3GPP TSG CN Plenary, Meeting #11 ***Tdoc NP-010134***
Palm Springs, USA. 14th - 16th March 2001

Source: **CN WG5**

Title: **CR 29.198: for moving 29.198 from R99 to Rel 4 (N5-010158)**

Agenda item: **6.5.2**

Document for: **Approval**

Doc-	Doc-	Spec	CR	R	Phas	Subject	Cat	Versio	Versi	Meeti	Workit
NP-010134	N5-010158	29.198	047		Rel4	**Add new features and Split R99 into a multi-part TS for upgrading to Rel 4**	B	3.2.0	4.0.0	N5-10	OSA1

Structure of the OSA API (29.198) and Mapping (29.998) documents

The Open Service Access (OSA) Application Programming Interface (API) specifications consist of two sets of documents:

API specification (3GPP TS 29.198)
The Parts of 29.198 - apart from Part 1 and Part 2 - define the interfaces, parameters and state models that belong to the API specification. UML (Unified Modelling Language) is used to specify the interface classes.
As such it provides a UML interface class description of the methods (API calls) supported by that interface and the relevant parameters and types. The interfaces are specified in IDL (Interface Description Language).

Mapping specification of the OSA APIs and network protocols (3GPP TR 29.998)
The Parts of 29.998 contain a possible mapping from the APIs defined in 29.198 to various network protocols (i.e. MAP [7], CAP [8], etc.). It is an informative document, since this mapping is considered as implementation- / vendor-dependent. On the other hand this mapping will provide potential service designers with a better understanding of the relationship of the OSA API interface classes and the behaviour of the network associated to these interface classes.

The purpose of the OSA API is to shield the complexity of the network, its protocols and specific implementation from the applications. This means that applications do not have to be aware of the network nodes, a Service Capability Server interacts with, in order to provide the Service Capability Features (SCF) to the application. The specific underlying network and its protocols are transparent to the application.

The **API specification** (3GPP TS 29.198) is structured in the following Parts:

29.198-1	Part 1:	Overview
29.198-2	Part 2:	Common Data Definitions
29.198-3	Part 3:	Framework
29.198-4	Part 4:	Call Control SCF
29.198-5	Part 5:	User Interaction SCF
29.198-6	Part 6:	Mobility SCF
29.198-7	Part 7:	Terminal Capabilities SCF
29.198-8	Part 8:	Data Session Control SCF
29.198-9	Part 9:	Generic Messaging SCF
29.198-10	Part 10:	Connectivity Manager SCF
29.198-11	Part 11:	Account Management SCF
29.198-12	Part 12:	Charging SCF

The **Mapping specification of the OSA APIs and network protocols** (3GPP TR 29.998) is also structured as above.
A mapping to network protocols is however not applicable for all Parts, but the numbering of Parts is kept.
Also in case a Part is not supported in a Release, the numbering of the parts is maintained.

3GPP TSG-CN5 Meeting #10
Antwerp, BELGIUM, 5 – 7 March 2001

Tdoc N5-010158

CR-Form-v3

CHANGE REQUEST

⌘ **29.198** **CR** **047** ⌘ rev **-** ⌘ Current version: **3.2.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: ⌘ (U)SIM ☐ ME/UE ☐ Radio Access Network ☐ Core Network X

Title:	⌘	**Add new features and Split R99 into a multi-part TS for upgrading to Rel 4**
Source:	⌘	CN5
Work item code:	⌘	OSA1-ECOM OSA1-CSCF OSA1-SEC OSA1-TC OSA1-LCSI
Date:	⌘	07/03/2001
Category:	⌘	**B** *Use one of the following categories* ***F*** *(essential correction)* ***A*** *(corresponds to a correction in an earlier release)* ***B*** *(Addition of feature),*
Release:	⌘	Rel 4 *Use one of the following releases:* *R99 (Release 1999)* *REL-4 (Release 4)* *REL-5 (Release 5)*

Reason for change: **Addition of features** (CN5's Work Tasks under Feature: OSA enhancements in the 3GPP Work Plan):

Interactions OSA - e-commerce	(Stages 2/3),
OSA APIs for MuMa CC	(Stages 2/3),
OSA security	(security related SCF(s) definition),
Retrieval of Terminal capabilities	(Stages 2/3),
LCS - OSA interfaces	(Stages 3)

Summary of change: **Split the TS into multiple documents (muliti-part TS) as follows:**

29.198-1	Part 1:	Overview	(to be put by CN#11 under Change Control)
29.198-2	Part 2:	Common Data Definitions	(to be put by CN#11 under Change Control)
29.198-3	Part 3:	Framework	(to be put by CN#11 under Change Control)
29.198-4	Part 4:	Call Control SCF	(to go as version 1.0.0 for Information to CN#11)
29.198-5	Part 5:	User Interaction SCF	(to be put by CN#11 under Change Control)
29.198-6	Part 6:	Mobility SCF	(to be put by CN#11 under Change Control)
29.198-7	Part 7:	Terminal Capabilities SCF	(to be put by CN#11 under Change Control)
29.198-8	Part 8:	Data Session Control SCF	(to be put by CN#11 under Change Control)
29.198-9	Part 9:	Void for Rel 4	(current thinking: Generic Messaging SCF)
29.198-10	Part 10:	Void for Rel 4	(current thinking: Connectivity Manager SCF)
29.198-11	Part 11:	Account Management SCF	(to go as version 1.0.0 for Information to CN#11)
29.198-12	Part 12:	Charging SCF	(to go as version 1.0.0 for Information to CN#11)

Consequences if not approved: ⌘ OSA will not be available for 3GPP Rel4/5

Clauses affected: ⌘ All

Other specs affected: ⌘

☐	Other core specifications
☐	Test specifications
☐	O&M Specifications

Other comments: The **Mapping specification of the OSA APIs and network protocols** (3GPP **TR 29.998**) is also structured as above. A mapping to network protocols is however not applicable for all Parts, but the numbering of Parts is kept. Also in case a Part is not supported in a Release, the numbering of the parts is maintained.

3GPP TS 29.198-1 V1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface;
Part 1: Overview;
(Release 4)

Keywords
API, OSA

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

The present document is the first part of the 3GPP Specification defining the Application Programming Interface (API) for Open Service Access (OSA), and provides an overview of the content and structure of the various parts of this specification, and of the relation to other standards documents .

The OSA-specifications define an architecture that enables service application developers to make use of network functionality through an open standardised interface, i.e. the OSA APIs. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

This specification has been defined jointly between ETSI SPAN12, 3GPP TSG CN WG5 and the Parlay consortium [24], in co-operation with the JAIN consortium [25].

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

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

[1] 3GPP TR 21.905: "3G Vocabulary".

3GPP TS 22.121: "Service aspects; The Virtual Home Environment (Release 4)".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

3GPP TS 23.057: "Mobile Station Application Execution Environment (MExE)".

[4] 3GPP TS 23.078: "CAMEL Phase 3, stage 2".

[5] 3GPP TS 22.101: "Universal Mobile Telecommunications System (UMTS): Service Aspects; Service Principles".

[6] World Wide Web Consortium Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation (www.w3.org).

[7] 3GPP TS 29.002: "Mobile Application Part (MAP)".

[8] 3GPP TS 29.078: "CAMEL Phase 3, , CAMEL Application Part (CAP) Specification".

[9] Wireless Application Protocol (WAP), Version 1.2, UAProf Specification (www.wapforum.org).

[10] Wireless Application Protocol (WAP), version 1.2, WAP Service Indication specification, (www.wapforum.org).

[11] Wireless Application Protocol (WAP), version 1.2, WAP Push Architecture Overview (www.wapforum.org).

[12] Wireless Application Protocol (WAP), version 1.2, WAP Architecture (www.wapforum.org).

[13] SUN IDL Compiler (www.javasoft.com/products/jdk/idl/index.html).

[14] UML Unified ModellingLanguage (www.rational.com/uml).

[15] Object Management Group (www.omg.org).

[16] 3GPP TS 22.002: "Circuit Bearer Services supported by a PLMN".

[17] 3GPP TS 22.003: "Circuit Teleservices supported by a PLMN".

[18] 3GPP TS 24.002: "Public Land Mobile Network (PLMN) Access Reference Configuration".

[19] ITU-T Q.763: "Signalling System No. 7 – ISDN user part formats and codes".

[20] ITU-T Q.931: "ISDN user-network interface layer 3 specification for basic call control".

[21] ISO 8601: "Data elements and interchange formats -- Information interchange -- Representation of dates and times".

[22] ISO 4217: "Codes for the representation of currencies and funds".

[23] 3GPP TS 23.127v4: "Service Requirements for Open Service Access".

[24] http://www.parlay.org

[25] http://www.jain.org

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of this specification, the following definitions apply:

Applications: Services, which are designed using service capability features.

Gateway: Synonym for Service Capability Server. From the viewpoint of applications, a Service Capability Server can be seen as a gateway to the core network.

HE-VASP: Home Environment Value Added Service Provider. This is a VASP that has an agreement with the Home Environment to provide services.

Home Environment: responsible for overall provision of services to users

Local Service: A service, which can be exclusively provided in the current serving network by a Value Added Service Provider.

OSA Interface: Standardised Interface used by application to access service capability features.

Personal Service Environment: contains personalised information defining how subscribed services are provided and presented towards the user. The Personal Service Environment is defined in terms of one or more User Profiles.

Service Capabilities: Bearers defined by parameters, and/or mechanisms needed to realise services. These are within networks and under network control.

Service Capability Feature: Functionality offered by service capabilities that are accessible via the standardised OSA interface

Service Capability Server: Functional Entity providing OSA interfaces towards an application

Service: term used as an alternative for Service Capability Feature in this specification

User Interface Profile: Contains information to present the personalised user interface within the capabilities of the terminal and serving network.

User Profile: This is a label identifying a combination of one user interface profile, and one user services profile.

User Services Profile: Contains identification of subscriber services, their status and reference to service preferences.

Value Added Service Provider: provides services other than basic telecommunications service for which additional charges may be incurred.

Virtual Home Environment: A concept for personal service environment portability across network boundaries and between terminals.

Further definitions are given in 3GPP TS 22.101 [5].

3.2 Abbreviations

For the purposes of this Standard the following abbreviations apply:

API	Application Programming Interface
CAMEL	Customised Application for Mobile network Enhanced Logic
CAP	CAMEL Application Part
CSE	Camel Service Environment
HE	Home Environment
HE-VASP	Home Environment - Value Added Service Provider
HLR	Home Location Register
INAP	Intelligent Networks Application Part
IDL	Interface Description Language
MAP	Mobile Application Part
ME	Mobile Equipment
MExE	Mobile Station (Application) Execution Environment
MS	Mobile Station
MSC	Mobile Switching Centre
OSA	Open Service Access
PLMN	Public Land Mobile Network
PSE	Personal Service Environment
SAT	SIM Application Tool-Kit
SCF	Service Capability Feature
SCP	Service Control Point
SIM	Subscriber Identity Module
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
UE	User Equipment
USIM	User Service Identity Module
VLR	Visited Location Register
VASP	Value Added Service Provider
VHE	Virtual Home Environment
WAP	Wireless Application Protocol
WGP	Wireless Gateway Proxy
WPP	Wireless Push Proxy

Further abbreviations are given in the 3GPP TR 21.905 [1].

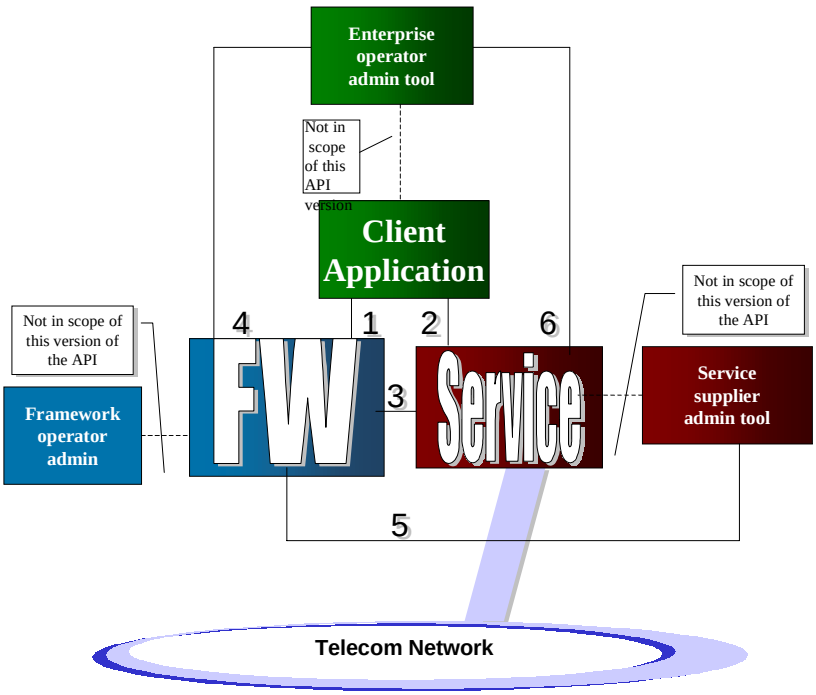
4 Open Service Access APIs

The OSA-specifications define an architecture that enables service application developers to make use of network functionality through an open standardised interface, i.e. the OSA APIs. The network functionality is describes as Service Capability Features or Services [note]. The OSA Framework is a general component in support of Services (Service Capabilities) and Applications. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

The OSA API is split into three types of interface classes, Service and Framework.

- Interface classes between the Applications and the Framework, that provide applications with basic mechanisms (e.g. Authentication) that enable them to make use of the service capabilities in the network.
- Interface classes between Applications and Service Capability Features (SCF), which are individual services that may be required by the client to enable the running of third party applications over the interface e.g. Messaging type service.
- Interface classes between the Framework and the Service Capability Features, that provide the mechanisms necessary for multi-vendorship.

These interfaces represent interfaces 1, 2 and 3 of the Figure below. The other interfaces are not yet part of the scope of the work.



Within the OSA concept a set of Service Capability Features has been specified. The OSA documentation is structured in parts. The first Part (this document) contains an overview, the second Part contains common data definitions, the third Part the Framework interfaces. The rest of the Parts contain the description of the SCF's.

NOTE:
The terms 'Service' and 'Service Capability Feature' are used as alternatives for the same concept in this specification. In the OSA API itself the Service Capability Features as identified in the 3GPP requirements and architecture are reflected as 'service', in terms like serviceFactory, serviceDiscovery.

5 Structure of the OSA API (29.198) and Mapping (29.998) documents

The Open Service Access (OSA) Application Programming Interface (API) specifications consist of two sets of documents:

API specification (3GPP TS 29.198)
The Parts of 29.198 - apart from Part 1 (the present document) and Part 2 - define the interfaces, parameters and state models that belong to the API specification. UML (Unified Modelling Language) is used to specify the interface classes.
As such it provides a UML interface class description of the methods (API calls) supported by that interface and the relevant parameters and types. The interfaces are specified in IDL (Interface Description Language).

Mapping specification of the OSA APIs and network protocols (3GPP TR 29.998)
The Parts of 29.998 contain a possible mapping from the APIs defined in 29.198 to various network protocols (i.e. MAP [7], CAP [8], etc.). It is an informative document, since this mapping is considered as implementation- / vendor-dependent. On the other hand this mapping will provide potential service designers with a better understanding of the relationship of the OSA API interface classes and the behaviour of the network associated to these interface classes.

The purpose of the OSA API is to shield the complexity of the network, its protocols and specific implementation from the applications. This means that applications do not have to be aware of the network nodes, a Service Capability Server interacts with, in order to provide the Service Capability Features (SCF) to the application. The specific underlying network and its protocols are transparent to the application.

The **API specification** (3GPP TS 29.198) is structured in the following Parts:

29.198-1	Part 1:	Overview
29.198-2	Part 2:	Common Data Definitions
29.198-3	Part 3:	Framework
29.198-4	Part 4	Call Control SCF
29.198-5	Part 5	User Interaction SCF
29.198-6	Part 6	Mobility SCF
29.198-7	Part 7	Terminal Capabilities SCF
29.198-8	Part 8	Data Session Control SCF
29.198-9	Part 9	Generic Messaging SCF
29.198-10	Part 10	Connectivity Manager SCF
29.198-11	Part 11	Account Management SCF
29.198-12	Part 12	Charging SCF

The **Mapping specification of the OSA APIs and network protocols** (3GPP TR 29.998) is also structured as above. A mapping to network protocols is however not applicable for all Parts, but the numbering of Parts is kept.
Also in case a Part is not supported in a Release, the numbering of the parts is maintained.

Sub-structure of the Parts of 29.198

The Parts with API specification themselves are sub-structured as follows:

- The Sequence diagrams give the reader a practical idea of how each of the service capability feature is implemented.
- The Class relationships section show how each of the interfaces applicable to the SCF, relate to one another

- The Interface specification section describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the progression of internal processes either in the application, or Gateway.
- The Data definitions clauses show a detailed expansion of each of the data types associated with the methods within the classes. It is to be noted that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.
- IDL description of the interface (normative Annex)

6 Methodology

Following is a description of the methodology used for the establishment of API specification for OSA.

6.1 Tools and Languages

The Unified Modelling Language (UML) [14] is used as the means to specify class and state transition diagrams. Additionally, Object Management Group's (OMG) [15] Interface Definition Language (IDL) is used as the means to programmatically define the interfaces. IDL files are either generated manually from class diagrams or by using a UML tool. In the case IDLs are manually written and/or being corrected manually, correctness has been verified using a CORBA2 (orbos/97-02-25) compliant IDL compiler, e.g. [13].

6.2 Packaging

A hierarchical packaging scheme is used to avoid polluting the global name space. The root is defined as:

org.open_service_access

Note that the CORBA module hierarchy defined in the IDLs does not necessarily parallels the logical UML package hierarchy.

6.3 Colours

For clarity, class diagrams follows a certain colour scheme. Blue for application interface packages and yellow for all the others.

6.4 Naming scheme

The following naming scheme is used for both documentation and IDLs.

packages

lowercase.

Using the domain-based naming (For example, org.threegpp.osa)

classes, structures and types. Start with T

TpCapitalizedWithInternalWordsAlsoCapitalized

Exception class:

TpClassNameEndsWithException

Interface. Start with Ip:

IpThisIsAnInterface

constants:

P_UPPER_CASE_WITH_UNDERSCORES_AND_START_WITH_P

methods:

firstWordLowerCaseButInternalWordsCapitalized()

method's parameters

firstWordLowerCaseButInternalWordsCapitalized

collections (set, array or list types)

TpCollectionEndsWithSet

class/structure members

FirstWordAndInternalWordsCapitalized

Spaces in between words are not allowed.

6.5 State Transition Diagram text and text symbols

The descriptions of the State Transitions in the State Transition Diagrams follow the convention:

when_this_event_is_received [guard condition is true] /do_this_action ^send_this_message

Furthermore, text underneath a line through the middle of a State indicates an exit or entry event (normally specified which one).

6.6 Error results

As OMG IDL supports exception handling with high efficiency, OSA methods communicate errors in the form of CORBA exceptions of type TpGeneralException in the IDLs; the CORBA methods themselves always return void. But in the documentation, errors are communicated using a return parameter of type TpGeneralResult.

