkey**	
109	**Boss Ringing**	
110	**Boss 2 Ringing**	
111	**Call transfer of undialed trunks**	
112	**Not used**	
113	**MSN-related CFW, busy**	
114	**MSN- related CFW, no answer**	
115	**MSN- related CFW, all**	
116	**Deactivate Trace**	

117	**Shift key**	
118	**Mulap**	
119	**Mulap Ringassignment**	
120	**Mulap call forward, no destination**	
121	**Mulap int. calls diverted, int. dest.**	
122	**Mulap int. calls diverted, ext. dest.**	
123	**Mulap ext. calls diverted, int. dest.**	
124	Mulap ext. calls diverted, ext. dest.	
125	**Mulap all calls diverted, int. dest.**	
126	**Mulap all calls diverted, ext. dest.**	
127	**Mobile Pin**	
128	**DISA internal**	
129	**Parallel ringing on external device**	
130	**DISA internal, complete configuration of key**	
131	**Not used**	
132	**Relocate phone**	
133	**Headset accept call**	in case of idle state headset turn on/off session goes to suspend, in case of talk state headset turn on/off session goes to suspend
134	**internal phonebook**	
135	**LDAP phonebook**	
136	**internal station-key w/o LED**	
137	**Individual call forwarding to internal number**	
138	**Individual call forwarding to external number**	
139	**procedur key**	
140	**Not used**	
141	**mobil logon/logoff**	
142	**Consultation key**	
143	**Call recording**	
144	**MULAP Privacy Release**	
145	**Diskretes ansprechen**	
146	**Data I/O services**	
147	**LOCAL APPLICATION FEATURE KEY**	