6.7 References

In the interface specification whenever parameters are to be passed by reference, the "Ref" suffix is appended to their corresponding data type (e.g. IpAnInterfaceRef anInterface), a reference can also be viewed as a logical indirection. Therefore, structured or primitive data type passed as *out* parameters are references. An interface passed as an *in* parameter is also a reference but an interface passed as an *out* parameter is a double indirection (i.e.: RefRef)

Original Data type	IN parameter declaration	OUT parameter declaration
TpPrimitive	parm : IN TpPrimitive	parm : OUT TpPrimitiveRef
TpStructured	parm : IN TpStructured	parm : OUT TpStructuredRef
IpInterface	parm : IN IpInterfaceRef	parm : OUT IpInterfaceRefRef

In IDL, however, the following rules apply:

- Interfaces are implicitly passed by reference.
- *out* parameters are also implicitly passed by reference.

This leads to:

- Interface as an *in* parameter: Passed by Reference.

- Structure or primitive type as an *in* parameter: Passed by Value.
- Structure or primitive type as an *out* parameter: Passed by Reference.
- Interface as an *out* parameter: As reference passed by reference.

To simplify the documentation without adding ambiguities, parameters (interfaces, structures and primitive data types) are used as is when specified as *in* or *out* parameters in the IDL. This means that there will be no "Ref" added after the data types of parameters in the IDL.

6.8 Number of out parameters

In order to support mapping to as many languages as possible, there is only 1 out parameter allowed per operation.

6.9 Strings and Collections

For character strings, the *String* data type is used without regard to the maximum length of the string. In IDL, the data type *String* is typedefed[1] from the CORBA primitive *string*. This CORBA primitive is made up of a length and a variable array of byte.

For homogeneous collections of instances of a particular data type the following naming scheme is used: <datatype>Set. In OMG IDL, this maps to a sequence of the data type. A CORBA sequence is implicitly made of a length and a variable array of elements of the same type.

***Example*:** typedef sequence<TpSessionID> TpSessionIDSet;

Collection types can be implemented (for example, in C++) as a structure containing an integer for the *number* part, and an array for the *data* part.

Example: The TpAddressSet data type may be defined in C++ as:

```
typedef struct {
    short       number;
    TpAddress   address [];
} TpAddressSet;
```

The array "address" is allocated dynamically with the exact number of required TpAddress elements based on "number".

6.10 Prefixes

OSA constants and data types are not defined in the global name space but in the *org.threegpp.osa* module.

6.11 Naming space across CORBA modules

The following shows the naming space used in this specification.

```
module org {
    module open_service_access {
            /* The fully qualified name of the following constant
             is org::open_service_access::P_THIS_IS_AN_OSA_GLOBAL_CONST */
            const long P_THIS_IS_AN_OSA_GLOBAL_CONST= 1999;
```

[1] A *typedef* is a type definition declaration in IDL.

```
                // Add other OSA global constants and types here
                module fw {
                  /* no scoping required to access  P_THIS_IS_AN_OSA_GLOBAL_CONST */
                     const long P_FW_CONST= THIS_IS_AN_OSA_GLOBAL_CONST;
                };
                module mm {
                    // scoping required to access P_FW_CONST
                     const long P_M_CONST= fw::P_FW_CONST;
                };
            };
};
```

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for approval and placement under Change Control

3GPP TS 29.198-2 V1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface
Part 2: Common data
(Release 4)

Keywords
OSA, API, Interface Class, IDL

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1. Scope

This document is part of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA). The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA API's. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

This document specifies the Common Data Definitions of the OSA. The Common Data definitions contain data-types that are common acros the rest of the OSA API. All aspects of the Common data are defined here, these being:

- Data definitions
- IDL Description of the interfaces

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2. References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

[1] 3GPP TS 29.198-1 "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in TS 29.198-1 [1] apply.

3.2 Symbols

For the purposes of the present document, the symbols in TS 29.198-1 [1] apply.

3.3 Abbreviations

For the purposes of the present document, the abbreviations in TS 29.198-1 [1] apply.

4 Common Data definitions

The following sections describe each aspect of the Common data definitions.

The order is as follows:

- The Data definitions section shows a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.

5 Common System Data Definitions

These data definitions are assumed to be provided by the client operating system.

5.1 Standard Data Types

The APIs assume that the following data types can be supported.

5.1.1 TpBoolean

Defines a Boolean data type.

5.1.2 TpInt32

Defines a signed 32 bit integer.

5.1.3 TpInt32Ref

Defines a 5.2.4 Reference to a TpInt32.

5.1.4 TpFloat

Defines a single precision real number

5.1.5 TpFloatRef

Defines a Reference to a TpFloat

5.1.6 TpLongstring

Defines a Byte string, comprising length and data. The length must be at least a 32 bit integer.

5.1.7 TpLongstringRef

Defines a 5.2.4 Reference to a TpLongstring.

5.1.8 TpString

Defines a Byte string, comprising length and data. The length must be at least a 16 bit integer.

5.1.9 TpStringRef

Defines a 5.2.4 Reference to a TpString.

5.1.10 TpAssignmentID

This data type is identical to a TpInt32. It specifies a number which identifies an individual event notification enabled by the application or service.

5.1.11 TpAssignmentIDRef

Defines a Reference to type TpAssignmentID.

5.1.12 TpSessionID

Defines a network unique session ID. The API uses this ID to identify sessions, e.g. call or call leg sessions, within an object implementing an interface capable of handling multiple sessions. For the different services, the sessionIDs are unique only in the context of a service manager instantiation (e.g., within the context of one generic call control manager). As such if an application creates two instances of the same service manager it shall use different instantiations of the callback objects which implement the callback interfaces.

The session ID is identical to a TpInt32 type.

5.1.13 TpSessionIDRef

Defines a Reference to a TpSessionID.

5.1.14 TpSessionIDSet

Defines a Numbered_Set_of_Data_Elements of TpSessionID.

5.2 Other Data Sorts

The APIs assumes that the following data syntaxes can be supported

5.2.1 Sequence of Data Elements

This describes a sequence of data types. This may be defined as a structure (for example, in C++) or simply a sequence of data elements within a structure.

Example

The TpAddress data type may be defined in C++ as:

```
typedef struct {
  TpAddressPlan       Plan;
  TpString         AddrString;
  TpString         Name;
  TpAddressPresentation.....Presentation;
....TpAddressScreening........Screening;
....TpString..................SubAddressString;
} TpAddress;
```

5.2.2 Tagged Choice of Data Elements

This describes a data type which actually evaluates to one of a choice of a number of data elements. This data element contains two parts: a tag data type (the *tag* part) which is used to identify the chosen data type, and the chosen data type itself (the *union* part). This form of data type is also referred to as a tagged union.

This data type can be implemented (for example, in C++) as a structure containing an integer for the *tag* part, and a union for the *union* part.

This data type is implementation specific. Please refer to the appropriate IDL documents (and the resulting language mappings) to see how this data type is implemented.

Example

The `TpCallError` data type may be defined in C++ as:

```
typedef struct {
  TpCallErrorType Tag;
  union {
    TpCallErrorInfoUndefined     Undefined;
    TpCallErrorInfoRoutingAborted  RoutingAborted;
    TpCallErrorInfoCallAbandoned   CallAbandoned;
    TpCallErrorInfoInvalidAddress  InvalidAddress;
    TpCallErrorInfoInvalidState   InvalidState;
    TpCallErrorInfoInvalidCriteria  InvalidCriteria;
  } callErrorInfo;
} TpCallError;
```

5.2.3 Numbered Set of Data Elements

This describes a data type which comprises an integer which indicates the total number of data elements in the set (the *number* part), and an **unordered** set of data elements (the *data* part). *Set* data types do not contain duplicate data elements.

Example

The TpAddressSet data type may be defined in MIDL as:

```
typedef struct TpAddressSet
{
TpInt32 Number; [size_is(Number)] TpAddress Set[];
}
TpAddressSet;
```

5.2.4 Reference

This describes a reference (or pointer) to a data type. This is primarily used to describe 'out' method parameters.

This data type may be implemented (for example, in C++) as a pointer. However, in some languages it may not be necessary for 'out' parameters to be implemented as pointers.

Example

The TpAddressRef data type may be defined in C++ as:

```
typedef TpAddress *   TpAddressRef
```

5.3 Interface Related Data Definitions

5.3.1 IpInterface

Defines the address of a generic interface instance.

5.3.2 IpInterfaceRef

Defines a 5.2.4 Reference to type IpInterface.

5.3.4 IpInterfaceRefRef

Defines a 5.2.4 Reference to type IpInterfaceRef.

5.4 Method Result Data Definitions

5.4.1 TpResult

Defines the 5.2.1 Sequence of Data Elements that specify the result of a method call. All methods in the APIs return a result of type TpResult.

Sequence Element Name	Sequence Element Type
ResultType	TpResultType
ResultFacility	TpResultFacility
ResultInfo	TpResultInfo

5.4.2 TpResultType

Defines whether the method was successful or not.

Name	Value	Description
P_RESULT_FAILURE	0	Method failed
P_RESULT_SUCCESS	1	Method was successful

5.4.3 TpResultFacility

Defines the facility code of a result. In phase 2 of the APIs, only P_RESULT_FACILITY_UNDEFINED must be used.

Name	Value	Description

P_RESULT_FACILITY_UNDEFINED	0	Undefined

5.4.4 TpResultInfo

Defines further information relating to the result of the method, such as error codes.

Name	Value	Description
P_RESULT_INFO_UNDEFINED	0000h	No further information present
P_INVALID_DOMAIN_ID	0001h	Invalid client ID
P_INVALID_AUTH_CAPABILITY	0002h	Invalid authentication capability
P_INVALID_AGREEMENT_TEXT	0003h	Invalid agreement text
P_INVALID_SIGNING_ALGORITHM	0004h	Invalid signing algorithm
P_INVALID_INTERFACE_NAME	0005h	Invalid interface name
P_INVALID_SERVICE_ID	0006h	Invalid service ID
P_INVALID_EVENT_TYPE	0007h	Invalid event type
P_SERVICE_NOT_ENABLED	0008h	The service ID does not correspond to a service that has been enabled
P_INVALID_ASSIGNMENT_ID	0009h	The assignment ID is invalid
P_INVALID_PARAMETER	000Ah	The method has been called with an invalid parameter
P_INVALID_PARAMETER_VALUE	000Bh	A method parameter has an invalid value
P_PARAMETER_MISSING	000Ch	A mandatory parameter has not been specified in the method call
P_RESOURCES_UNAVAILABLE	000Dh	The required resources in the network are not available
P_TASK_REFUSED	000Eh	The requested method has been refused
P_TASK_CANCELLED	000Fh	The requested method has been cancelled
P_INVALID_DATE_TIME_FORMAT	0010h	Invalid date and time format provided
P_NO_CALLBACK_ADDRESS_SET	0011h	The requested method is refused because no callback address is set
P_INVALID_SIGNATURE	0012h	Invalid digital signature
P_INVALID_SERVICE_TOKEN	0013h	The service token has not been issued, or it has expired.
P_ACCESS_DENIED	0014h	The client is not currently authenticated with the framework
P_INVALID_PROPERTY	0015h	The framework does not recognise the property supplied by the client
P_METHOD_NOT_SUPPORTED	0016h	The method is not allowed or supported within the context of the current service agreement.
P_NO_ACCEPTABLE_AUTH_CAPABILITY	0017h	An authentication mechanism, which is acceptable to the framework, is not supported by the client
P_INVALID_INTERFACE_TYPE	0018h	The interface reference supplied by the client is the wrong type.
P_INVALID_ACCESS_TYPE	0019h	The framework does not support the type of access interface requested by the client.
P_SERVICE_ACCESS_DENIED	001Ah	The client application is not allowed to access this service.
P_USER_NOT_SUBSCRIBED	0030h	An application is unauthorised to access information and request services with regards to users that are not subscribed to the application.
P_APPLICATION_NOT_ACTIVATED	0031h	An application is unauthorised to access information and request services with regards to users that have deactivated that particular application.
P_USER_PRIVACY	0032h	An application is unauthorised to access information and request services with regards to users that have set their privacy flag regarding that particular service.

Name	Value	Description
P_GCCS_SERVICE_INFORMATION_MISSING	0100h	Information relating to the Call Control service could not be found
P_GCCS_SERVICE_FAULT_ENCOUNTERED	0101h	Fault detected in the Call Control service
P_GCCS_UNEXPECTED_SEQUENCE	0102h	Unexpected sequence of methods, i.e., the sequence does not match the specified state diagrams for the call or the call leg.

Name	Value	Description
P_GCCS_INVALID_ADDDRESS	0103h	Invalid address specified
P_GCCS_INVALID_CRITERIA	0104h	Invalid criteria specified
P_GCCS_INVALID_NETWORK_STATE	0105h	Although the sequence of method calls is allowed by the gateway, the underlying protocol can not support it. E.g., in some protocols some methods are only allowed by the protocol, when the call processing is suspended, e.g., after reporting an event that was monitored in interrupt mode.

Name	Value	Description
P_GMS_INVALID_MAILBOX	0200h	Invalid mailbox number
P_GMS_INVALID_AUTHENTICATION_INFO	0201h	Invalid authentication information
P_GMS_INVALID_SESSION_ID	0202h	Invalid session ID
P_GMS_LOCKING_LOCKED_MAILBOX	0203h	Application attempts to lock a mailbox that has already been locked
P_GMS_UNLOCKING_UNLOCKED_MAILBOX	0204h	The session ID does not correspond to a locked mailbox
P_GMS_INVALID_MESSAGE_FORMAT	0205h	Invalid message format
P_GMS_HEADER_NUMBER_TOO_LARGE	0206h	The number is too large for the service to handle
P_GMS_INSUFFICIENT_HEADERS	0207h	Mandatory headers are not included
P_GMS_MESSAGE_NOT_REMOVED	0208h	The message cannot be removed
P_GMS_INSUFFICIENT_PRIVILEGE	0209h	The application does not have sufficient privilege to remove the message
P_GMS_INVALID_FOLDER_ID	020Ah	The identity of the folder is not valid
P_GMS_FOLDER_DOES_NOT_EXIST	020Bh	The folder does not exist
P_GMS_NUMBER_NOT_POSITIVE	020Ch	The number given is not positive
P_GMS_INVALID_MESSAGE_ID	020Dh	Message ID is not valid
P_GMS_CHANGING_READONLY_PROPERTY	020Eh	The change has not been carried out because some of the properties cannot be modified.
P_GMS_HEADER_DOES_NOT_EXIST	020Fh	Some of the headers do not exist
P_GMS_MAILBOX_LOCKED	0210h	Attempting to update a locked mailbox
P_GMS_CANNOT_UNLOCK_MAILBOX	0211h	Attempting to unlock a mailbox which is locked by another application
P_GMS_PROPERTY_NOT_SET	0212h	Failed attempt to set a property
P_GMS_FOLDER_IS_OPEN	0213h	Failed attempt to open the same folder more than once
P_GMS_MAILBOX_OPEN	0214h	Failed attempt to remove an open mailbox

Name	Value	Description
P_GUIS_INVALID_CRITERIA	0300h	Invalid criteria specified
P_GUIS_ILLEGAL_ID	0301h	Information id specified is invalid
P_GUIS_ID_NOT_FOUND	0302h	A legal information id is not known to the User Interaction Service
P_GUIS_ILLEGAL_RANGE	0303h	The values for minimum and maximum collection length are out of range.
P_GUIS_INVALID_COLLECTION_CRITERIA	0304h	Invalid collection criteria specified
P_GUIS_INVALID_NETWORK_STATE	0305h	Although the sequence of method calls is allowed by the gateway, the underlying protocol can not support it. E.g., in some protocols some methods are only allowed by the protocol, when the call processing is suspended, e.g., after reporting an event that was monitored in interrupt mode.
P_GUIS_UNEXPECTED_SEQUENCE	0306h	Unexpected sequence of methods, i.e., the sequence does not match the specified state diagrams.

5.5 Date and Time Related Data Definitions

5.5.1 TpDate

This data type is identical to a TpString. It specifies the data in accordance with International Standard ISO 8601. This is defined as the string of characters in the following format:

YYYY-MM-DD

where the date is specified as:

YYYY	four digits year
MM	two digits month
DD	two digits day

The date elements are separated by a hyphen character (-).

EXAMPLE 1: The 4 December 1998, is encoded as the string:

`1998-12-04`

5.5.2 TpTime

This data type is identical to a TpString. It specifies the time in accordance with International Standard ISO 8601. This is defined as the string of characters in the following format:

HH:MM:SS.mmm

or

HH:MM:SS.mmmZ

where the time is specified as:

HH	two digits hours (24h notation)
MM	two digits minutes
SS	two digits seconds
mmm	three digits fractions of a second (i.e. milliseconds)

The time elements are separated by a colon character (:).The date and time are separated by a space. Optionally, a capital letter Z may be appended to the time field to indicate Universal Time (UTC). Otherwise, local time is assumed.

EXAMPLE 2: 10:30 and 15 seconds is encoded as the string:

`10:30:15.000`

for local time, or in UTC it would be: `10:30:15.000Z`

5.5.3 TpDateAndTime

This data type is identical to a TpString. It specifies the data and time in accordance with International Standard ISO 8601. This is defined as the string of characters in the following format:

YYYY-MM-DD HH:MM:SS.mmm

or

YYYY-MM-DD HH:MM:SS.mmmZ

where the date is specified as:

YYYY	four digits year
MM	two digits month
DD	two digits day

The date elements are separated by a hyphen character (-).

The time is specified as:

HH	two digits hours (24h notation)
MM	two digits minutes
SS	two digits seconds
mmm	three digits fractions of a second (i.e. milliseconds)

The time elements are separated by a colon character (:).The date and time are separated by a space. Optionally, a capital letter Z may be appended to the time field to indicate Universal Time (UTC). Otherwise, local time is assumed.

EXAMPLE 3: The 4 December 1998, at 10:30 and 15 seconds is encoded as the string:

```
1998-12-04 10:30:15.000
```

for local time, or in UTC it would be:

```
1998-12-04 10:30:15.000Z
```

5.5.4 TpDateAndTimeRef

Defines a 5.2.4 Reference to type TpDateAndTime.

5.5.5 TpDuration

This data type is a TpInt32 representing a time interval in milliseconds. A value of "-1" defines infinite duration and a value of "-2" represents a default duration.

5.6 Address Related Data Definitions

5.6.1 TpAddress

Defines the 5.2.1 Sequence of Data Elements that specify an address.

Sequence Element Name	Sequence Element Type
Plan	TpAddressPlan
AddrString	TpString
Name	TpString
Presentation	TpAddressPresentation
Screening	TpAddressScreening
SubAddressString	TpString

The AddrString defines the actual address information and the structure of the string depends on the Plan. The following table gives an overview of the format of the AddrString for the different address plans.

Address Plan	AddrString Format Description	Example
P_ADDRESS_PLAN_NOT_PRESENT	Not applicable	

P_ADDRESS_PLAN_UNDEFINED	Not applicable	
P_ADDRESS_PLAN_IP	For Ipv4 the dotted quad notation is used. Also for IPv6 the dotted notation is used. The address can optionally be followed by a port number separated by a colon.	"127.0.0.1:42"
P_ADDRESS_PLAN_MULTICAST	An Ipv4 class D address or Ipv6 equivalent in dotted notation.	"224.0.0.0"
P_ADDRESS_PLAN_UNICAST	A non multicast or broadcast IP address in dotted notation.	"127.0.0.1"
P_ADDRESS_PLAN_E164	An international number without the international access code, including the country code and excluding the leading zero of the area code.	"31161249111"
P_ADDRESS_PLAN_AESA	The ATM End System Address in binary format (40 bytes)	01234567890ABCDEF01234567890ABCDEF01234567
P_ADDRESS_PLAN_URL	A uniform resource locator as defined in IETF RFC 1738	"http://www.parlay.org"
P_ADDRESS_PLAN_NSAP	The binary representation of the Network Service Access Point	490001AA000400010420
P_ADDRESS_PLAN_SMTP	An e-mail address as specified in IETF RFC822	"webmaster@parlay.org"
P_ADDRESS_PLAN_MSMAIL	Identical to P_ADDRESS_PLAN_SMTP	"john.doe@hitech.com"
P_ADDRESS_PLAN_X400	The X400 address structured as a set of attibute value pairs separated by semicolons.	"C=nl;ADMD= ;PRMD=uninet;O=parlay;S=Doe;I=S;G =John'

5.6.2 TpAddressRef

Defines a 5.2.4 Reference to type TpAddress.