2.17.3 Sample: getSwitchingFunctionCapabilities

```
> rOSE.rors-apdu
> { -- SEQUENCE --
>    invokeID = 1,
>    { -- SEQUENCE --
>       operation-value = 203 (getSwitchingFunctionCapabilities),
>       result
>       { -- SEQUENCE --
>          switchingSubDomainName = "customer name" '63 75 73 74 6F 6D 65 72 20 6E 61 6D 65'H,
>          manufacturerName = "Unify " '53 69 65 6D 65 6E 73 20 45 6E 74 65 72 70 72 69 73 65 20 43 6F 6D 6D 75 6E 69 63 61 74 69
6F 6E 73 20 47 6D 62 48 20 26 20 43 6F 2E 20 4B 47'H,
>          profiles = '10'B -- NrBits = 2 --
>          { -- true bits --
>             basicTelephonyProfile
>          },
>          deviceIDFormat = '11000000000001000000000000000000'B -- NrBits = 31 --
>          { -- true bits --
>             dialableDigitsAsterix,
>             dialableDigitsHash,
>             sFReprSlash
```

```
>         },
>         swDomainFeatures = '100100110011110'B -- NrBits = 15 --
>         { -- true bits --
>           isForwardingBefore,
>           userSpecific,
>           negativeAcknowledgment,
>           supportFailedWithAssConn,
>           recall,
>           callBack,
>           extCallsIncoming,
>           extCallsOutgoing
>         },
>         swAppearanceAddressability = '10'B -- NrBits = 2 --
>         { -- true bits --
>           nonAddressable
>         },
>         swAppearanceTypes = '001000'B -- NrBits = 6 --
>         { -- true bits --
>           basicBridged
>         },
>         ignoreUnsupportedParameters = 0 (ignoreParameters),
>         mediaClassSupport = '11000'B -- NrBits = 5 --
>         { -- true bits --
>           voice,
>           data
>         },
>         currentTime = "20130110542315" '32 30 30 38 30 31 31 30 35 34 32 33 31 35'H,
>         timeStampMode = '100'B -- NrBits = 3 --
>         { -- true bits --
>           allEvents
>         },
>         privateDataFormat = '0101'B -- NrBits = 4 --
>         { -- true bits --
>           otherTypeFromSF,
>           otherTypeToSF
>         },
>         transAndConfSetup = '11100'B -- NrBits = 5 --
>         { -- true bits --
>           consultationCall,
>           holdCallMakeCall,
>           alternateCall
>         },
>         monitorFilterItems
>         { -- SEQUENCE --
>           deviceOnDeviceMonitorFilter
>           { -- SEQUENCE --
>             callControl = '011111111111110000'B -- NrBits = 18 --
>             { -- true bits --
>               conferenced,
>               connectionCleared,
>               delivered,
>               diverted,
>               established,
>               failed,
>               held,
>               networkReached,
>               originated,
>               queued,
>               retrieved,
>               serviceInitiated,
>               transferred
>             },
>             callAssociated = '01000'B -- NrBits = 5 --
>             { -- true bits --
>               charging
>             },
>             physicalDeviceFeature = '11010100000'B -- NrBits = 11 --
```

```
>            { -- true bits --
>              buttonInformation,
>              buttonPress,
>              hookswitch,
>              messageWaiting
>            },
>            logicalDeviceFeature = '11111100000110'B -- NrBits = 14 --
>            { -- true bits --
>              agentBusy,
>              agentLoggedOn,
>              agentLoggedOff,
>              agentNotReady,
>              agentReady,
>              agentWorkingAfterCall,
>              doNotDisturb,
>              forwarding
>            },
>            maintenance = '110'B -- NrBits = 3 --
>            { -- true bits --
>              backInService,
>              outOfService
>            },
>            voiceUnit = '1100000'B -- NrBits = 7 --
>            { -- true bits --
>              play,
>              stop
>            }
>          }
>        },
>        miscMonitorCaps = '00001'B -- NrBits = 5 --
>        { -- true bits --
>          acdDeviceExclusiv
>        },
>        callLinkageOptions = '10'B -- NrBits = 2 --
>        { -- true bits --
>          callLinkageFeatureSupported
>        },
>        aCDModels = '10'B -- NrBits = 2 --
>        { -- true bits --
>          visibleACDRelatedDevices
>        },
>        agentLogOnModels = '0001'B -- NrBits = 4 --
>        { -- true bits --
>          logOnACDGroupImplOneStep
>        },
>        agentStateModels = '100'B -- NrBits = 3 --
>        { -- true bits --
>          multiState
>        },
>        connectionView = 0 (fixed),
>        maxLengthParameters
>        { -- SEQUENCE --
>          accountInfo = 11,
>          authCode = 16,
>          agentID = 3,
>          agentPassword = 0,
>          callIDInConnectionID = 2,
>          correlatorData = 0,
>          privateData = 255,
>          deviceIdentifiers = 32,
>          userData = 0,
>          buttonLabel = 0,
>          lampLabel = 0,
>          charactersToSend = 26
>        },
>        servEvtsList
>        { -- SEQUENCE --
```

```
>          capExchangeServList
>          { -- SEQUENCE --
>            getLogicalDeviceInformation = '0000000000000000000000'B -- NrBits = 22 --,
>            getPhysicalDeviceInformation = '000000000'B -- NrBits = 9 --,
>            getSwitchingFunctionCaps NULL,
>            getSwitchingFunctionDevices = '11111111111100'B -- NrBits = 14 --
>            { -- true bits --
>              requestedDeviceID,
>              requestedDeviceCategoryACD,
>              requestedDeviceCategoryACDGroup,
>              requestedDeviceCategoryHuntGroup,
>              requestedDeviceCategoryPickGroup,
>              requestedDeviceCategoryOtherGroup,
>              requestedDeviceCategoryNetwInterface,
>              requestedDeviceCategoryPark,
>              requestedDeviceCategoryRouteingDevice,
>              requestedDeviceCategoryStation,
>              requestedDeviceCategoryVoiceUnit,
>              requestedDeviceCategoryOther
>            },
>            switchingFunctionDevices = '110000'B -- NrBits = 6 --
>            { -- true bits --
>              segmentID,
>              deviceListDeviceCategory
>            }
>          },
>          systemServList
>          { -- SEQUENCE --
>            requestSystemStatus = '0111111111011'B -- NrBits = 13 --
>            { -- true bits --
>              systemStatusInitializing,
>              systemStatusEnabled,
>              systemStatusNormal,
>              systemStatusMessageLost,
>              systemStatusDisabled,
>              systemStatusPartiallyDisabled,
>              systemStatusOverloadImminent,
>              systemStatusOverloadReached,
>              systemStatusOverloadRelieved,
>              switchingFunctionSupportsSending,
>              switchingFunctionSupportsReceiving
>            },
>            systemStatus = '1111111110011'B -- NrBits = 13 --
>            { -- true bits --
>              systemStatusInitializing,
>              systemStatusEnabled,
>              systemStatusNormal,
>              systemStatusMessageLost,
>              systemStatusDisabled,
>              systemStatusPartiallyDisabled,
>              systemStatusOverloadImminent,
>              systemStatusOverloadReached,
>              systemStatusOverloadRelieved,
>              switchingFunctionSupportsSending,
>              switchingFunctionSupportsReceiving
>            }
>          },
>          monitoringServList
>          { -- SEQUENCE --
>            changeMonitorFilter = '00'B -- NrBits = 2 --,
>            monitorStart = '10101000001000'B -- NrBits = 14 --
>            { -- true bits --
>              monitorObjectDevice,
>              requestedMonitorFilter,
>              monitorTypeDevice,
>              monitorExistingCallsInAck
>            },
```

```
>            monitorStop = '0001'B -- NrBits = 4 --
>            { -- true bits --
>              switchingFunctionSupportsReceiving
>            }
>          },
>          snapshotServList
>          { -- SEQUENCE --
>            snapshotCall = '01000000000000'B -- NrBits = 14 --
>            { -- true bits --
>              locaConnectionStateInAck
>            },
>            snapshotDevice = '010000000000'B -- NrBits = 12 --
>            { -- true bits --
>              localCallStateCompoundCallState
>            },
>            snapshotCallData = '01010101'B -- NrBits = 8 --
>            { -- true bits --
>              localConnectionState,
>              mediaServiceVersion,
>              mediaStreamID,
>              privateData
>            },
>            snapshotDeviceData = '01010101010'B -- NrBits = 11 --
>            { -- true bits --
>              localCallStateCompoundCallState,
>              localCallStateUnknown,
>              mediaServiceVersion,
>              mediaStreamID,
>              mediaCallCharacteristics
>            }
>          },
>          callControlServList
>          { -- SEQUENCE --
>            alternateCall = '0100000000010'B -- NrBits = 13 --
>            { -- true bits --
>              hold,
>              ackModelMultiStep
>            },
>            answerCall = '1000000001'B -- NrBits = 10 --
>            { -- true bits --
>              alerting,
>              ackModelMultiStep
>            },
>            clearConnection = '111111000001'B -- NrBits = 12 --
>            { -- true bits --
>              alerting,
>              connected,
>              fail,
>              queued,
>              initiated,
>              hold,
>              ackModelMultiStep
>            },
>            conferenceCall = '1001000100001'B -- NrBits = 13 --
>            { -- true bits --
>              activeCallConnected,
>              heldCallHold,
>              resultingConnectionInfo,
>              ackModelMultiStep
>            },
>            consultationCall = '00000000000000000000000100001'B -- NrBits = 29 --
>            { -- true bits --
>              initiatedCallInfoInAck,
>              ackModelMultiStep
>            },
>            deflectCall = '10000000001'B -- NrBits = 11 --
>            { -- true bits --
```

```
>                 alerting,
>                 ackModelMultiStep
>               },
>               directedPickupCall = '100100010000001'B -- NrBits = 15 --
>               { -- true bits --
>                 alerting,
>                 connected,
>                 pickedCallInAck,
>                 ackModelMultiStep
>               },
>               groupPickupCall = '110000000000001'B -- NrBits = 15 --
>               { -- true bits --
>                 alerting,
>                 connected,
>                 ackModelMultiStep
>               },
>               holdCall = '00001'B -- NrBits = 5 --
>               { -- true bits --
>                 ackModelMultiStep
>               },
>               joinCall = '001000000000000'B -- NrBits = 15 --
>               { -- true bits --
>                 participationTypeSilent
>               },
>               makeCall = '010000000000000000010000001'B -- NrBits = 26 --
>               { -- true bits --
>                 null,
>                 initiatedCallInfoInAck,
>                 ackModelMultiStep
>               },
>               parkCall = '00000001'B -- NrBits = 8 --
>               { -- true bits --
>                 ackModelMultiStep
>               },
>               reconnectCall = '011100001'B -- NrBits = 9 --
>               { -- true bits --
>                 connected,
>                 fail,
>                 initiated,
>                 ackModelMultiStep
>               },
>               retrieveCall = '0001'B -- NrBits = 4 --
>               { -- true bits --
>                 ackModelMultiStep
>               },
>               singleStepTransfer = '0000000100001'B -- NrBits = 13 --
>               { -- true bits --
>                 resultionConnectionInformation,
>                 ackModelMultiStep
>               },
>               transferCall = '1001000100001'B -- NrBits = 13 --
>               { -- true bits --
>                 activeCallConnected,
>                 heldCallHold,
>                 resultingConnectionInformation,
>                 ackModelMultiStep
>               }
>             },
>             callControlEvtsList
>             { -- SEQUENCE --
>               conferenced = '11000000'B -- NrBits = 8 --
>               { -- true bits --
>                 confereceConnectionsEndpointDeviceID,
>                 confereceConnectionsresultingConnectionInfo
>               },
>               connectionCleared = '000000000000000000000'B -- NrBits = 21 --,
>               delivered = '100000000'B -- NrBits = 9 --
```

```
>                { -- true bits --
>                   originatingNIDConnection
>                },
>                diverted = '10000000000'B -- NrBits = 11 --
>                { -- true bits --
>                   callingDevice
>                },
>                established = '100000000'B -- NrBits = 9 --
>                { -- true bits --
>                   originatingNIDConnection
>                },
>                failed = '0000000000'B -- NrBits = 10 --,
>                held = '0000000'B -- NrBits = 7 --,
>                netwReached = '00000000000000000000000000000001'B -- NrBits = 33 --
>                { -- true bits --
>                   privateData
>                },
>                originated = '00000000'B -- NrBits = 8 --,
>                queued = '1100000000'B -- NrBits = 10 --
>                { -- true bits --
>                   numberQueued,
>                   callsInFront
>                },
>                retrieved = '000000'B -- NrBits = 6 --,
>                serviceInitiated = '0000000'B -- NrBits = 7 --,
>                transferred = '000000001000000000000'B -- NrBits = 21 --
>                { -- true bits --
>                   typeOfChargingInfoTotal
>                }
>             },
>             callAssociatedServList
>             { -- SEQUENCE --
>                generateDigits = '00000000000'B -- NrBits = 11 --,
>                generateTelephonyTones =
'0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000'B -- NrBits = 126 --
>             },
>             callAssociatedEvtsList
>             { -- SEQUENCE --
>                charging = '1000000010000'B -- NrBits = 13 --
>                { -- true bits --
>                   numberUnitsNumberOfChargingUnits,
>                }
>             },
>             physDevServList
>             { -- SEQUENCE --
>                buttonPress = '000'B -- NrBits = 3 --,
>                getButtonInformation = '100011000'B -- NrBits = 9 --
>                { -- true bits --
>                   button,
>                   buttonFunctionInAck,
>                   buttonAssociatedNumberInAck
>                },
>                getHookSwitchStatus = '100'B -- NrBits = 3 --
>                { -- true bits --
>                   hookSwitch
>                },
>                getMessageWaitingIndicator = '0110'B -- NrBits = 4 --
>                { -- true bits --
>                   deviceForMsgInAck,
>                   lampIsPresentInAck
>                },
>                getMicrophoneMute = '100'B -- NrBits = 3 --
>                { -- true bits --
>                   auditoryApparatus
>                },
>                getSpeakerMute = '100'B -- NrBits = 3 --
```

```
>             { -- true bits --
>               auditoryApparatus
>             },
>             getSpeakerVolume = '1010'B -- NrBits = 4 --
>             { -- true bits --