5.6.3 TpAddressSet

Defines a Numbered_Set_of_Data_Elements of TpAddress.

5.6.4 TpAddressSetRef

Defines a 5.2.4 Reference to type TpAddressSet.

5.6.5 TpAddressPresentation

Defines whether an address can be presented to an end user.

Name	Value	Description
P_ADDRESS_PRESENTATION_UNDEFINED	0	Undefined
P_ADDRESS_PRESENTATION_ALLOWED	1	Presentation Allowed
P_ADDRESS_PRESENTATION_RESTRICTED	2	Presentation Restricted
P_ADDRESS_PRESENTATION_ADDRESS_NOT_AVAILABLE	3	Address not available for presentation

5.6.6 TpAddressScreening

Defines whether an address can be presented to an end user.

Name	Value	Description
P_ADDRESS_SCREENING_UNDEFINED	0	Undefined
P_ADDRESS_SCREENING_USER_VERIFIED_PASSED	1	user provided address verified and passed
P_ADDRESS_SCREENING_USER_NOT_VERIFIED	2	user provided address not verified
P_ADDRESS_SCREENING_USER_VERIFIED_FAILED	3	user provided address verified and failed
P_ADDRESS_SCREENING_NETWORK	4	Network provided address (Note that even though the application may provide the address to the gateway, from the end-user point of view it is still regarded as a network provided address)

5.6.7 TpAddressPlan

Defines the address plan (or numbering plan) used. It is also used to indicate whether an address is actually defined in a TpAddress data element.

Name	Value	Description
P_ADDRESS_PLAN_NOT_PRESENT	-1	No Address Present
P_ADDRESS_PLAN_UNDEFINED	0	Undefined
P_ADDRESS_PLAN_IP	1	IP
P_ADDRESS_PLAN_MULTICAST	2	Multicast
P_ADDRESS_PLAN_UNICAST	3	Unicast
P_ADDRESS_PLAN_E164	4	E.164
P_ADDRESS_PLAN_AESA	5	AESA
P_ADDRESS_PLAN_URL	6	URL
P_ADDRESS_PLAN_NSAP	7	NSAP
P_ADDRESS_PLAN_SMTP	8	SMTP
P_ADDRESS_PLAN_MSMAIL[1]	9	Microsoft Mail
P_ADDRESS_PLAN_X400	10	X.400

For the case where the P_ADDRESS_PLAN_NOT_PRESENT is indicated, the rest of the information in the TpAddress is not valid.

5.6.8 TpAddressError

Defines the reasons why an address is invalid.

Name	Value	Description
P_ADDRESS_INVALID_UNDEFINED	0	Undefined error
P_ADDRESS_INVALID_MISSING	1	Mandatory address not present
P_ADDRESS_INVALID_MISSING_ELEMENT	2	Mandatory address element not present
P_ADDRESS_INVALID_OUT_OF_RANGE	3	Address is outside of the valid range
P_ADDRESS_INVALID_INCOMPLETE	4	Address is incomplete

[1] This value is not used in the scope of 3GPP

Name	Value	Description
P_ADDRESS_INVALID_CANNOT_DECODE	5	Address cannot be decoded

5.6.9 TpAddressRange

This type is identical to TpAddress with the difference that the AddrString can contain wildcards.

Two wildcards are allowed: * which matches zero or more characters and ? which matches exactly one character. The wildcards are only allowed at the end or at the beginning of the AddrString.

Some examples for E164 addresses:

- "123" matches specifies number;
- "123*" matches all numbers starting with 123 (including 123 itself);
- "123??*" matches all numbers starting with 123 and at least 5 digits long;
- "123???" matches all numbers starting with 123 and exactly 6 digits long;

For e-mail style addresses, the wildcards are allowed at the beginning of the AddrString:

- "*@parlay.org" matches all email addresses in the parlay.org domain.

The following address ranges are illegal:

- 1?3
- 1*3
- ?123*

Legal occurrences of the '*' and '?' characters in AddrString should be escaped by a '\' character. To specify a '\' character '\\' must be used.

5.6.10 TpURL

This data type is identical to a TpString and contains a URL address. The usage of this type is distinct from TpAddress, which can also hold a URL. The latter contains a user address which can be specified in many ways: IP, e-mail, URL etc. On the other hand, the TpURL type does not hold the address of a user and always represents a URL. This type is used in user interaction and defines the URL of the test or stream to be sent to an end-user. It is therefore inappropriate to use a general address here.

5.7 Price-related Data Definitions

5.7.1 TpPrice

This data type is identical to a TpString. It specifies price information. This is defined as a string of characters (digits) in the following format:

DDDDDD.DD

5.7.2 TpAoCInfo

Defines the Sequence of Data Elements that specify the Advice Of Charge information to be sent to the terminal.

Sequence Element Name	Sequence Element Type	Description

ChargeOrder	TpAoCOrder	Charge order
Currency	TpString	Currency unit according to ISO-4217:1995

5.7.3 TpAoCOrder

Defines the Tagged Choice of Data Elements that specify the charge plan for the call.

	Tag Element Type	
	TpAoCOrderCategory	

Tag Element Value	Choice Element Type	Choice Element Name
P_CHARGE_ADVICE_INFO	TpChargeAdviceInfo	ChargeAdviceInfo
P_CHARGE_PER_TIME	TpChargePerTime	ChargePerTime
P_CHARGE_NETWORK	TpString	NetworkCharge

5.7.4 TpCallAoCOrderCategory

Name	Value	Description
P_CHARGE_ADVICE_INFO	0	Set of GSM Charge Advice Information elements according to 3GPP TS 22.024
P_CHARGE_PER_TIME	1	Charge per time
P_CHARGE_NETWORK	2	Operator specific charge plan specification, e.g. charging table name / charging table entry

5.7.5 TpChargeAdviceInfo

Defines the Sequence of Data Elements that specify the two sets of Advice of Charge parameters. The first set defines the current tariff. The second set may be used in case of a tariff switch in the network.

Sequence Element Name	Sequence Element Type	Description
CurrentCAI	TpCAIElements	Current tariff
NextCAI	TpCAIElements	Next tariff after tariff switch

5.7.6 TpCAIElements

Defines the Sequence of Data Elements that specify theCharging Advice Information elements according to 3GPP TS 22.024.

Sequence Element Name	Sequence Element Type	Description
UnitsPerInterval	TpInt32	Units per interval
SecondsPerTimeInterval	TpInt32	Seconds per time interval
ScalingFactor	TpInt32	Scaling factor
UnitIncrement	TpInt32	Unit increment
UnitsPerDataInterval	TpInt32	Units per data interval
SegmentsPerDataInteral	TpInt32	Segments per data interal
InitialSecsPerTimeInterval	TpInt32	Initial secs per time interval

5.7.7 TpChargePerTime

Defines the Sequence of Data Elements that specify the time based charging information.

Sequence Element Name	Sequence Element Type	Description
InitialCharge	TpInt32	Initial charge amount (in currency units * 0.0001)
CurrentChargePerMinute	TpInt32	Current tariff (in currency units * 0.0001)
NextChargePerMinute	TpInt32	Next tariff (in currency units * 0.0001) after tariff switch Only used in setAdviceOfCharge()

5.7.8 TpLanguage

This data type is identical to a TpString, and defines the language. In case an indication for the language is not needed an empty string must be used. In other cases valid language strings are defined in ISO 639.

Annex A (normative): OMG IDL Description of the Common Data definitions

The OMG IDL representation of this specification is contained in a text file (osa.idl contained in archive 2919802IDL.ZIP) which accompanies the present document.

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for approval and placement under Change Control

3GPP TS 29.198-3 V1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface
Part 3: Framework
(Release 4)

Keywords
API, OSA, IDL, FW, Framework

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

This document is part of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA). The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA API's. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

This document specifies the Framework aspects of the interface. All aspects of the Framework are defined here, these being:

- Sequence Diagrams
- Class Diagrams
- Interface specification plus detailed method descriptions
- State Transition diagrams
- Data definitions
- IDL Description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modelling Language (UML).

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

[1] 3GPP TS 29.198-1 "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

[4] IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in TS 29.198-1 [1] apply.

3.2 Symbols

For the purposes of the present document, the symbols in TS 29.198-1 [1] apply.

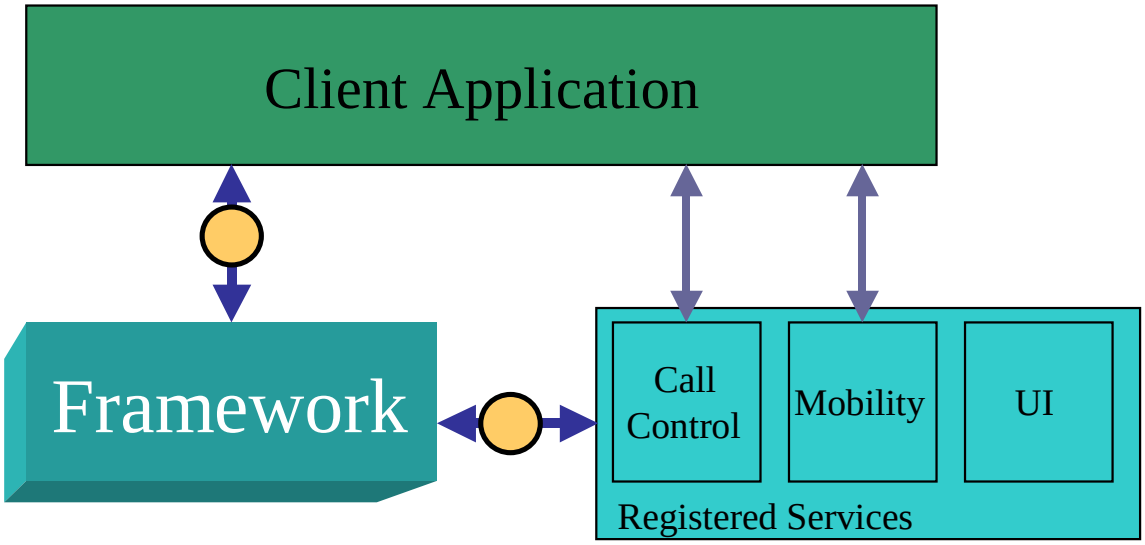
3.3 Abbreviations

For the purposes of the present document, the abbreviations in TS 29.198-1 [1] apply.

4 Overview of the Framework

This subclause explains which basic mechanisms are executed in the OSA Framework prior to offering and activating applications.

The Framework API contains interfaces between the Application Server and the Framework, and between Network Service Capability Server (SCS) and the Framework (these interfaces are represented by the yellow circles in the diagram below). The description of the Framework in this document separates the interfaces into these two distinct sets: Framework to Application interfaces and Framework to Service interfaces.



Some of the mechanisms are applied only once (e.g. establishment of service agreement), others are applied each time a user subscription is made to an application (e.g. enabling the call attempt event for a new user).

Basic mechanisms between Application and Framework:

- **Authentication:** Once an off-line service agreement exists, the application can access the authentication interface. The authentication model of OSA is a peer-to-peer model. The application must authenticate the framework and vice versa. The application must be authenticated before it is allowed to use any other OSA interface.

- **Authorisation:** Authorisation is distinguished from authentication in that authorisation is the action of determining what a previously authenticated application is allowed to do. Authentication must precede authorisation. Once authenticated, an application is authorised to access certain service capability features.

- **Discovery of framework and network service capability features:** After successful authentication, applications can obtain available framework interfaces and use the discovery interface to obtain information on authorised network service capability features. The Discovery interface can be used at any time after successful authentication.

- **Establishment of service agreement:** Before any application can interact with a network service capability feature, a service agreement must be established. A service agreement may consist of an off-line (e.g. by physically exchanging documents) and an on-line part. The application has to sign the on-line part of the service agreement before it is allowed to access any network service capability feature.

- **Access to network service capability features:** The framework must provide access control functions to authorise the access to service capability features or service data for any API method from an application, with the specified security level, context, domain, etc.

Basic mechanism between Framework and Service Capability Server:

- **Registering of network service capability features**. SCFs offered by a Service Capability Server can be registered at the Framework. In this way the Framework can inform the Applications upon request about available service capability features (Discovery). For example, this mechanism is applied when installing or upgrading a Service Capability Server.

The following sections describe each aspect of the Framework in the following order:

- The *sequence diagrams* give the reader a practical idea of how each of the Framework is implemented.
- The *class diagrams* section show how each of the interfaces applicable to the Framework relate to one another.
- The *interface specification* section describes in detail each of the interfaces shown within the class diagram part.
- The *State Transition Diagrams (STD)* show the progression of internal processes, either in the application or in the gateway.
- The *data definitions* section show a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the common data types part of this specification.

5 The Base Interface Specification

5.1 Interface Specification Format

This section defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

5.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>

5.1.2 Method descriptions

Each method (API method “call”) is described. All methods in the API return a value of type `TpResult`, indicating, amongst other things, if the method invocation was sucessfully executed or not.

Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a '`Req`' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a '`Res`' or '`Err`' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name> or IpSvc<name>` interfaces to provide the callback mechanism.

5.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.

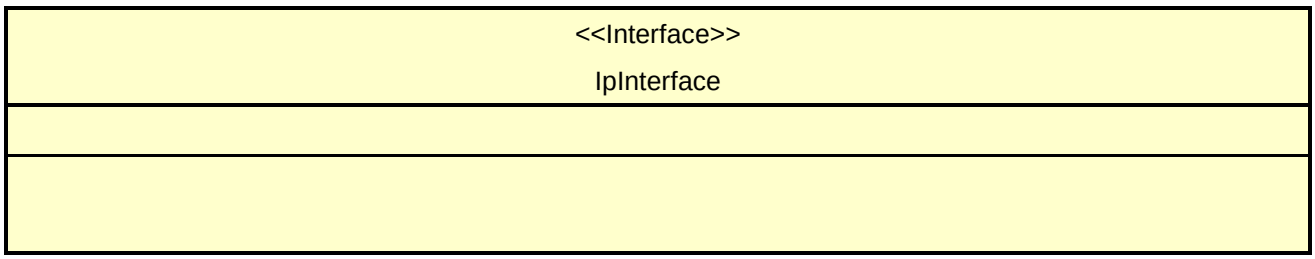
5.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

5.2 Base Interface

5.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.



5.3 Service Interfaces

5.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

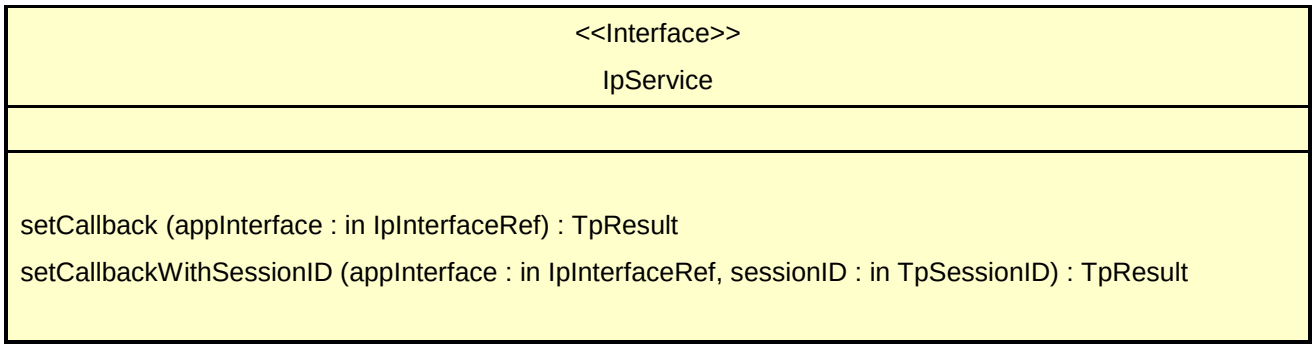
The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

5.4 Generic Service Interface

5.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.



Method

setCallback()

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

Raises

TpGeneralException

Method

setCallbackWithSessionID()

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

sessionID : in TpSessionID

Specifies the session for which the service can invoke the application's callback interface.

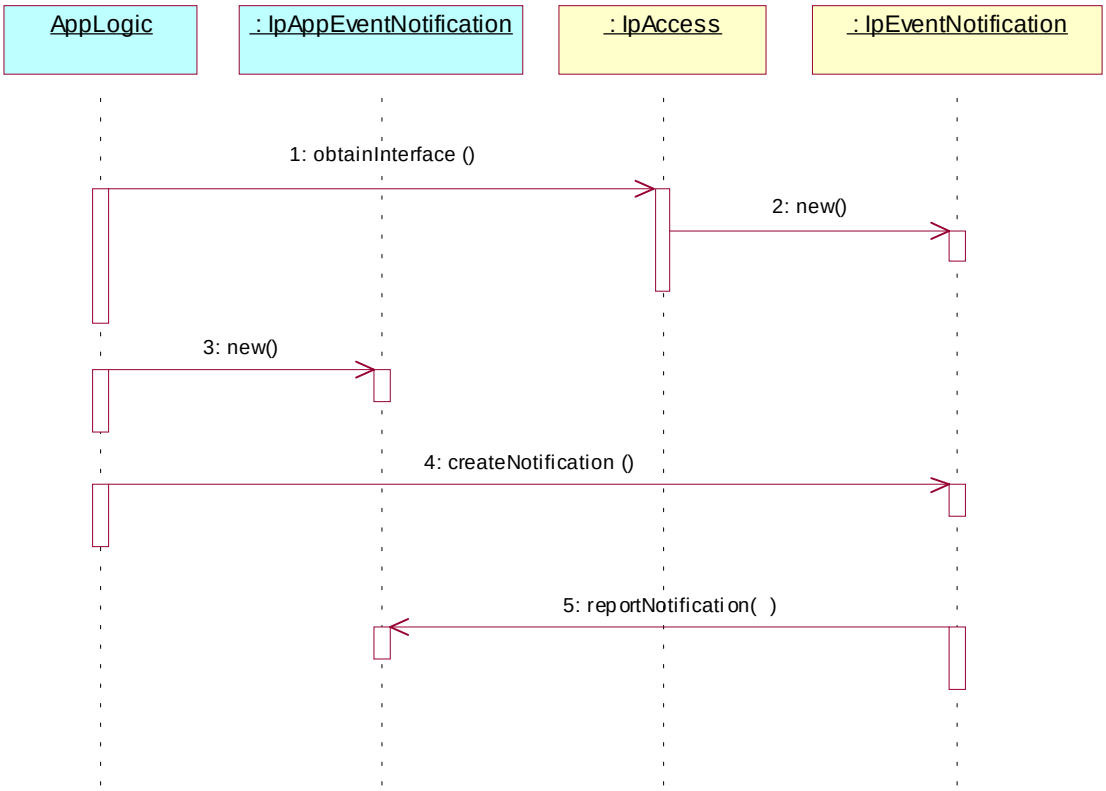
Raises

TpGeneralException

6 Framework-to-Application Sequence Diagrams

6.1 Event Notification Sequence Diagrams

6.1.1 Enable Event Notification



1: This message is used to receive a reference to the object implementing the IpEventNotification interface.

2: If there is currently no object implementing the IpEventNotification interface, then one is created using this message.

3: This message is used to create an object implementing the IpAppEventNotification interface.

4: enableNotification(eventCriteria : in TpFwEventCriteria, assignmentID : out TpAssignmentIDRef) : TpResult

This message is used to enable the notification mechanism so that subsequent framework events can be sent to the application. The framework event the application requests to be informed of is the availability of new SCFs.

Newly installed SCFs become available after the invocation of registerService and announceServiceAvailability on the Framework. The application uses the input parameter eventCriteria to specify the SCFs of whose availability it wants to be notified: those specified in ServiceTypeNameList.

The result of this invocation has many similarities with the result of invoking listServiceTypes: in both cases the application is informed of the availability of a list of SCFs. The differences are:

· in the case of invoking listServiceTypes, the application has to take the initiative, but it is informed of ALL SCFs available

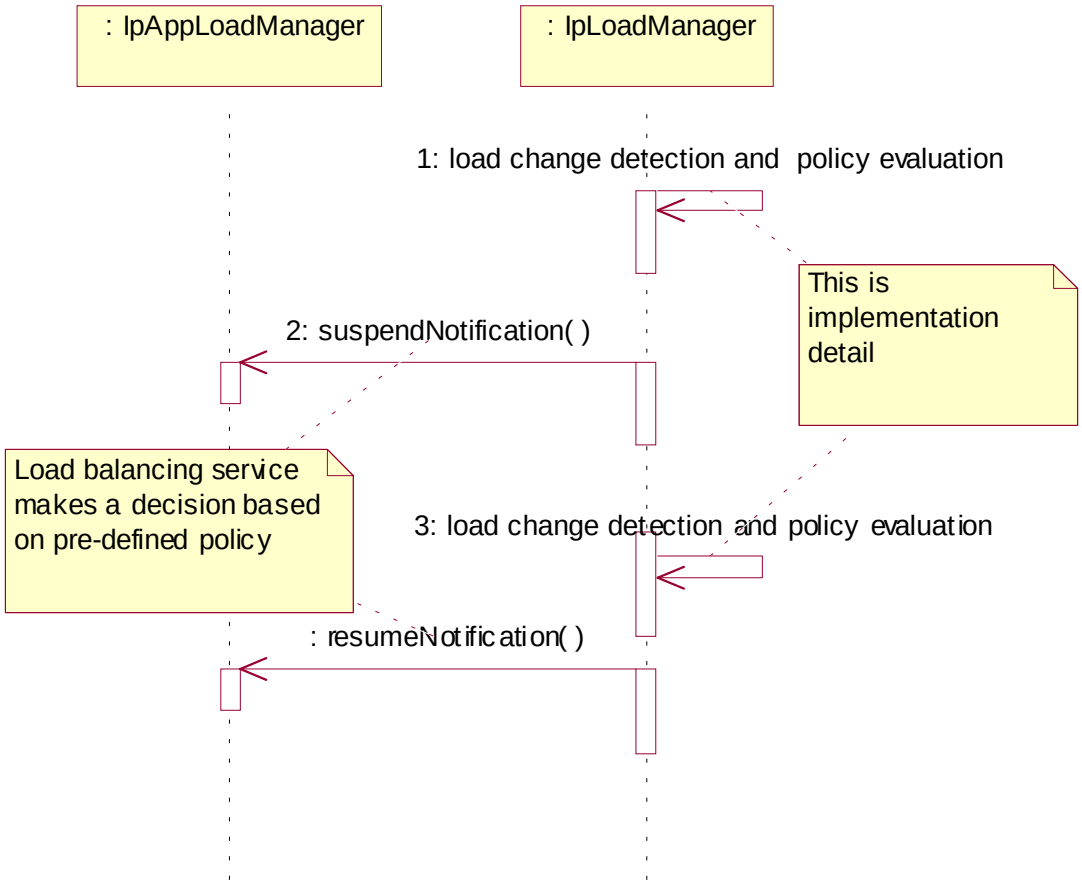
· in the case of using the event notification mechanism, the application needs not take the initiative to ask about the availability of SCFs, but it is only informed of the ones that are newly available.

5: The application is notified of the availability of new SCFs of the requested type(s).

6.2 Integrity Management Sequence Diagrams

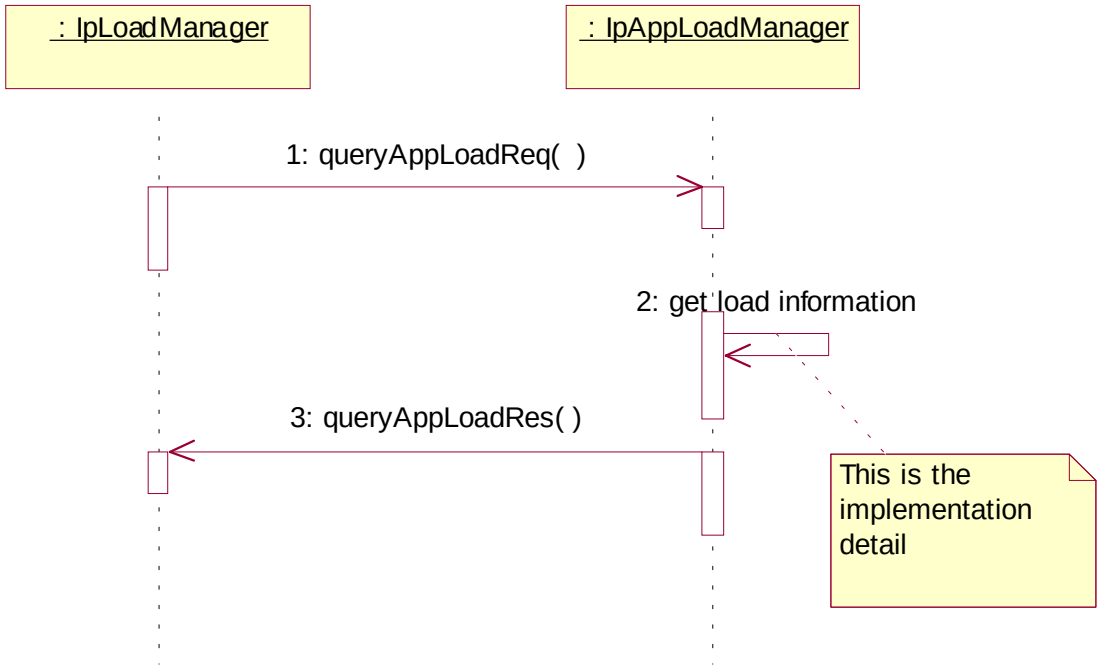
6.2.1 Load Management: Suspend/resume notification from application

This sequence diagram shows the scenario of suspending or resuming notifications from the application based on the evaluation of the load balancing policy as a result of the detection of a change in load level of the framework.



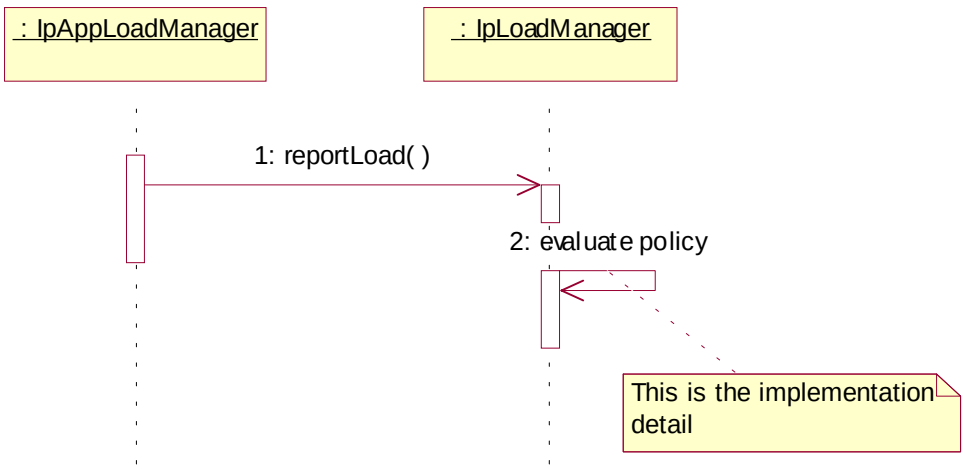
6.2.2 Load Management: Framework queries load status

This sequence diagram shows how the framework requests load statistics for an application.



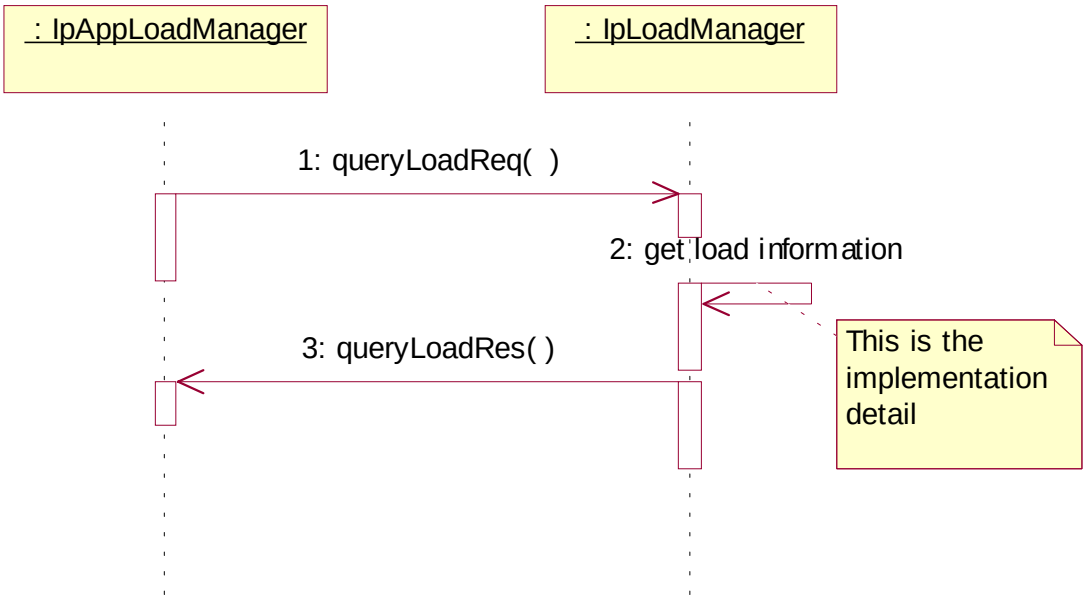
6.2.3 Load Management: Application reports current load condition

This sequence diagram shows how an application reports its load condition to the framework load manager.



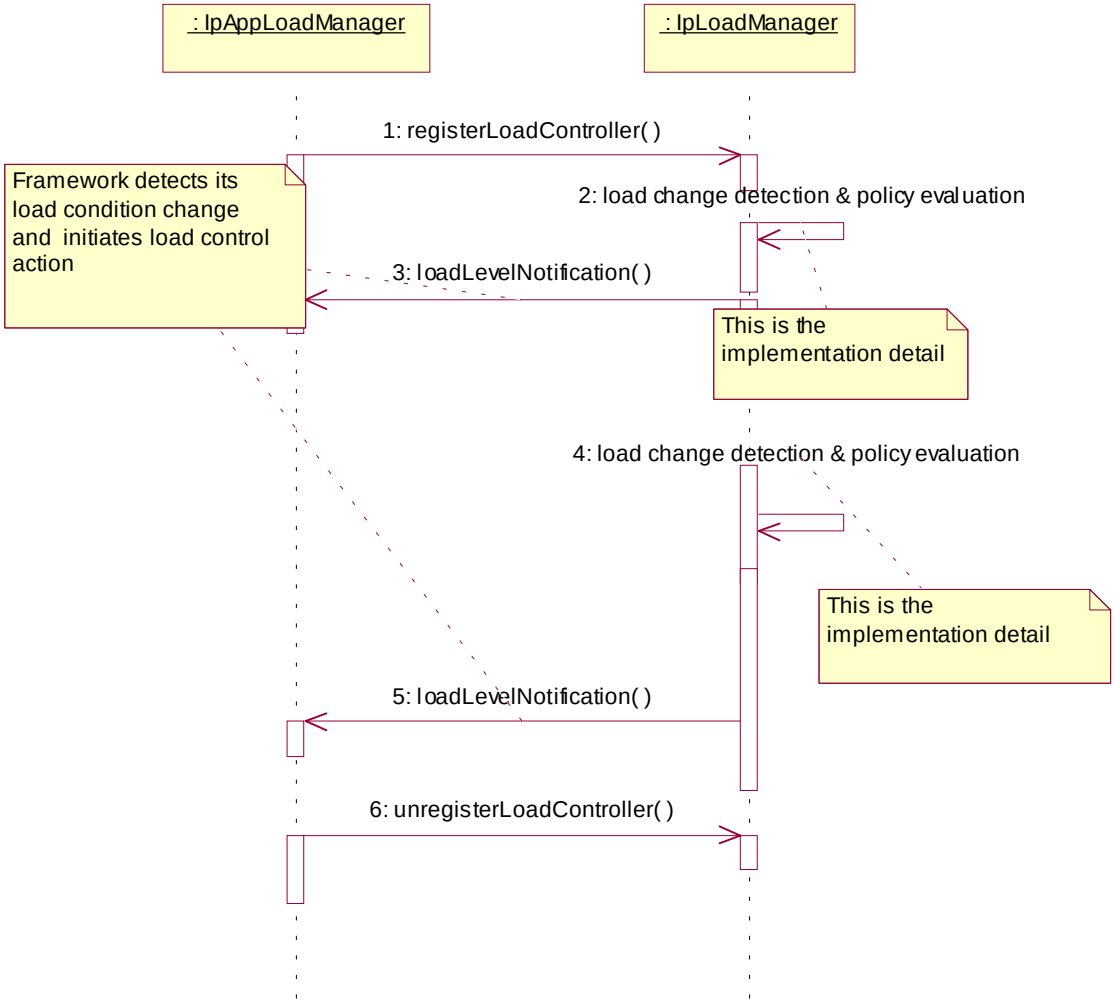
6.2.4 Load Management: Application queries load status

This sequence diagram shows how an application requests load statistics for the framework.

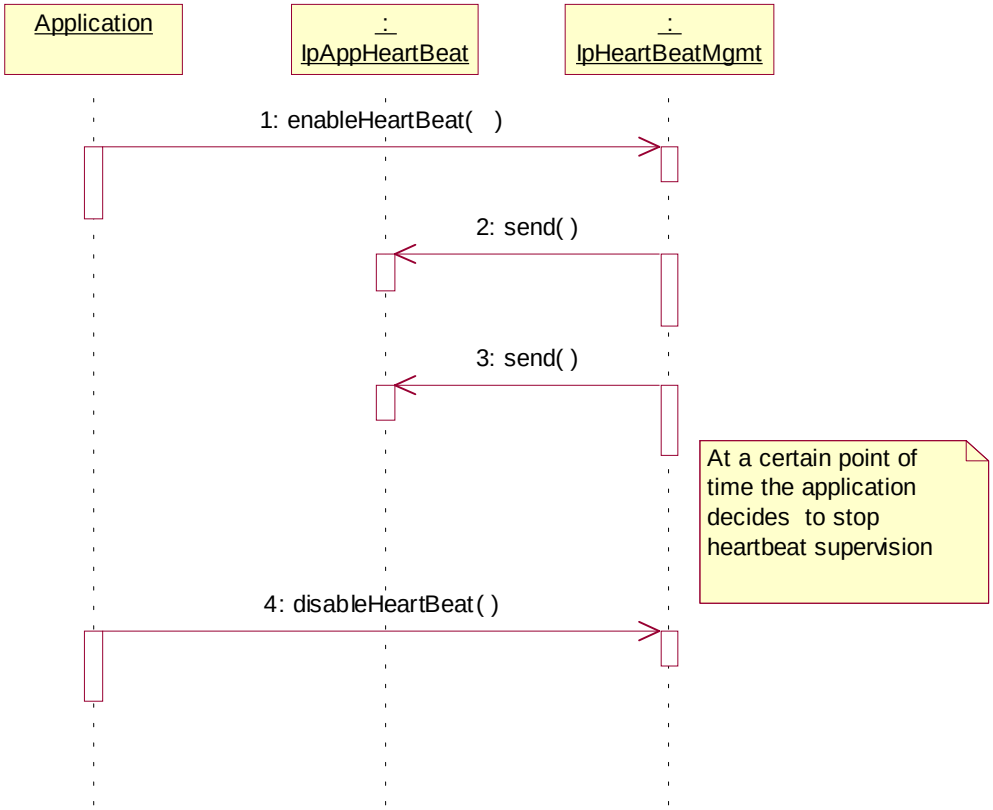


6.2.5 Load Management: Application callback registration and load control

This sequence diagram shows how an application registers itself and the framework invokes load management function based on policy.