>               auditoryApparatus,
>               speakerVolAbsInAck
>             },
>             setDisplay = '0000000'B -- NrBits = 7 --,
>             setLampMode =
'0111100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'B --
NrBits = 208 --
>             { -- true bits --
>               lampModeFlutter,
>               lampModeOff,
>               lampModeSteady,
>               lampModeWink
>             },
>             setMessageWaitingIndicator = '0000'B -- NrBits = 4 --,
>             setMicrophoneMute = '001'B -- NrBits = 3 --
>             { -- true bits --
>               ackModelMultiStep
>             },
>             setSpeakerMute = '001'B -- NrBits = 3 --
>             { -- true bits --
>               ackModelMultiStep
>             },
>             setSpeakerVolume = '1100010'B -- NrBits = 7 --
>             { -- true bits --
>               speakerVolumeAbs,
>               speakerVolumeInc,
>               resettedAfterCall
>             }
>           },
>           physDevEvtsList
>           { -- SEQUENCE --
>             buttonPress = '000'B -- NrBits = 3 --,
>             hookswitch = '1'B -- NrBits = 1 --
>             { -- true bits --
>               privateData
>             },
>             messageWaiting = '10'B -- NrBits = 2 --
>             { -- true bits --
>               deviceForMsg
>             }
>           },
>           logicalServList
>           { -- SEQUENCE --
>             callBackNonCallRel = '1000'B -- NrBits = 4 --
>             { -- true bits --
>               privateData
>             },
>             cancelCallBackMsg = '0000'B -- NrBits = 4 --,
>             getAgentState = '00000000'B -- NrBits = 8 --,
>             getAutoAnswer = '000'B -- NrBits = 3 --,
>             getAutoWorkMode = '010'B -- NrBits = 3 --
>             { -- true bits --
>               autoWorkIntervalInAck
>             },
>             getCallerIDStatus = '00'B -- NrBits = 2 --,
>             getDoNotDisturb = '000000'B -- NrBits = 6 --,
>             getForwarding = '001000000000110000000'B -- NrBits = 21 --
>             { -- true bits --
>               forwardListImmediateInAck,
>               forwardListImmIntInAck,
>               forwardListImmExtInAck
```

```
>            },
>            setAgentState = '111111000000000000'B -- NrBits = 18 --
>            { -- true bits --
>              requestedAgentStateLoggedOn,
>              requestedAgentStateLoggedOff,
>              requestedAgentStateNotReady,
>              requestedAgentStateReady,
>              requestedAgentStateWorkingAfterCall,
>              agentID
>            },
>            setCallerIDStatus = '000'B -- NrBits = 3 --,
>            setDoNotDisturb = '000000'B -- NrBits = 6 --,
>            setForwarding = '00000000011100000'B -- NrBits = 17 --
>            { -- true bits --
>              forwardingTypeImmediate,
>              forwardingTypeImmInt,
>              forwardingTypeImmExt
>            }
>          },
>          logicalEvtsList
>          { -- SEQUENCE --
>            agentBusy = '00000000'B -- NrBits = 8 --,
>            agentLoggedOff = '00000'B -- NrBits = 5 --,
>            agentLoggedOn = '01111'B -- NrBits = 5 --
>            { -- true bits --
>              acdGroup,
>              agentPassword,
>              cause,
>              privateData
>            },
>            agentNotReady = '0000'B -- NrBits = 4 --,
>            agentReady = '0000'B -- NrBits = 4 --,
>            agentWorkingAfterCall = '0000000'B -- NrBits = 7 --,
>            doNotDisturb = '00000'B -- NrBits = 5 --,
>            forwarding = '000000001111111111'B -- NrBits = 18 --
>            { -- true bits --
>              forwardingTypeImmediate,
>              forwardingTypeImmInt,
>              forwardingTypeImmExt,
>            }
>          },
>          deviceMaintEvtsList
>          { -- SEQUENCE --
>            backInService = '11'B -- NrBits = 2 --
>            { -- true bits --
>              cause,
>              privateData
>            },
>            outOfService = '11'B -- NrBits = 2 --
>            { -- true bits --
>              cause,
>              privateData
>            }
>          },
>          voiceUnitServList
>          { -- SEQUENCE --
>            playMsg = '000000000'B -- NrBits = 9 --,
>            stop = '00'B -- NrBits = 2 --
>          },
>          voiceUnitEvtsList
>          { -- SEQUENCE --
>            play = '111111'B -- NrBits = 6 --
>            { -- true bits --
>            },
>            stop = '111111'B -- NrBits = 6 --
>            { -- true bits --
>            }
```

```
>          },
>          cdrServList
>          { -- SEQUENCE --
>            cdrReport = '0000111111101000000100111100000001101010100100000001000000'B -- NrBits = 57 --
>            { -- true bits --
>              recordCreationTime,
>              callingDevice,
>              calledDevice,
>              assocCallingDevice,
>              assocCalledDevice,
>              netwCallingDevice,
>              netwCalledDevice,
>              mediaCallCharacteristics,
>              tarifTable,
>              connectionDuration,
>              accessCode,
>              carrier,
>              selectedRoute,
>              chargingInfo,
>              suppServiceInfoNormalCall,
>              suppServiceInfoTransferCall,
>              suppServiceInfoCallForwarding,
>              suppServiceInfoConferencing,
>              suppServiceInfoOther,
>              accountInfo
>            },
>            startCDRTransmission = '10000'B -- NrBits = 5 --
>            { -- true bits --
>              transferModeTransferAtEndOfCall
>            },
>            stopCDRTransmission = '00000001'B -- NrBits = 8 --
>            { -- true bits --
>              swFunctionSupportsReceiving
>            }
>          },
>          statusReportingServList
>          { -- SEQUENCE --
>            monitoringServices
>            { -- SEQUENCE --
>              changeMonitorFilter = '00'B -- NrBits = 2 --,
>              monitorStart = '10101000001000'B -- NrBits = 14 --
>              { -- true bits --
>                monitorObjectDevice,
>                requestedMonitorFilter,
>                monitorTypeDevice,
>                monitorExistingCallsInAck
>              },
>              monitorStop = '0001'B -- NrBits = 4 --
>              { -- true bits --
>                switchingFunctionSupportsReceiving
>              }
>            },
>            snapshotServices
>            { -- SEQUENCE --
>              snapshotCall = '01000000000000'B -- NrBits = 14 --
>              { -- true bits --
>                locaConnectionStateInAck
>              },
>              snapshotDevice = '010000000000'B -- NrBits = 12 --
>              { -- true bits --
>                localCallStateCompoundCallState
>              },
>            },
>            systemServices
>            { -- SEQUENCE --
>              requestSystemStatus = '0111111111011'B -- NrBits = 13 --
>              { -- true bits --
```

```
>                 },
>                 systemStatus = '1111111110010'B -- NrBits = 13 --
>                 { -- true bits --
>                   systemStatusInitializing,
>                   systemStatusEnabled,
>                   systemStatusNormal,
>                   systemStatusMessageLost,
>                   systemStatusDisabled,
>                   systemStatusPartiallyDisabled,
>                   systemStatusOverloadImminent,
>                   systemStatusOverloadReached,
>                   systemStatusOverloadRelieved,
>                   switchingFunctionSupportsSending
>                 }
>               }
>             }
>           },
>           systemStatusTimer = 15
>         }
>       }
> }
```