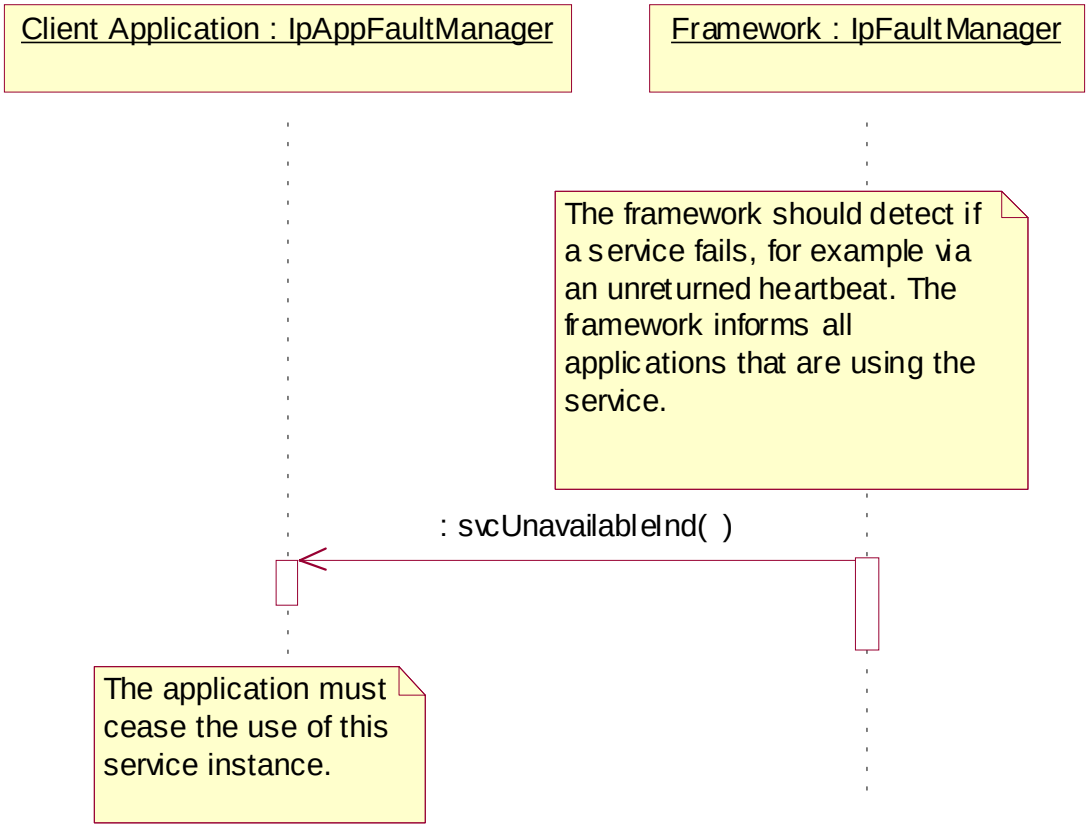


6.2.6 Heartbeat Management: Start/perform/end heartbeat supervision of application



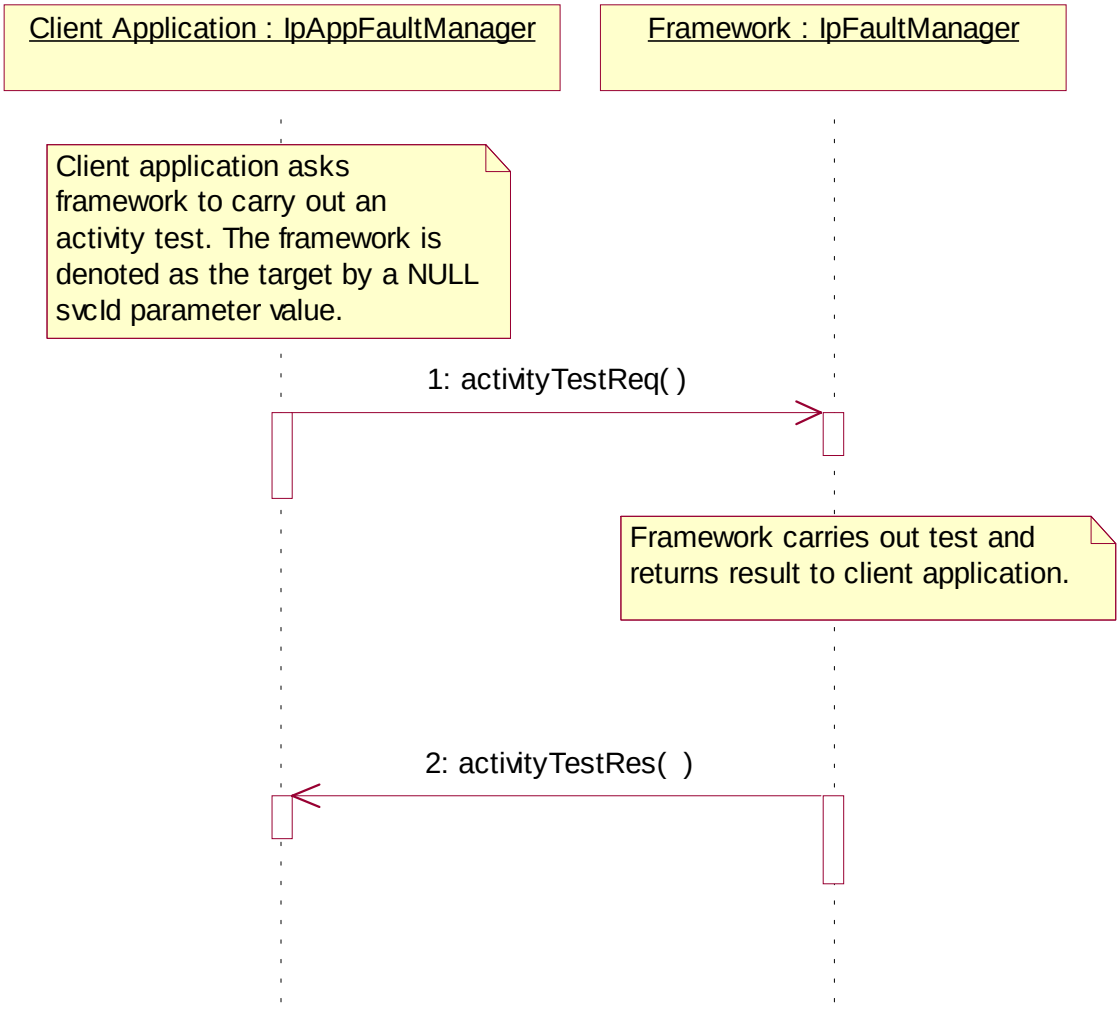
6.2.7 Fault Management: Framework detects a Service failure

The framework has detected that the service has failed (probably by the use of the heartbeat mechanism). The framework updates its own records and informs any client applications that are using the service to stop.



1: The framework informs each client application that is using the service instance that the service is unavailable. The client application is then expected to abandon use of this service instance and access a different service instance via the usual means (e.g. discovery, selectService etc.). The client application should not need to re-authenticate in order to discover and use an alternative service instance. The framework will also need to make the relevant updates to its internal records to make sure the service instance is removed from service and no client applications are still recorded as using it.

6.2.8 Fault Management: Application requests a Framework activity test



1: The client application asks the framework to do an activity test. The client identifies that it would like the activity test done for the framework, rather then a service, by supplying a NULL value for the svcId parameter.

2: The framework does the requested activity test and sends the result to the client application.

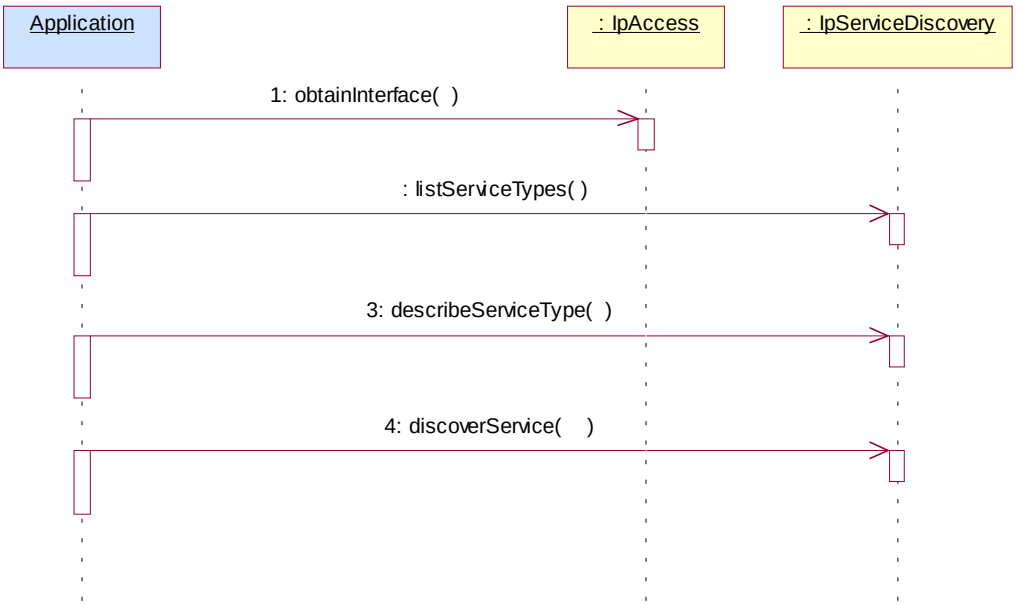
6.3 Service Discovery Sequence Diagrams

6.3.1 Service Discovery

The following figure shows how Applications discover a new Service Capability Feature in the network. Even applications that have already used the OSA API of a certain network know that the operator may upgrade it any time; this is why they use the Service Discovery interfaces.

Before the discovery process can start, the Application needs a reference to the Framework's Service Discovery interface; this is done via an invocation the method obtainInterface on the Framework's Access interface.

Discovery is a three-step process:



2: Discovery: first step - list service types

In this first step the application asks the Framework what service types that are available from this network. Service types are standardized or non-standardised SCF names, and thus this first step allows the Application to know what SCFs are supported by the network.

The following output is the result of this first discovery step:

· out listTypes

This is a list of service type names, i.e., a list of strings, each of them the name of a SCF or a SCF specialization (e.g. "P_MPCC").

3: Discovery: second step - describe service type

In this second step the application requests what are the properties that describe a certain service type that it is interested in, among those listed in the first step.

The following input is necessary:

· in name

This is a service type name: a string that contains the name of the SCF whose description the Application is interested in (e.g. "P_MPCC") .

And the output is:

· out serviceTypeDescription

The description of the specified SCF type. The description provides information about:

· the property names associated with the SCF,

· the corresponding property value types,

· the corresponding property mode (mandatory or read only) associated with each SCF property,

· the names of the super types of this type, and

· whether the type is currently enabled or disabled.

4: Discovery: third step - discover service

In this third step the application requests for a service that matches its needs by tuning the service properties (i. e., assigning values for certain properties).

The Framework then checks whether there is a match, in which case it sends the Application the serviceID that is the identifier this network operator has assigned to the SCF version described in terms of those service properties. This is the moment where the serviceID identifier is shared with the application that is interested on the corresponding service.

This is done for either one service or more (the application specifies the maximum number of responses it wishes to accept).

Input parameters are:

· in serviceTypeName

This is a string that contains the name of the SCF whose description the Application is interested in (e.g. "P_MPCC").

· in desiredPropertyList

This is again a list like the one used for service registration, but where the value of the service properties have been fine tuned by the Application to (they will be logically interpreted as "minimum", "maximum", etc. by the Framework).

The following parameter is necessary as input:

· in max

This parameter states the maximum number of SCFs that are to be returned in the "ServiceList" result.

And the output is:

· out serviceList

This is a list of duplets: (serviceID, servicePropertyList). It provides a list of SCFs matching the requirements from the Application, and about each: the identifier that has been assigned to it in this network (serviceID), and once again the service property list.

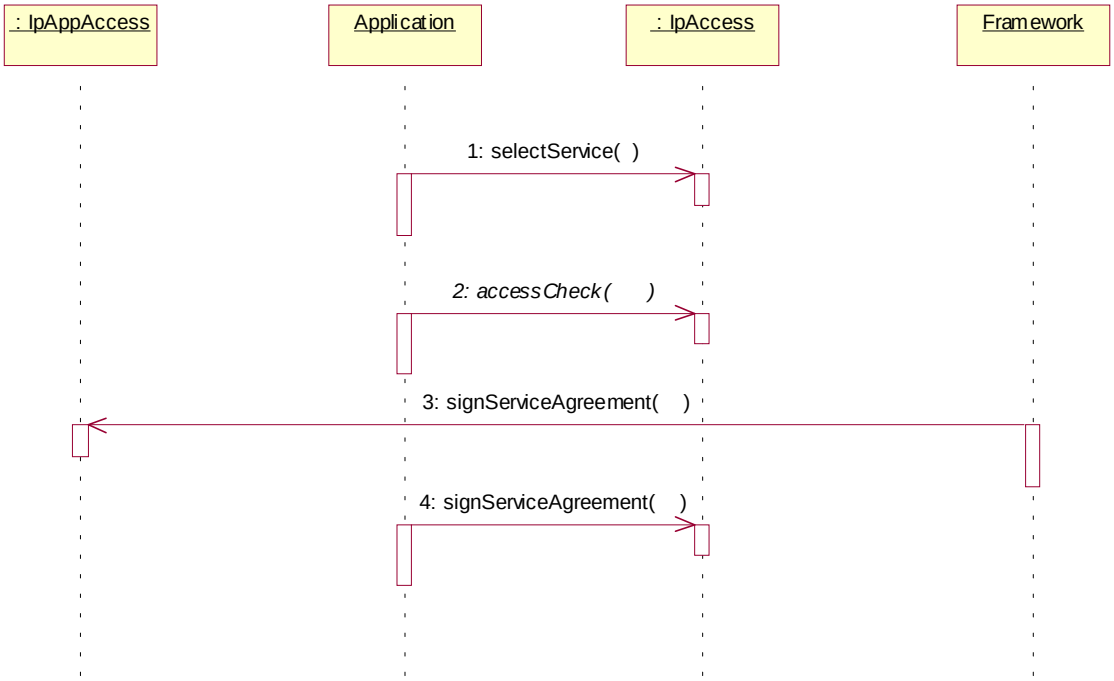
6.4 Trust and Security Management Sequence Diagrams

6.4.1 Service Selection

The following figure shows the process of selecting an SCF.

After discovery the Application gets a list of one or more SCF versions that match its required description. It now needs to decide which service it is going to use; it also needs to actually get a way to use it.

This is achieved by the following two steps:



1: Service Selection: first step - selectService

In this first step the Application identifies the SCF version it has finally decided to use. This is done by means of the serviceID, which is the agreed identifier for SCF versions. The Framework acknowledges this selection by returning to the Application a new identifier for the service chosen: a service token, that is a private identifier for this service between this Application and this network, and is used for the process of signing the service agreement.

Input is:

· in serviceID

This identifies the SCF required.

And output:

· out serviceToken

This is a free format text token returned by the framework, which can be signed as part of a service agreement. It contains operator specific information relating to the service level agreement.

3: Service Selection: second step - signServiceAgreement

In this second step an agreement is signed that allows the Application to use the chosen SCF version. And once this contractual details have been agreed, then the Application can be given the means to actually use it. The means are a reference to the manager interface of the SCF version (remember that a manager is an entry point to any SCF). By calling the getServiceManager operation on the service factory the Framework retrieves this interface and returns it to the Application. The service properties suitable for this application are also fed to the SCF (via the service factory interface) in order for the SCS to instantiate an SCF version that is suitable for this application.

Input:

· in serviceToken

This is the identifier that the network and Application have agreed to privately use for a certain version of SCF.

· in agreementText

This is the agreement text that is to be signed by the Framework using the private key of the Framework.

· in signingAlgorithm

This is the algorithm used to compute the digital signature.

Output:

· out signatureAndServiceMgr

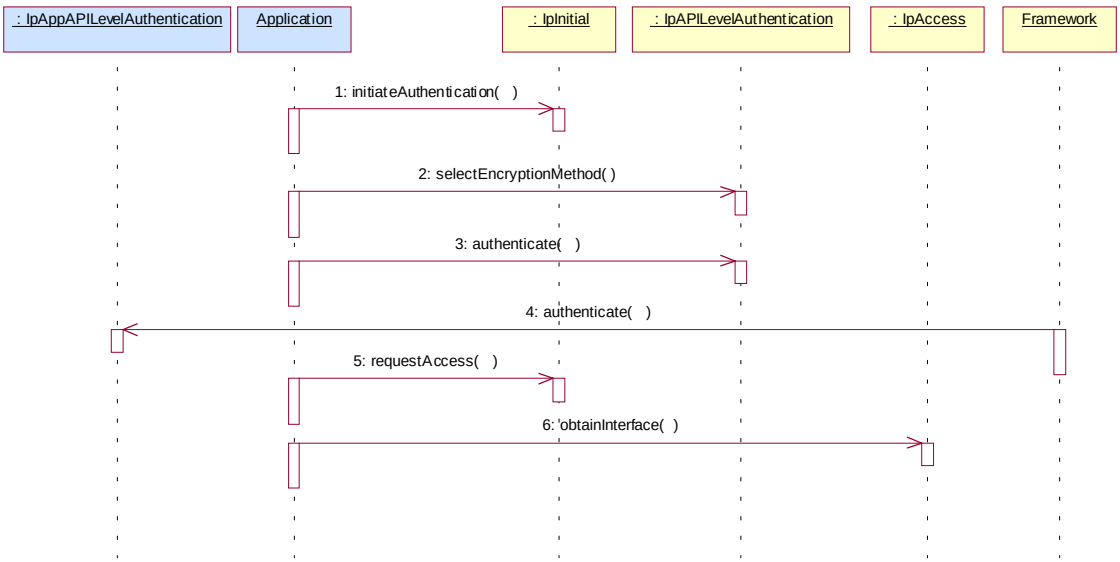
This is a reference to a structure containing the digital signature of the Framework for the service agreement, and a reference to the manager interface of the SCF.

6.4.2 Initial Access

The following figure shows an application accessing the OSA Framework for the first time.

Before being authorized to use the OSA SCFs, the Application must first of all authenticate itself with the Framework. For this purpose the application needs a reference to the Initial Contact interfaces for the Framework; this may be obtained through a URL, a Naming or Trading Service or an equivalent service, a stringified object reference, etc. At this stage, the Application has no guarantee that this is a Framework interface reference, but it to initiate the authentication process with the Framework. The Initial Contact interface only supports the initiateAuthentication method to allow the authentication process to take place.

Once the Application has authenticated with the Framework, it can gain access to other framework interfaces and SCFs. This is done by invoking the requestAccess method, by which the application requests a certain type of access SCF.



1: Initiate Authentication

The Application invokes initiateAuthentication on the Framework's "public" (initial contact) interface to initiate the authentication process. It provides in turn a reference to its own authentication interface. The Framework returns a reference to its authentication interface.

2: Select Encryption Method

The Application invokes selectAuthMethod on the Framework's API Level Authentication interface, identifying the authentication methods it supports. The Framework prescribes the method to be used.

3: Authenticate

4: The Application and Framework authenticate each other using the prescribed method. The sequence diagram illustrates one of a series of one or more invocations of the authenticate method on the Framework's API Level Authentication interface. In each invocation, the Application supplies a challenge and the Framework returns the correct response. Alternatively or additionally the Framework may issue its own challenges to the Application using the authenticate method on the Application's API Level Authentication interface.

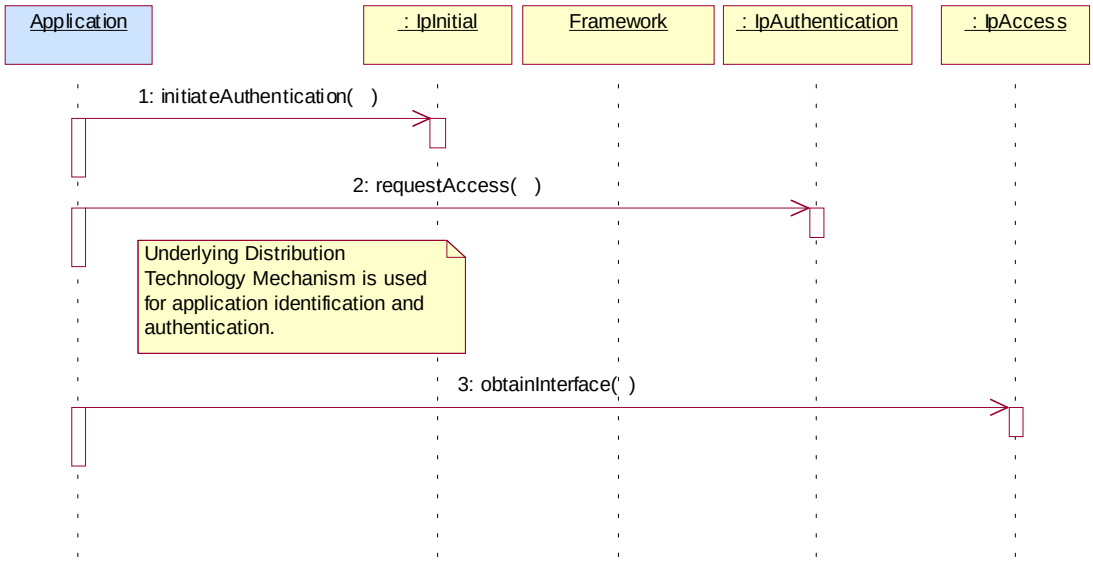
5: Request Access

Upon successful (mutual) authentication, the Application invokes requestAccess on the Framework's API Level Authenticaiton interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

6: The client invokes obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

6.4.3 Authentication

This sequence diagram illustrates the two-way mechanism by which the client and the framework mutually authenticate one another using an underlying distribution tecnology mechanism.



1: The application calls initiateAuthentication on the OSA Framework Initial interface. This allows the application to specify the type of authentication process. In this case, the application selects to use the underlying distribution technology mechanism for identification and authentication.