2.17.4 Button and Lamp map

The button-IDs in the following description are in hexadecimal format. The button-IDs are the same like the lamp-IDs (if lamp available).

For all phones the keypad is mapped to the following button-numbers:

Key	Button-ID (hex)
Dialpad: 1	0x01
Dialpad: 2	0x02
Dialpad: 3	0x03
Dialpad: 4	0x04
Dialpad: 5	0x05
Dialpad: 6	0x06
Dialpad: 7	0x07
Dialpad: 8	0x08
Dialpad: 9	0x09
Dialpad: *	0x0B
Dialpad: 0	0x0A
Dialpad: #	0x0C

Only OptiPoint-Phones:

Key	Button-ID (hex)
+	0x0D
-	0x0E
<	0x0F
>	0x10
√	0x11

2.17.4.1 optiPoint 410 entry

Button-ID (hex)
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19

2.17.4.2 optiPoint 410 economy / economy plus / standard

Button-ID (hex)
0x12
0x13
0x14
0x15

Button-ID (hex)
0x16
0x17
0x18
0x19
0x1A
0x1B
0x1C
0x1D

2.17.4.3 optiPoint 410 advanced

Button-ID (hex)
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19
0x1A
0x1B
0x1C
0xF2
0xF3
0xF4
0xF5
0xF6
0xF7
0xF8
0x1D