2: The application invokes the requestAccess method on the Framework's Authentication interface. The Framework now uses the underlying distribution technology mechanism for identification and authentication of the application.

3: If the authentication was successful, the application can now invoke obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

6.4.4 API Level Authentication

This sequence diagram illustrates the two-way mechanism by which the client application and the framework mutually authenticate one another.

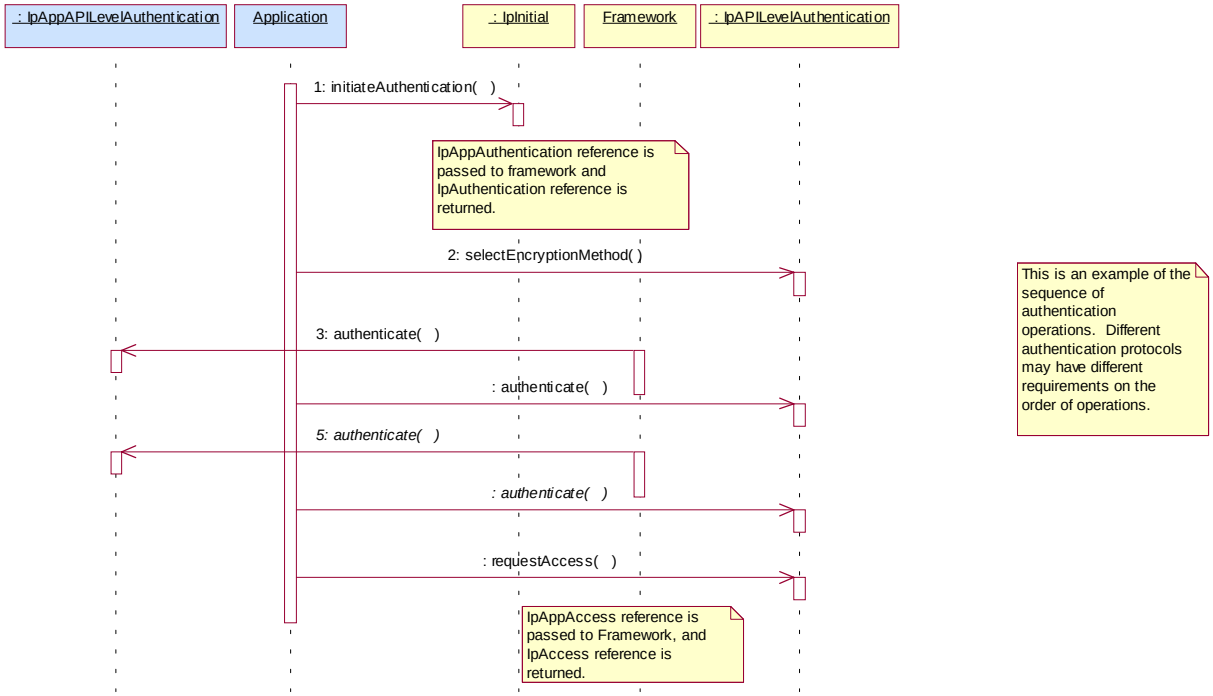
The OSA API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity. The inclusion of cryptographic processes and digital signatures in the authentication procedure depends on the type of authentication technique selected. In some cases strong authentication may need to be enforced by the Framework to prevent misuse of resources. In addition it may be necessary to define the minimum encryption key length that can be used to ensure a high degree of confidentiality.

The application must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

1) The application calls initiateAuthentication on the OSA Framework Initial interface. This allows the application to specify the type of authentication process. This authentication process may be specific to the provider, or the implementation technology used. The initiateAuthentication method can be used to specify the specific process, (e.g. CORBA security). OSA defines generic a authentication interface (API Level Authentication), which can be used to perform the authentication process. The initiateAuthentication method allows the application to pass a reference to its own authentication interface to the Framework, and receive a reference to the authentication interface preferred by the client, in return. In this case the API Level Authentication interface.

2) The application invokes the selectEncryptionMethod on the Framework's API Level Authentication interface. This includes the authentication capabilities of the application. The framework then chooses an authentication method based on the authentication capabilities of the application and the Framework. If the application is capable of handling more than one authentication method, then the Framework chooses one option, defined in the prescribedMethod parameter. In some instances, the authentication capability of the application may not fulfil the demands of the Framework, in which case, the authentication will fail.

3) The application and Framework interact to authenticate each other. Depending on the method prescribed, this procedure may consist of a number of messages e.g. a challenge/ response protocol. This authentication protocol is performed using the authenticate method on the API Level Authentication interface. Depending on the authentication method selected, the protocol may require invocations on the API Level Authentication interface supported by the Framework; or on the application counterpart; or on both.



Framework-to-Application Class Diagrams

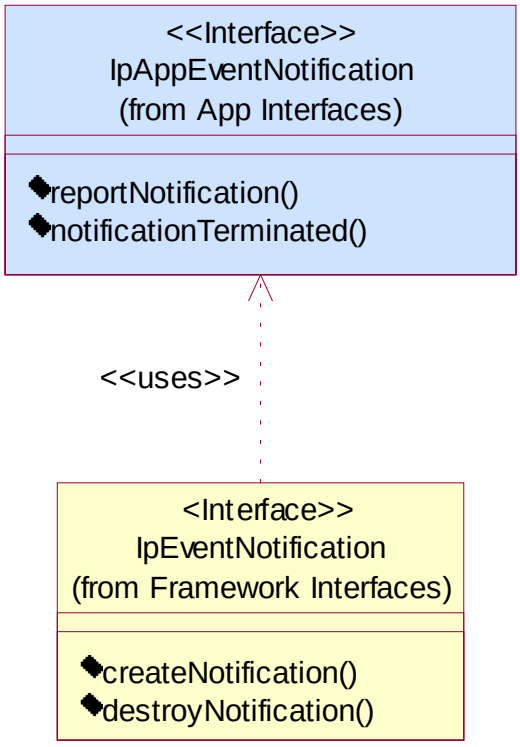


Figure: Event Notification Class Diagram

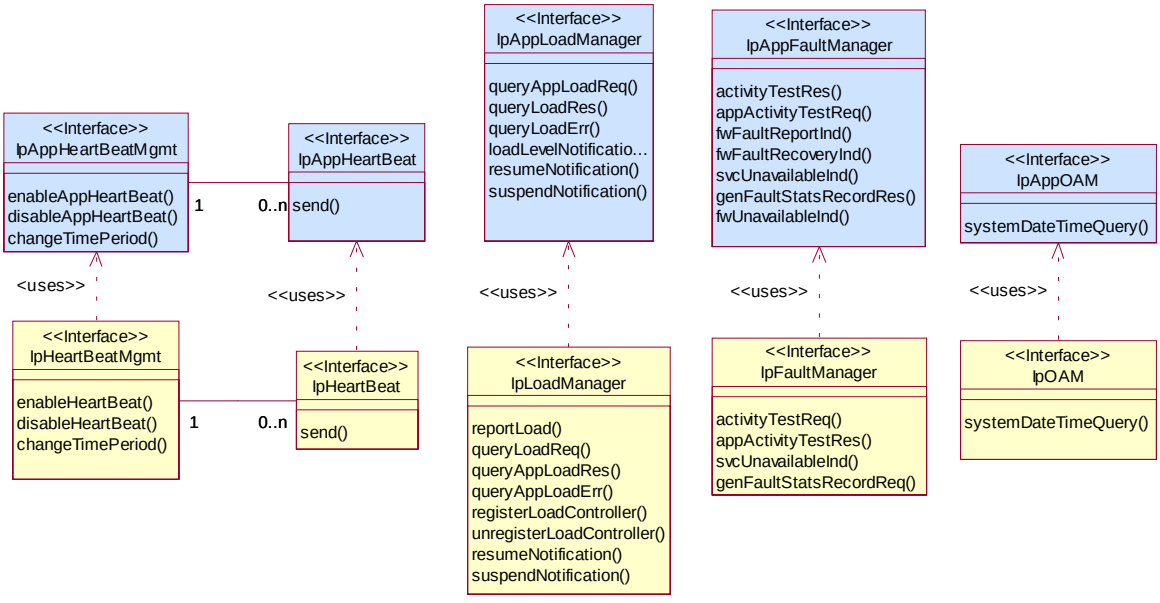


Figure: Integrity Management Package Overview

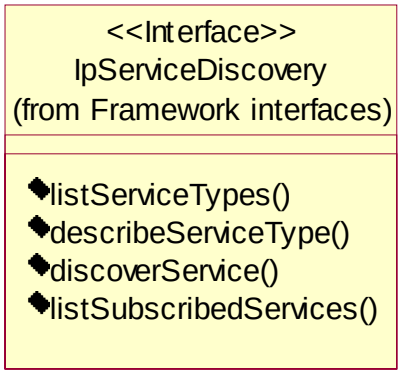


Figure: Service Discovery Package Overview

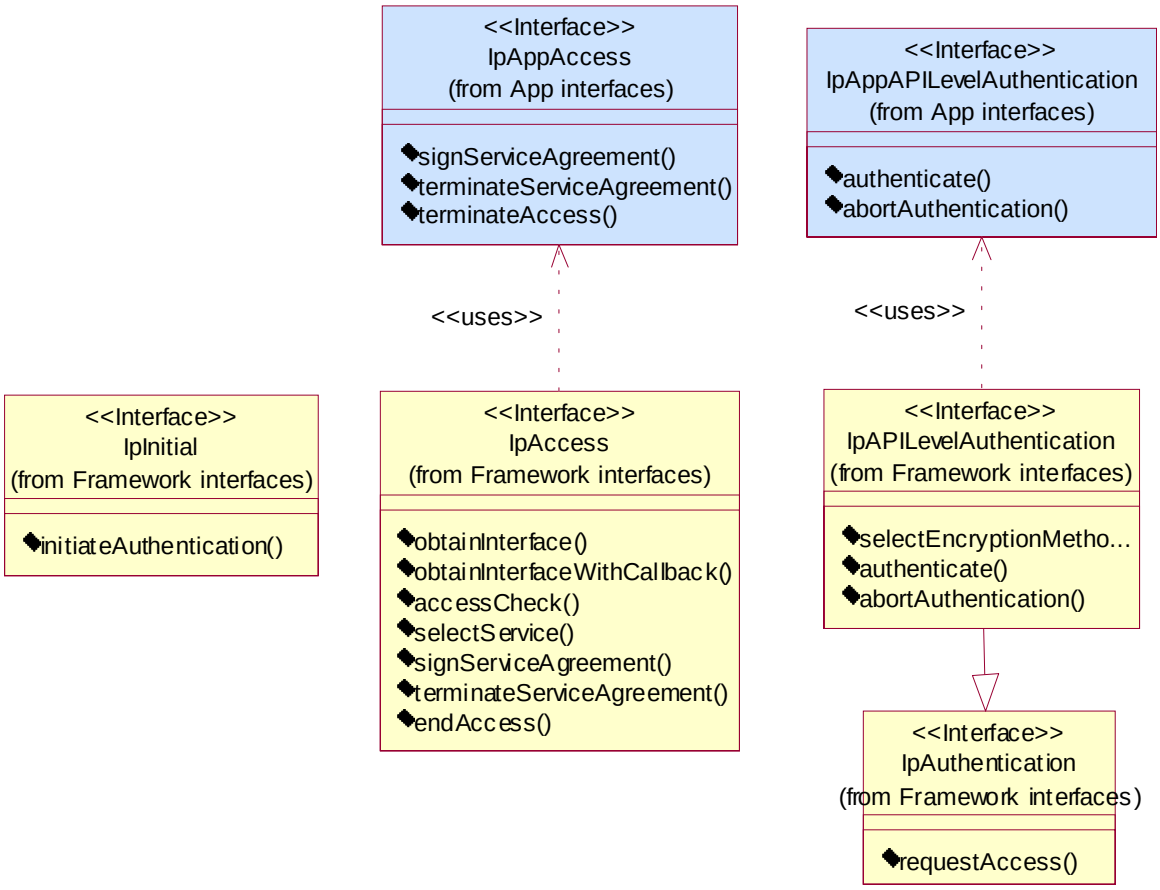


Figure: Trust and Security Management Package Overview

8 Framework-to-Application Interface Classes

8.1 Trust and Security Management Interface Classes

The Trust and Security Management Interfaces provide:

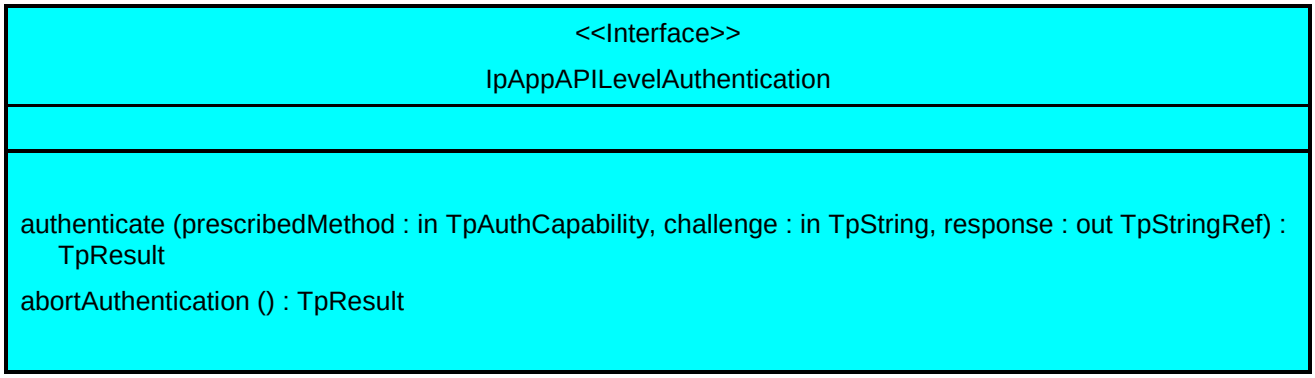
- the first point of contact for an application to access a Home Environment;
- the authentication methods for the application and Home Environment to perform an authentication protocol;
- the application with the ability to select a service capability feature to make use of;
- the application with a portal to access other Framework interfaces.

The process by which the application accesses the Home Environment has been separated into 3 stages, each supported by a different Framework interface:

1) Initial Contact with the Framework;
2) Authentication to the Framework;
3) Access to Framework and Service Capability Features.

8.1.1 Interface Class IpAppAPILevelAuthentication

Inherits from: IpInterface.



Method

authenticate()

This method is used by the framework to authenticate the client application using the mechanism indicated in prescribedMethod. The client application must respond with the correct responses to the challenges presented by the framework. The number of exchanges and the order of the exchanges is dependent on the prescribedMethod. (These may be interleaved with authenticate() calls by the client application on the IpAPILevelAuthentication interface. This is defined by the prescribedMethod.)

Parameters

prescribedMethod : in TpAuthCapability

see selectEncryptionMethod() on the IpAPIlLevelAuthentication interface. This parameter contains the agreed method for authentication. If this is not the same value as returned by selectEncryptionMethod(), then an error code (P_INVALID_AUTH_CAPABILITY) is returned.

challenge : in TpString

The challenge presented by the framework to be responded to by the client application. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

response : out TpStringRef

This is the response of the client application to the challenge of the framework in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectEncryptionMethod().

Raises

TpGeneralException,TpFWException

Method

abortAuthentication()

The framework uses this method to abort the authentication process. This method is invoked if the framework wishes to abort the authentication process, (e.g. if the client application responds incorrectly to a challenge.) If this method has

been invoked, calls to the requestAccess operation on IpAPILevelAuthentication will return an error code (P_ACCESS_DENIED), until the client application has been properly authenticated.

Parameters
No Parameters were identified for this method

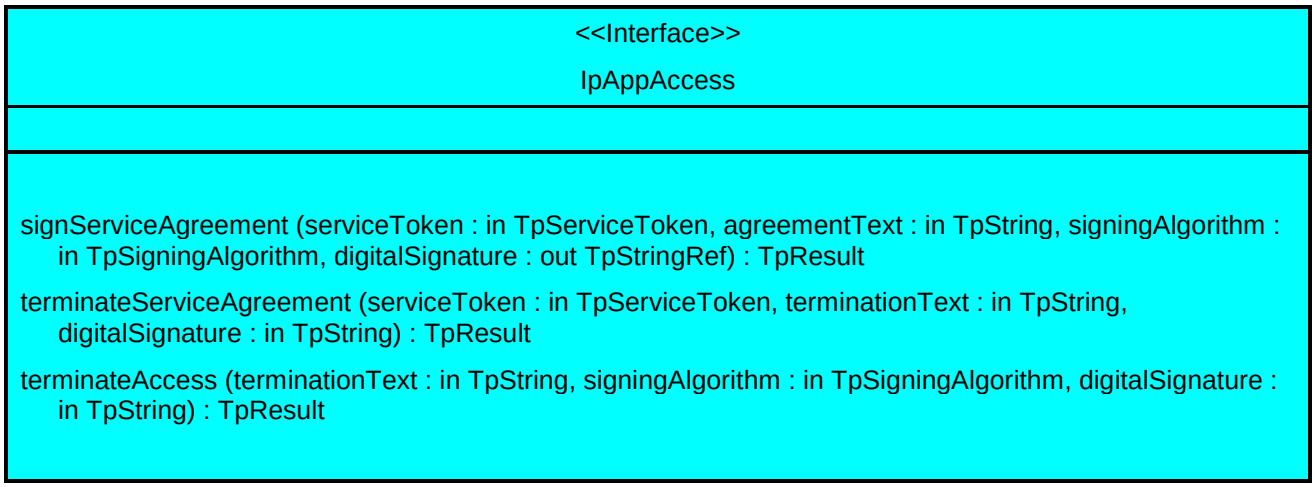
Raises

TpGeneralException,TpFWException

8.1.2 Interface Class IpAppAccess

Inherits from: IpInterface.

The Access client application interface is used by the Framework to perform the steps that are necessary in order to allow it to service access.



Method
signServiceAgreement()

This method is used by the framework to request that the client application sign an agreement on the service. It is called in response to the client application calling the selectService() method on the IpAccess interface of the framework. The framework provides the service agreement text for the client application to sign. If the client application agrees, it signs the service agreement, returning its digital signature to the framework.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance to which this service agreement corresponds. (If the client application selects many services, it can determine which selected service corresponds to the service agreement by matching the service token.) If the serviceToken is invalid, or not known by the client application,then an error code (P_INVALID_SERVICE_TOKEN) is returned.

agreementText : in TpString

This is the agreement text that is to be signed by the client application using the private key of the client application. If the agreementText is invalid, then an error code (P_INVALID_AGREEMENT_TEXT) is returned.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. If the signingAlgorithm is invalid, or unknown to the client application, an error code (P_INVALID_SIGNING_ALGORITHM) is returned.

digitalSignature : out TpStringRef

The digitalSignature is the signed version of a hash of the service token and agreement text given by the framework.

Raises

TpGeneralException,TpFWException

Method

terminateServiceAgreement()

This method is used by the framework to terminate an agreement for the service.

Parameters

serviceToken : in TpServiceToken

This is the token passed back from the framework in a previous selectService() method call. This token is used to identify the service agreement to be terminated. If the serviceToken is invalid, or unknown to the client application, an error code (P_INVALID_SERVICE_TOKEN) is returned.

terminationText : in TpString

This is the termination text that describes the reason for the termination of the service agreement.

digitalSignature : in TpString

This is a signed version of a hash of the service token and the termination text. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the service agreement is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

Raises

TpGeneralException,TpFWException

Method

terminateAccess()

The terminateAccess operation is used to end the client application's access session with the framework. The framework is terminating the client application's access session. (For example, this may be done if the framework believes the client application is masquerading as someone else. Using this operation will force the client application to re-authenticate if it wishes to continue using the framework's services.)