2.17.4.4 optiPoint 420 economy, economy plus /standard

Button-ID (hex)
0x12
0x13
0x14
0x15
0x16

Button-ID (hex)
0x17
0x18
0x19
0x1A
0x1B
0x1C
0x1D

2.17.4.5 optiPoint 420 advanced

Button-ID (hex)
0x12
0x13
0x14
0x15
0x16

Button-ID (hex)
0x17
0x18
0x19
0x1A
0x1B
0x1C
0xF2
0xF3
0xF4
0xF5
0xF6
0xF7
0x1D

2.17.4.6 optiPoint self labeling key module

Button-ID (hex) 1st module	Button-ID (hex) 2nd module
0x1E	0x2E
0x1F	0x2F
0x20	0x30
0x21	0x31
0x22	0x32
0x23	0x33
0x24	0x34
0x25	0x35
0x26	0x36
0x27	0x37
0x28	0x38
0x29	0x39
0x2D	0x3D

2.17.4.7 Openstage 20

Key	Button-ID (hex)
Disconnect	0x17
Redial	0x13
Call forward	0x16
Mute	0x14
Key	Button-ID (hex)
-	0x0E
loudspeakr	0x12
+	0x0D

2.17.4.8 Openstage 40

Button-ID (hex)
0x1A
0x1B
0x1C
0xF2
0xF3
0x1D

Other keys like OpenStage 20

2.17.4.9 Openstage 60/80

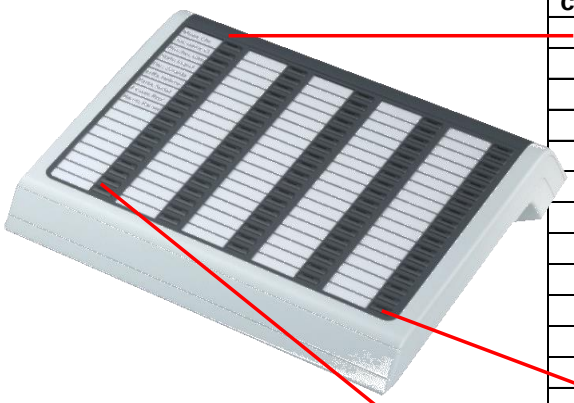
Key	Button-ID (hex)
	0x1A
	0x1B
	0x1C
	0xF2
	0xF3
	0xF4
	0xF5
(only OS 80)	0xF6
	0x1D

Other keys like OpenStage 20

2.17.4.10 Openstage KeyModule

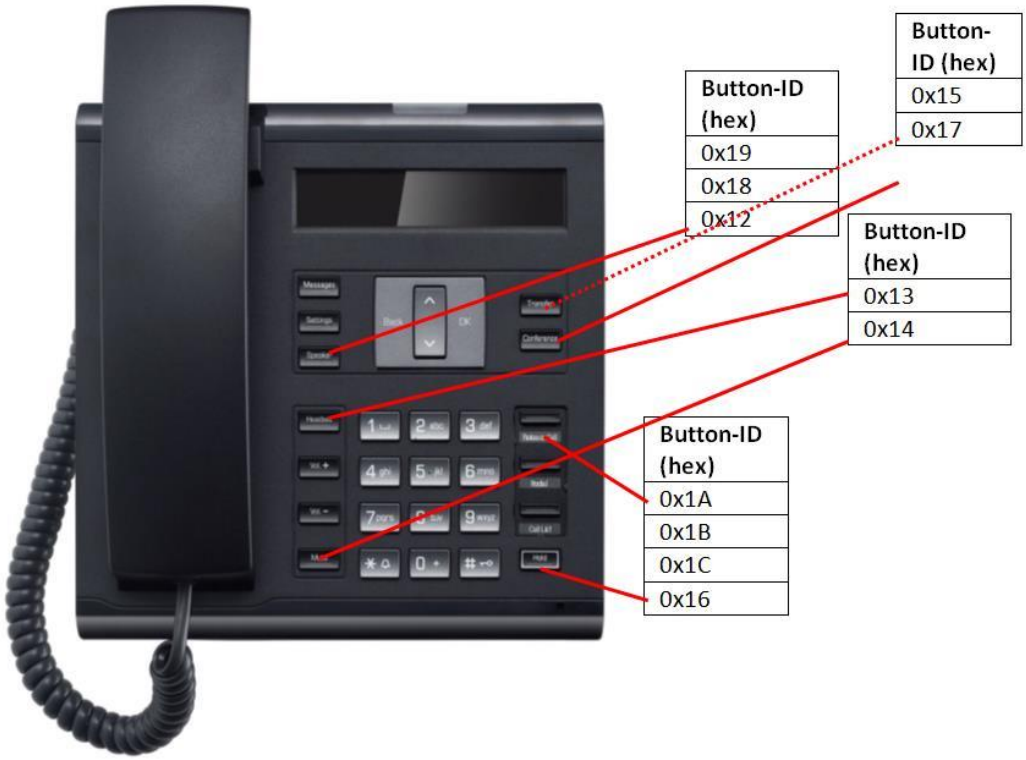
Button-ID (hex) 1st module	Button-ID (hex) 2nd module
0x1E	0x2E
0x1F	0x2F
0x20	0x30
0x21	0x31
0x22	0x32
0x23	0x33
0x24	0x34
0x25	0x35
0x26	0x36
0x27	0x37
0x28	0x38
0x29	0x39