After terminateAccess() is invoked, the client application will not longer be authenticated with the framework. The client application will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail.

Parameters

terminationText : in TpString

This is the termination text describes the reason for the termination of the access session.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. If the signingAlgorithm is invalid, or unknown to the client application, an error code (P_INVALID_SIGNING_ALGORITHM) is returned.

digitalSignature : in TpString

This is a signed version of a hash of the termination text. The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the access session is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

Raises

TpGeneralException,TpFWException

8.1.3 Interface Class IpInitial

Inherits from: IpInterface.

The Initial Framework interface is used by the client application to initiate the mutual authentication with the Framework.

<<Interface>> IpInitial
initiateAuthentication (appDomain : in TpAuthDomain, authType : in TpAuthType, fwDomain : out TpAuthDomainRef) : TpResult

Method

initiateAuthentication()

This method is invoked by the client application to start the process of mutual authentication with the framework, and request the use of a specific authentication method.

Parameters

appDomain : in TpAuthDomain

This identifies the application domain to the framework, and provides a reference to the domain's authentication interface.

```
structure TpAuthDomain {
    domainID:       TpDomainID;
    authInterface:  IpInterfaceRef;
};
```

The domainID parameter is an identifier either for a client application (i.e. TpClientAppID) or for an enterprise operator (i.e. TpEntOpID). It is used to identify the enterprise domain to the framework, (see authenticate() on IpAPILevelAuthentication). If the framework does not recognise the domainID, the framework returns an error code (P_INVALID_DOMAIN_ID).

The authInterface parameter is a reference to call the authentication interface of the client application. The type of this interface is defined by the authType parameter. If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

`authType : in TpAuthType`

This identifies the type of authentication mechanism requested by the client. It provides operators and clients with the opportunity to use an alternative to the API level Authentication interface, e.g. an implementation specific authentication mechanism like CORBA Security, using the Authentication interface, or Operator specific Authentication interfaces. OSA API level Authentication is the default authentication mechanism (P_OSA_AUTHENTICATION). If P_OSA_AUTHENTICATION is selected, then the appDomain and fwDomain authInterface parameters are references to interfaces of type Ip(App)APILevelAuthentication. If P_AUTHENTICATION is selected, the authInterface parameters are refereces to interfaces of type Ip(App)Authentication which is used when an underlying distibution technology authentication mechanism is used.

`fwDomain : out TpAuthDomainRef`

This provides the application domain with a framework identifier, and a reference to call the authentication interface of the framework.

```
structure TpAuthDomain {
    domainID:       TpDomainID;
    authInterface:  IpInterfaceRef;
    };
```

The domainID parameter is an identifier for the framework (i.e. TpFwID). It is used to identify the framework to the enterprise domain.

The authInterface parameter is a reference to the authentication interface of the framework. The type of this interface is defined by the authType parameter. The application domain uses this interface to authenticate with the framework.

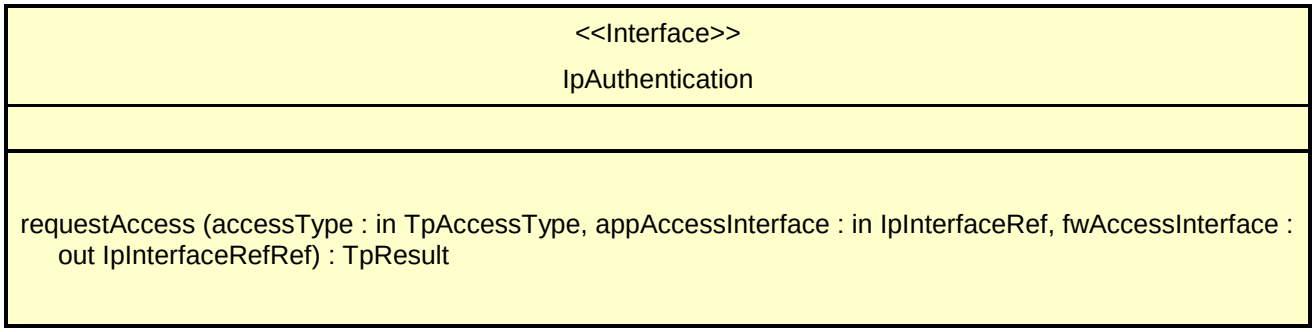
Raises

`TpGeneralException,TpFWException`

8.1.4 Interface Class IpAuthentication

Inherits from: IpInterface.

The Authentication Framework interface is used by client application to request access to other interfaces supported by the Framework. The mutual authentication process should in this case be done with some underlying distribution technology authentication mechanism, e.g. CORBA Security.



Method

`requestAccess()`

Once application and framework are authenticated, the client application invokes the requestAccess operation on the IpAuthentication or IpAPILevelAuthentication interface. This allows the client application to request the type of access they require. If they request P_OSA_ACCESS, then a reference to the IpAccess interface is returned. (Operators can define their own access interfaces to satisfy client requirements for different types of access.)

If this method is called before the client application and framework have successfully completed the authentication process, then the request fails, and an error code (P_ACCESS_DENIED) is returned.

Parameters

accessType : in TpAccessType

This identifies the type of access interface requested by the client application. If the framework does not provide the type of access identified by accessType, then an error code (P_INVALID_ACCESS_TYPE) is returned.

appAccessInterface : in IpInterfaceRef

This provides the reference for the framework to call the access interface of the client application. If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

fwAccessInterface : out IpInterfaceRefRef

This provides the reference for the client application to call the access interface of the framework.

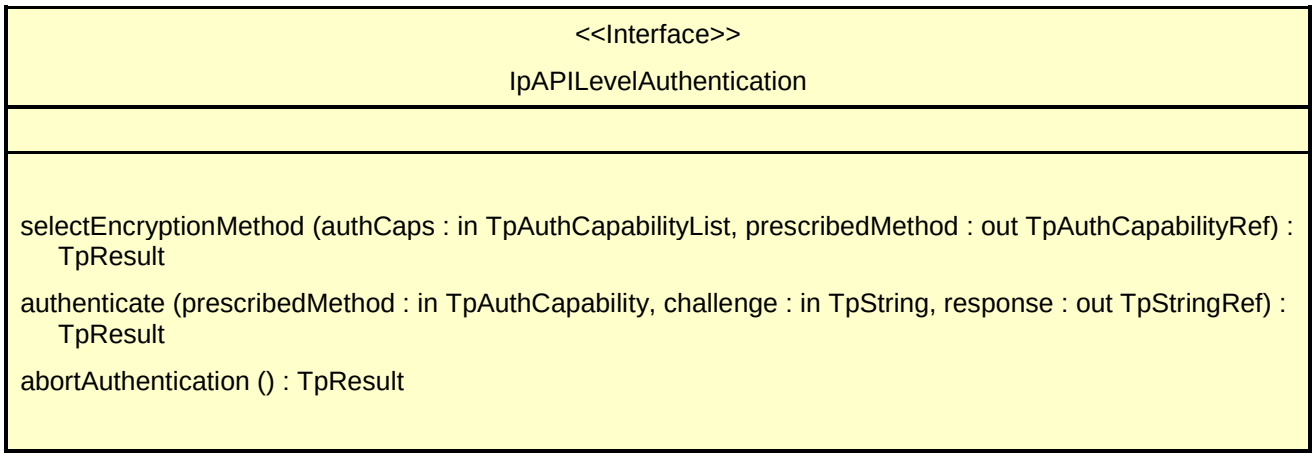
Raises

TpGeneralException,TpFWException

8.1.5 Interface Class IpAPILevelAuthentication

Inherits from: IpAuthentication.

The API Level Authentication Framework interface is used by client application to perform its part of the mutual authentication process with the Framework necessary to be allowed to use any of the other interfaces supported by the Framework.



Method

selectEncryptionMethod()

The client application uses this method to initiate the authentication process. The framework returns its preferred mechanism. This should be within capability of the client application. If a mechanism that is acceptable to the framework within the capability of the client application cannot be found, the framework returns an error code (P_NO_ACCEPTABLE_AUTH_CAPABILITY).

Parameters

authCaps : in TpAuthCapabilityList

This is the means by which the authentication mechanisms supported by the client application are conveyed to the framework.

prescribedMethod : out TpAuthCapabilityRef

This is returned by the framework to indicate the mechanism preferred by the framework for the authentication process. If the value of the prescribedMethod returned by the framework is not understood by the client application, it is considered a catastrophic error and the client application must abort.

Raises

TpGeneralException,TpFWException

Method

authenticate()

This method is used by the client application to authenticate the framework using the mechanism indicated in prescribedMethod. The framework must respond with the correct responses to the challenges presented by the client application. The clientAppID received in the initiateAuthentication() can be used by the framework to reference the correct public key for the client application (the key management system is currently outside of the scope of the OSA APIs). The number of exchanges and the order of the exchanges is dependent on the prescribedMethod.

Parameters

prescribedMethod : in TpAuthCapability

see selectEncryptionMethod(). This parameter contains the method that the framework has specified as acceptable for authentication. If this is not the same value as returned by selectEncryptionMethod(), then the framework returns an error code (P_INVALID_AUTH_CAPABILITY).

challenge : in TpString

The challenge presented by the client application to be responded to by the framework. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

response : out TpStringRef

This is the response of the framework to the challenge of the client application in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectEncryptionMethod().

Raises

TpGeneralException,TpFWException

Method

abortAuthentication()

The client application uses this method to abort the authentication process. This method is invoked if the client application no longer wishes to continue the authentication process, (e.g. if the framework responds incorrectly to a challenge.) If this method has been invoked, calls to the requestAccess operation on IpAPILevelAuthentication will return an error code (P_ACCESS_DENIED), until the client application has been properly authenticated.

Parameters

No Parameters were identified for this method

Raises

TpGeneralException,TpFWException

8.1.6 Interface Class IpAccess

Inherits from: IpInterface.

<<Interface>> IpAccess
obtainInterface (interfaceName : in TpInterfaceName, fwInterface : out IpInterfaceRefRef) : TpResult obtainInterfaceWithCallback (interfaceName : in TpInterfaceName, appInterface : in IpInterfaceRef, fwInterface : out IpInterfaceRefRef) : TpResult accessCheck (serviceToken : in TpServiceToken, securityContext : in TpSecurityContext, securityDomain : in TpSecurityDomain, group : in TpSecurityGroup, serviceAccessTypes : in TpServiceAccessType, serviceAccessControl : out TpServiceAccessControlRef) : TpResult selectService (serviceID : in TpServiceID, serviceToken : out TpServiceTokenRef) : TpResult signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm, signatureAndServiceMgr : out TpSignatureAndServiceMgrRef) : TpResult terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpString) : TpResult endAccess (endAccessProperties : in TpEndAccessProperties) : TpResult

Method

obtainInterface()

This method is used to obtain other framework interfaces. The client application uses this method to obtain interface references to other framework interfaces. (The obtainInterfacesWithCallback method should be used if the client application is required to supply a callback interface to the framework.)

Parameters

interfaceName : in TpInterfaceName

The name of the framework interface to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

fwInterface : out IpInterfaceRefRef

This is the reference to the interface requested.

Raises

TpGeneralException,TpFWException

Method

obtainInterfaceWithCallback()

This method is used to obtain other framework interfaces. The client application uses this method to obtain interface references to other framework interfaces, when it is required to supply a callback interface to the framework. (The obtainInterface method should be used when no callback interface needs to be supplied.)

Parameters

interfaceName : in TpInterfaceName

The name of the framework interface to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

appInterface : in IpInterfaceRef

This is the reference to the client application interface, which is used for callbacks. If an application interface is not needed, then this method should not be used. (The obtainInterface method should be used when no callback interface needs to be supplied.) If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

fwInterface : out IpInterfaceRefRef

This is the reference to the interface requested.

Raises

TpGeneralException,TpFWException

Method

accessCheck()

This method may be used by the client application to check if it is authorised to access the specified service. The response is used to indicate whether the request for access has been granted or denied and if granted the level of trust that will be applied. The securityModelID and the relevant securityLevel are defined as part of the registration data for the service, and the service agreement. They are specific to the service.

securityModelID:

The identity of the specific Security Model that is to be used to define a set of appropriate policies for the service that can be used by the framework to determine access rights. The model may include blanket permission, session permission or one shot permission. A number of security models will be stored by the framework, and referenced by the access control module, according to the security model identifier of the service.

securityLevel:

The trust level required by the service for granting access. The Security Level is used by the framework's access control module when it checks for access rights.

Parameters

serviceToken : in TpServiceToken

The serviceToken identifies the specific service that the client application wishes to access. The service Token identifies the service type and service properties selected by the client application when it invoked selectService().

securityContext : in TpSecurityContext

A context is a group of security relevant attributes that may have an influence on the result of the accessCheck request.

securityDomain : in TpSecurityDomain

The security domain in which the client application is operating may influence the access control decisions and the specific set of features that the requestor is entitled to use.

group : in TpSecurityGroup

A group can be used to define the access rights associated with all client applications that belong to that group. This simplifies the administration of access rights.

serviceAccessTypes : in TpServiceAccessType

These are defined by the specific Security Model in use but are expected to include: Create, Read, Update, Delete as well as those specific to services.

serviceAccessControl : out TpServiceAccessControlRef

This contains the access control policy information that controls access to the service feature, and the trustLevel that the service provider has assigned to the client application.

```
structure TpServiceAccessControl {
                policy:     TpString;
                trustLevel:     TpString;
                };
```

The policy parameter indicates whether access has been granted or denied. If granted then the parameter trustLevel must also have a value.

The trustLevel parameter indicates the trust level that the service provider has assigned to the client application.

Raises

TpGeneralException,TpFWException

Method

selectService()

This method is used by the client application to identify the service that the client application wishes to use. If the client application is not allowed to access the service, then an error code (P_SERVICE_ACCESS_DENIED) is returned.

Parameters

serviceID : in TpServiceID

This identifies the service required. If the serviceID is not recognised by the framework, an error code (P_INVALID_SERVICE_ID) is returned.

serviceToken : out TpServiceTokenRef

This is a free format text token returned by the framework, which can be signed as part of a service agreement. This will contain operator specific information relating to the service level agreement. The serviceToken has a limited lifetime. If the lifetime of the serviceToken expires, a method accepting the serviceToken will return an error code (P_INVALID_SERVICE_TOKEN). Service Tokens will automatically expire if the client application or framework invokes the endAccess method on the other's corresponding access interface.

Raises

TpGeneralException,TpFWException

Method

signServiceAgreement()

This method is used by the client application to request that the framework sign an agreement on the service, which allows the client application to use the service. If the framework agrees, both parties sign the service agreement, and a reference to the service manager interface of the service is returned to the client application. If the client application is not allowed to access the service, then an error code (P_SERVICE_ACCESS_DENIED) is returned.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance requested by the client application. If the serviceToken is invalid, or has expired, an error code (P_INVALID_SERVICE_TOKEN) is returned.

agreementText : in TpString

This is the agreement text that is to be signed by the framework using the private key of the framework. If the agreementText is invalid, then an error code (P_INVALID_AGREEMENT_TEXT) is returned.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. If the signingAlgorithm is invalid, or unknown to the framework, an error code (P_INVALID_SIGNING_ALGORITHM) is returned.

signatureAndServiceMgr : out TpSignatureAndServiceMgrRef

This contains the digital signature of the framework for the service agreement, and a reference to the service manager interface of the service.

```
structure TpSignatureAndServiceMgr {
        digitalSignature: TpString;
        serviceMgrInterface: IpInterfaceRef;
    };
```

The digitalSignature is the signed version of a hash of the service token and agreement text given by the client application.

The serviceMgrInterface is a reference to the service manager interface for the selected service.

Raises

TpGeneralException,TpFWException

Method

terminateServiceAgreement()

This method is used by the client application to terminate an agreement for the service.

Parameters

serviceToken : in TpServiceToken

This is the token passed back from the framework in a previous selectService() method call. This token is used to identify the service agreement to be terminated. If the serviceToken is invalid, or has expired, an error code (P_INVALID_SERVICE_TOKEN) is returned.

terminationText : in TpString

This is the termination text describes the reason for the termination of the service agreement.

digitalSignature : in TpString

This is a signed version of a hash of the service token and the termination text. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement().The framework uses this to check that the terminationText has been signed by the client application. If a match is made, the service agreement is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

Raises

TpGeneralException,TpFWException

Method

endAccess()

The endAccess operation is used to end the client application's access session with the framework. The client application requests that its access session is ended. After it is invoked, the client application will no longer be authenticated with the framework. The client application will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail.

Parameters

endAccessProperties : in TpEndAccessProperties

This is a list of properties that can be used to tell the framework the actions to perform when ending the access session (e.g. existing service sessions may be stopped, or left running). If a property is not recognised by the framework, an error code (P_INVALID_PROPERTY) is returned.

Raises

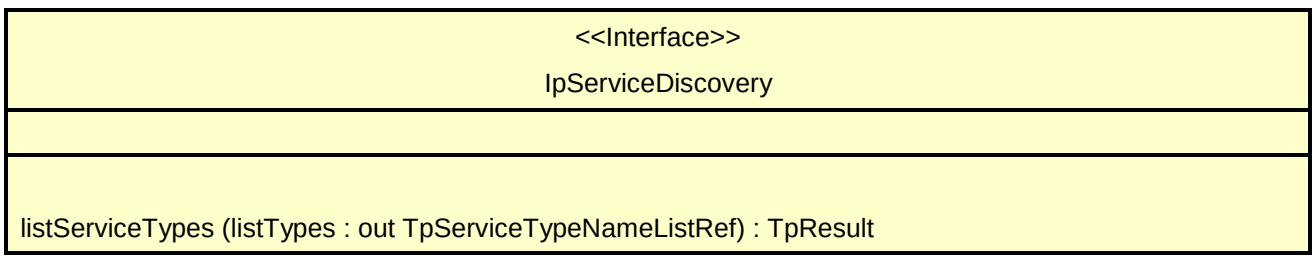
TpGeneralException,TpFWException

8.2 Service Discovery Interface Classes

8.2.1 Interface Class IpServiceDiscovery

Inherits from: IpInterface.

The service discovery interface, shown below, consists of four methods. Before a service can be discovered, the enterprise operator (or the client applications) must know what "types" of services are supported by the Framework and what service "properties" are applicable to each service type. The "listServiceType() method returns a list of all "service types" that are currently supported by the framework and the "describeServiceType()" returns a description of each service type. The description of service type includes the "service-specific properties" that are applicable to each service type. Then the enterprise operator (or the client applications) can discover a specific set of registered services that both belong to a given type and possess the desired "property values", by using the "discoverService() method. Once the enterprise operator finds out the desired set of services supported by the framework, it subscribes to (a sub-set of) these services using the Subscription Interfaces. The enterprise operator (or the client applications in its domain) can find out the set of services available to it (i.e., the service that it can use) by invoking "listSubscribedServices()". The service discovery APIs are invoked by the enterprise operators or client applications. They are described below.



describeServiceType (name : in TpServiceTypeName, serviceTypeDescription : out TpServiceTypeDescriptionRef) : TpResult

discoverService (serviceTypeName : in TpServiceTypeName, desiredPropertyList : in TpServicePropertyList, max : in TpInt32, serviceList : out TpServiceListRef) : TpResult

listSubscribedServices (serviceList : out TpServiceListRef) : TpResult

Method

listServiceTypes()

This operation returns the names of all service types that are in the repository. The details of the service types can then be obtained using the describeServiceType() method.