2.17.4.11 OpenStage Busy Lamp Field

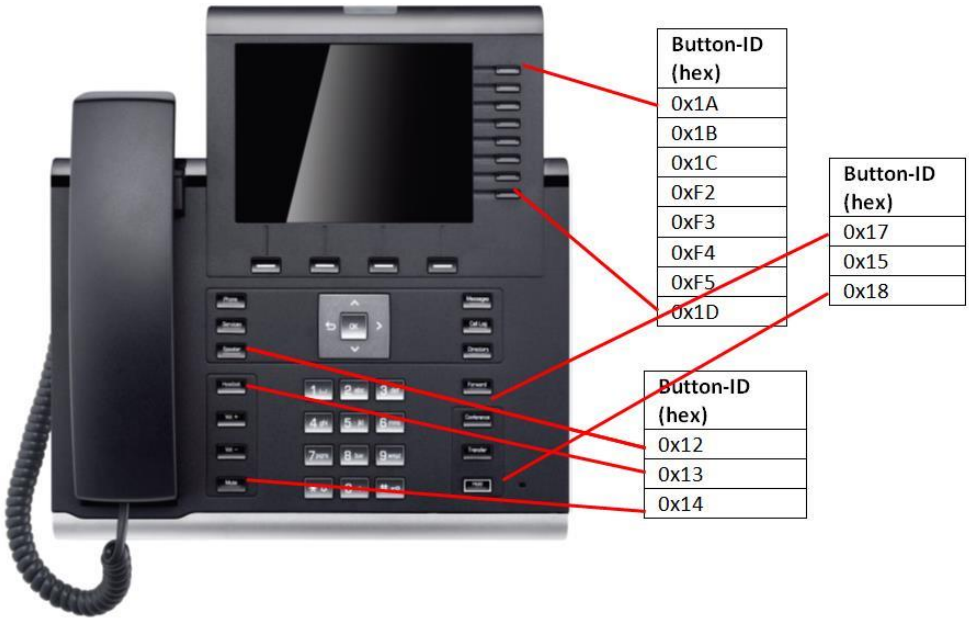


Button-ID (hex) 1st column	Button-ID (hex) 2nd column	Button-ID (hex) 3rd column	Button-ID (hex) 4th column	Button-ID (hex) 5th column
0x3E	0x50	0x62	0x74	0x86
0x3F	0x51	0x63	00x75	0x87
0x40	0x52	0x64	0x76	0x88
0x41	0x53	0x65	0x77	0x89
0x42	0x54	0x66	0x78	0x8A
0x43	0x55	0x67	0x79	0x8B
0x44	0x56	0x68	0x7A	0x8C
0x45	0x57	0x69	0x7B	0x8D
0x46	0x58	0x6A	0x7C	0x8E
0x47	0x59	0x6B	0x7D	0x8F
0x48	0x5A	0x6C	0x7E	0x90
0x49	0x5B	0x6D	0x7F	0x91
0x4A	0x5C	0x6E	0x80	0x92
0x4B	0x5D	0x6F	0x81	0x93
0x4C	0x5E	0x70	0x82	0x94
0x4D	0x5F	0x71	0x83	0x95
0x4E	0x60	0x72	0x84	0x96
0x4F	0x61	0x73	0x85	0x97

2.17.4.12 OpenScape Desk Phone IP 35G



2.17.4.13 OpenScape Desk Phone IP 55G



2.17.5 Key Module 55

Button-ID (hex)
0x1E
0x1F
0x20
0x21
0x22
0x23
0x24
0x25
0x26
0x27
0x28
0x29

2.18 OpenScape Business

2.18.1 OpenScape Business Parameter Constraints

2.18.1.1 Single and slave nodes

Parameter	Maximum value
Max. amount of concurrent MonitorCrossRefID	1500
Max. amount of concurrent Displaybuffers for SetDisplay CSTA Service only	64
Max. amount of waiting calls in any hunt group	15
Max. amount of waiting calls in any UCD group (value can be reduced via administration tool)	30
Max. amount of waiting calls in last UCD group (value can be reduced via administration tool)	72
Max. amount of waiting calls connected to an announcement port	10

2.18.1.2 Master-node of homogenous network

Parameter	Maximum value
Max. amount of concurrent MonitorCrossRefID	2000
Max. amount of concurrent Displaybuffers for SetDisplay CSTA Service only	64
Max. amount of waiting calls in any hunt group	15
Max. amount of waiting calls in any UCD group (value can be reduced via administration tool)	30
Max. amount of waiting calls in last UCD group (value can be reduced via administration tool)	72
Max. amount of waiting calls connected to an announcement port	10

2.18.2 OpenScape Business CSTA link Communication Constraints

Parameter	Min. / Max. value
Min. recommended time gap between any two CSTA messages sent to OpenScape Business (CSTA link over IP)	50 ms
Max. time gap between systemStatus service request and service response (Check SystemStatusTimer in GetSwitchingFunctionCapabilitiesResult)	15 s (default)

2.18.3 List of monitorable OpenScape Business Devices

OpenScape Business devices that can be monitored and generate an event report are:

- virtual stations
- UCD groups
- Trunks
- Mulap (Multi Line Appearance), Basic-, Executive, Team, TOP
- analog phones (P.O.T, a/b device)
- OpenScape Personal Edition (HFA and SIP)
- OpenStage 5, 15, 20, 20E, 20G, 40, 40G, 60, 60G, 80, 80G HFA and SIP
- OpenStage key module (only for OpenStage 15, 40, 60 and 80)
- OpenScape Desk Phone IP 35G/55G (SIP)
- optiPoint WL2/WL3
- System telephones (HFA): optiPoint 410 entry, optiPoint 410 economy, optiPoint 410 economy plus, optiPoint 410 standard, optiPoint 410 advanced und optiPoint 420 economy, optiPoint 420 economy plus, optiPoint 420 standard, optiPoint 420 advanced, OptiPoint 500
- Optional add-on devices: optiPoint key module, optiPoint BLF, optiPoint self labeling key module, optiPoint application module.
- Cordless (DECT)

2.18.4 CSTA Services Overload-Rejection

There is an overload rejection algorithm implemented for the following CSTA Services:

- MakeCall

The overload rejection algorithm is activated automatically if a number of service request limit per time interval is exceeded. The algorithm reflects peak and average limits. If a MakeCall-request is rejected because of overload, then an error roer.performanceManagement.performanceLimitExceeded is returned.

2.18.5 Device Identifier

With OpenScape Business there are several possibilities, how trunk-access-numbers can show up on CSTA:

In OpenScape Business they can be configured with only digits: e.g. ‚7801‘, or they can be configured with leading Hashes, e.g. ‚##700‘.

Trunk access numbers do not need to be unique, therefor the CSTA access point of the master-node of a network of OpenScape Business uses the SFR presentation to make the trunk access numbers unique using the NodeID of each node:

e.g. the trunk 7801 in node 56 is presented as ‚N<7801>Node056‘, the trunk ##704 in node 3 as ‚N<##704>Node003‘:

	Trunk access number	Presentation on CSTA
Single node	7801	7801
slave node	7801	7801
master node	7801	N<7801>Nodexxx (xxx is the node ID)

With closed numbering plan all other device IDs must be unique and are presented on CSTA unchanged.

2.18.6 Dial string formats

Application can use the format “Dialable Addresses” or “Canonical Addresses” to create calls.

2.18.6.1 Dialable Addresses

Dialable addresses are used by OpenScape Business without any transformation. Only digits 0…9 are allowed. The address has to be like the digits dialled manually on phone, including trunk access code for external calls.

2.18.6.2 Canonical Addresses

Canonical addresses are also accepted as destination. Canonical addresses have to start with a '+' and followed by digits 0…9 only. Whitespaces are not allowed.
Supported formats like:

+498970070

Not supported formats like:

+49(89)7007-0
+49(089)7007-0
+49(0)8970070
+49 89 70070

Please note that canonical addresses can only be used as destination. In event report and results device ids are always presented as dialable address. Services which are able to handle canonical addresses are:

- Consultation Call
- Deflect Call
- Make Call
- Single Step Transfer Call

2.18.7 Hold-service

In OpenScapeBusiness there are two different 'hold'-services:

1. 'exclusive hold': i.e. consultation, only the holding party can unhold. This is done through phone or consultCall-service on CSTA. After initiating 'exclusive hold' always a new call is started on the phone. I.e. the new call is implicitly created through this service. Event cause 'consultation' is reported via HeldEvent.
2. 'common hold': this can be done through 'hold'-key on the phone and hold-service on CSTA. Everyone can unhold these calls.After initiating 'common hold' there is no implicit creation of a new call. Only if phone is off-hook this creates a new call. Event cause 'normal' is reported via HeldEvent.

Copyright © Unify Software and Solutions GmbH & Co. KG 2015
Mies-van-der-Rohe-Str. 6, 80807 Munich/Germany
All rights reserved.

The information provided in this document contains merely general descriptions or characteristics of performance which in case of actual use do not always apply as described or which may change as a result of further development of the products. An obligation to provide the respective characteristics shall only exist if expressly agreed in the terms of contract.

Availability and technical specifications are subject to change without notice.

Unify, OpenScape, OpenStage and HiPath are registered trademarks of Unify Software and Solutions GmbH & Co. KG. All other company, brand, product and service names are trademarks or registered trademarks of their respective holders.

unify Harmonize your enterprise