Parameters

listTypes : out TpServiceTypeNameListRef

The names of the requested service types.

Raises

TpGeneralException,TpFWException

Method

describeServiceType()

This operation lets the caller obtain the details for a particular service type.

Parameters

name : in TpServiceTypeName

The name of the service type to be described.

· If the "name" is malformed, then the P_ILLEGAL_SERVICE_TYPE exception is raised.

· If the "name" does not exist in the repository, then the P_UNKNOWN_SERVICE_TYPE exception is raised.

serviceTypeDescription : out TpServiceTypeDescriptionRef

The description of the specified service type. The description provides information about:
- · the service properties associated with this service type: i.e. a list of service property {name, mode and type} tuples,
- · the names of the super types of this service type, and
- · whether the service type is currently enabled or disabled.

Raises

TpGeneralException,TpFWException

Method

discoverService()

The discoverService operation is the means by which a client application is able to obtain the service IDs of the services that meet its requirements. The client application passes in a list of desired service properties to describe the service it is looking for, in the form of attribute/value pairs for the service properties. The client application also specifies the maximum number of matched responses it is willing to accept. The framework must not return more matches than the specified maximum, but it is up to the discretion of the Framework implementation to choose to return less than the specified maximum. The discoverService() operation returns a serviceID/Property pair list for those services that match the desired service property list that the client application provided.

Parameters

serviceTypeName : in TpServiceTypeName

The "serviceTypeName" parameter conveys the required service type. It is key to the central purpose of "service trading". It is the basis for type safe interactions between the service exporters (via registerService) and service importers (via discoverService). By stating a service type, the importer implies the service type and a domain of discourse for talking about properties of service.

· If the string representation of the "type" does not obey the rules for service type identifiers, then the P_ILLEGAL_SERVICE_TYPE exception is raised.

· If the "type" is correct syntactically but is not recognised as a service type within the Framework, then the P_UNKNOWN_SERVICE_TYPE exception is raised.

The framework may return a service of a subtype of the "type" requested. A service sub-type can be described by the properties of its supertypes.

desiredPropertyList : in TpServicePropertyList

The "desiredPropertyList"parameter is a list of service property {name, mode and value list} tuples that the discovered set of services should satisfy. These properties deal with the non-functional and non-computational aspects of the desired service. The property values in the desired property list must be logically interpreted as "minimum", "maximum", etc. by the framework (due to the absence of a Boolean constraint expression for the specification of the service criterion). It is suggested that, at the time of service registration, each property value be specified as an appropriate range of values, so that desired property values can specify an "enclosing" range of values to help in the selection of desired services.

max : in TpInt32

The "max" parameter states the maximum number of services that are to be returned in the "serviceList" result.

serviceList : out TpServiceListRef

This parameter gives a list of matching services. Each service is characterised by its service ID and a list of service property {name, mode and value list} tuples associated with the service.

Raises

TpGeneralException,TpFWException

Method

listSubscribedServices()

Returns a list of services so far subscribed by the enterprise operator. The enterprise operator (or the client applications in the enterprise domain) can obtain a list of subscribed services that they are allowed to access.

Parameters

serviceList : out TpServiceListRef

The "serviceList" parameter returns a list of subscribed services. Each service is characterised by its service ID and a list of service property {name, mode and value list} tuples associated with the service.

Raises

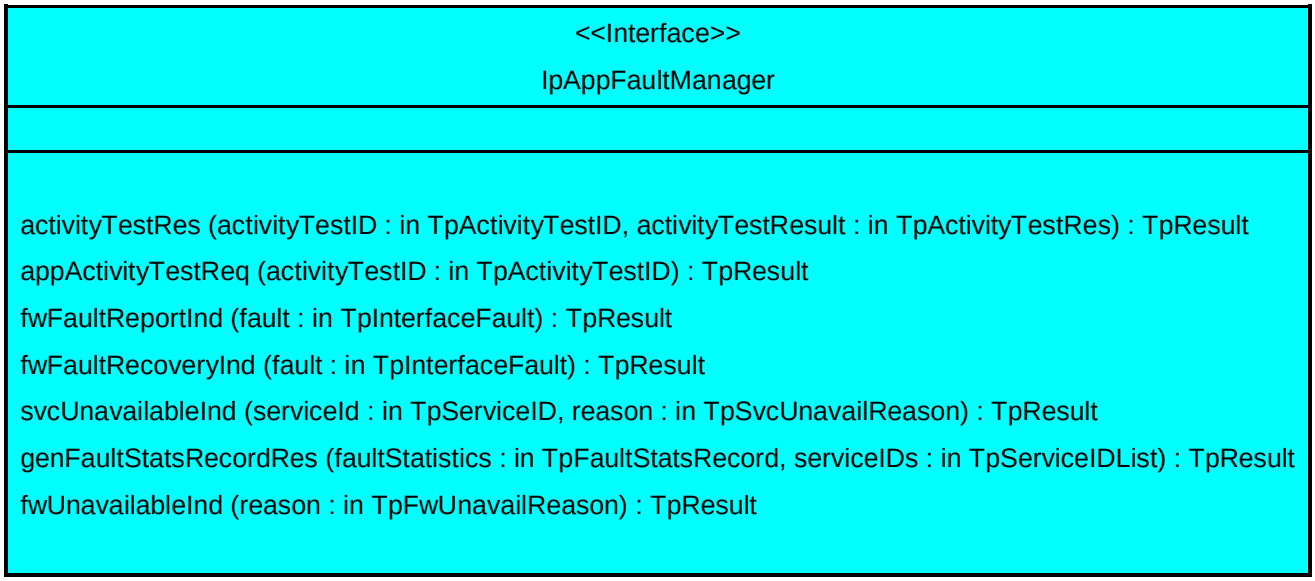
TpGeneralException,TpFWException

8.3 Integrity Management Interface Classes

8.3.1 Interface Class IpAppFaultManager

Inherits from: IpInterface.

This interface is used to inform the application of events that affect the integrity of the Framework, Service or Client Application. The Fault Management Framework will invoke methods on the Fault Management Application Interface that is specified when the client application obtains the Fault Management interface: i.e. by use of the obtainInterfaceWithCallback operation on the IpAccess interface



Method

activityTestRes()

The framework uses this method to return the result of a client application-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the client application to correlate this response (when it arrives) with the original request.

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpGeneralException,TpFWException

Method

appActivityTestReq()

The framework invokes this method to test that the client application is operational. On receipt of this request, the application must carry out a test on itself, to check that it is operating correctly. The application reports the test result by invoking the appActivityTestRes method on the IpFaultManager interface.

Parameters

activityTestID : in TpActivityTestID

The identifier provided by the framework to correlate the response (when it arrives) with this request.

Raises

TpGeneralException,TpFWException

Method

fwFaultReportInd()

The framework invokes this method to notify the client application of a failure within the framework. The client application must not continue to use the framework until it has recovered (as indicated by a fwFaultRecoveryInd).

Parameters

fault : in TpInterfaceFault

Specifies the fault that has been detected by the framework.

Raises

TpGeneralException,TpFWException

Method

fwFaultRecoveryInd()

The framework invokes this method to notify the client application that a previously reported fault has been rectified. The application may then resume using the framework.

Parameters

fault : in TpInterfaceFault

Specifies the fault from which the framework has recovered.

Raises

TpGeneralException,TpFWException

Method

svcUnavailableInd()

The framework invokes this method to inform the client application that it can no longer use the indicated service. On receipt of this request, the client application must act to reset its use of the specified service (using the normal mechanisms, such as the discovery and authentication interfaces, to stop use of this service instance and begin use of a different service instance).

Parameters

serviceId : in TpServiceID

Identifies the affected service.

reason : in TpSvcUnavailReason

Identifies the reason why the service is no longer available

Raises

TpGeneralException,TpFWException

Method

genFaultStatsRecordRes()

This method is used by the framework to provide fault statistics to a client application in response to a genFaultStatsRecordReq method invocation on the IpFaultManager interface.

Parameters

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

serviceIDs : in TpServiceIDList

Specifies the framework and/or services that are included in the general fault statistics record. The framework is designated by a null value.

Raises

TpGeneralException,TpFWException

Method

fwUnavailableInd()

The framework invokes this method to inform the client application that it is no longer available.

Parameters

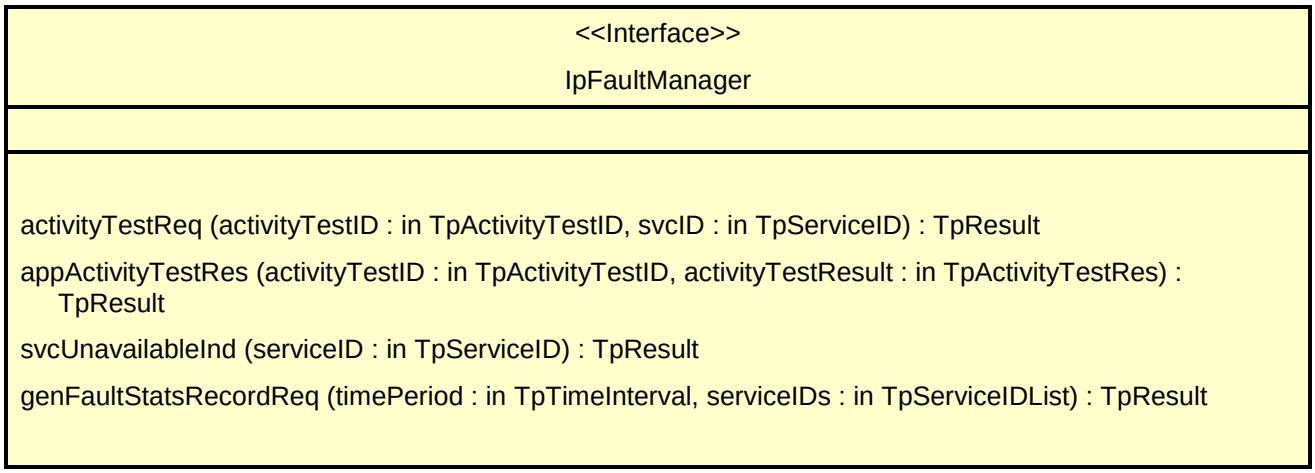
reason : in TpFwUnavailReason

Identifies the reason why the framework is no longer available

8.3.2 Interface Class IpFaultManager

Inherits from: IpInterface.

This interface is used by the application to inform the framework of events that affect the integrity of the framework and services, and to request information about the integrity of the system. The fault manager operations do not exchange callback interfaces as it is assumed that the client application supplies its Fault Management callback interface at the time it obtains the Framework's Fault Management interface, by use of the obtainInterfaceWithCallback operation on the IpAccess interface.



Method

activityTestReq()

The application invokes this method to test that the framework or a service is operational. On receipt of this request, the framework must carry out a test on itself or on the specified service, to check that it is operating correctly. The framework reports the test result by invoking the activityTestRes method on the IpAppFaultManager interface.

Parameters

activityTestID : in TpActivityTestID

The identifier provided by the client application to correlate the response (when it arrives) with this request.

svcID : in TpServiceID

Identifies either the framework or a service for testing. The framework is designated by a null value.

Raises

TpGeneralException,TpFWException

Method

appActivityTestRes()

The client application uses this method to return the result of a framework-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the framework to correlate this response (when it arrives) with the original request.

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpGeneralException,TpFWException

Method

svcUnavailableInd()

This method is used by the client application to inform the framework that it can no longer use the indicated service (either due to a failure in the client application or in the service). On receipt of this request, the framework should take the appropriate corrective action. The framework assumes that the session between this client application and service instance is to be closed and updates its own records appropriately as well as attempting to inform the service instance and/or its administrator. Attempts by the client application to continue using this session should be rejected.

Parameters

serviceID : in TpServiceID

Identifies the service that the application can no longer use.

Raises

TpGeneralException,TpFWException

Method

genFaultStatsRecordReq()

This method is used by the application to solicit fault statistics from the framework. On receipt of this request the framework must produce a fault statistics record, for the framework and/or for specified services during the specified time interval, which is returned to the client application using the genFaultStatsRecordRes operation on the IpAppFaultManager interface.

Parameters

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. A null value leaves this to the discretion of the framework.

serviceIDs : in TpServiceIDList

Specifies the framework and/or services to be included in the general fault statistics record. The framework is designated by a null value.

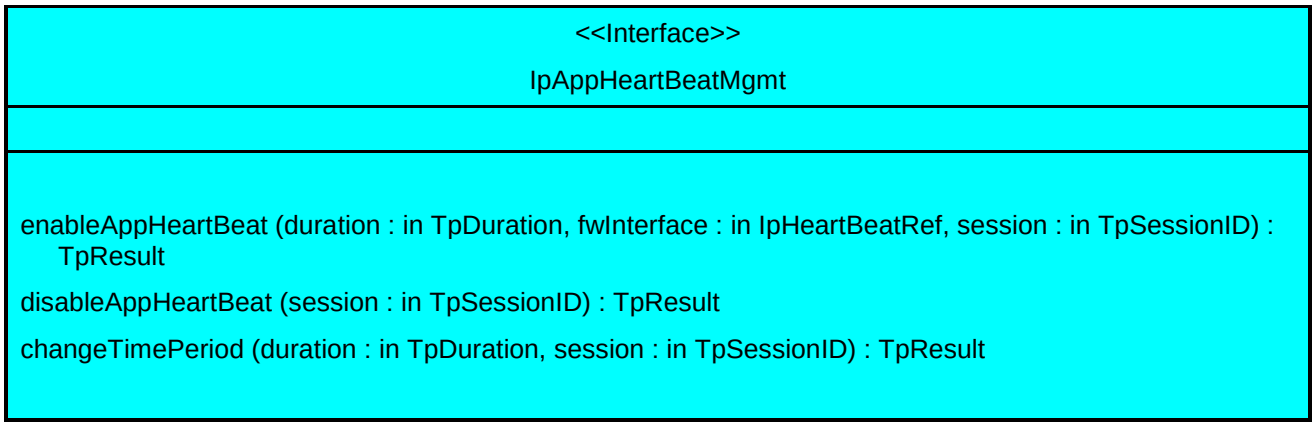
Raises

TpGeneralException,TpFWException

8.3.3 Interface Class IpAppHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the Framework by the Client application. Since the OSA APIs are inherently synchronous, the heartbeats themselves are synchronous for efficiency reasons. The return of the TpResult is interpreted as a heartbeat response.



Method

enableAppHeartBeat()

With this method, the framework registers at the client application for heartbeat supervision of itself.

Parameters

duration : in TpDuration

The time interval in milliseconds between the heartbeats.

fwInterface : in IpHeartBeatRef

This parameter refers to the callback interface the heartbeat is calling.

session : in TpSessionID

Identifies the heartbeat session.

Raises

TpGeneralException,TpFWException

Method

disableAppHeartBeat()

Allows the stop of the heartbeat supervision of the application.

Parameters

session : in TpSessionID

Identifies the heartbeat session.

Raises

TpGeneralException,TpFWException

Method

changeTimePeriod()

Allows the administrative change of the heartbeat period.

Parameters

duration : in TpDuration

The time interval in milliseconds between the heartbeats.

session : in TpSessionID

Identifies the heartbeat session.

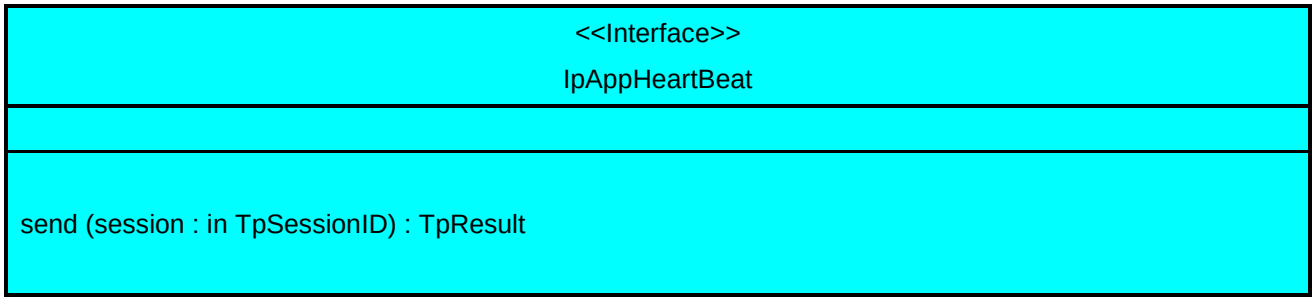
Raises

TpGeneralException,TpFWException

8.3.4 Interface Class IpAppHeartBeat

Inherits from: IpInterface.

The Heartbeat Application interface is used by the Framework to supervise the Application. The return of the TpResult is interpreted as a heartbeat response.



Method

send()

This is the method the framework uses in case it supervises the client application. The sender must raise an exception if no result comes back after a certain, user-defined time..

Parameters

session : in TpSessionID

Identifies the heartbeat session.

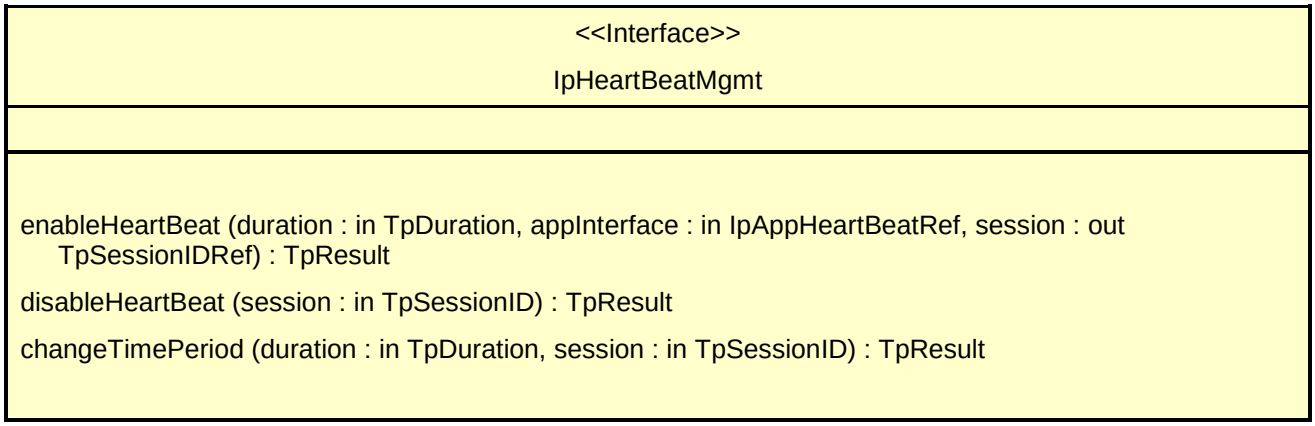
Raises

TpGeneralException,TpFWException

8.3.5 Interface Class IpHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the client application. Since the APIs are inherently synchronous, the heartbeats themselves are synchronous for efficiency reasons. The return of the TpResult is interpreted as a heartbeat response.



Method

enableHeartBeat()

With this method, the client application registers at the framework for heartbeat supervision of itself.

Parameters

duration : in TpDuration

The duration in milliseconds between the heartbeats.

appInterface : in IpAppHeartBeatRef

This parameter refers to the callback interface the heartbeat is calling.

session : out TpSessionIDRef

Identifies the heartbeat session. In general, the application has only one session. In case of framework supervision by the client application (see the application interfaces), the application may maintain more than one session.

Raises

TpGeneralException,TpFWException

Method

disableHeartBeat()

Allows the stop of the heartbeat supervision of the application.

Parameters

session : in TpSessionID

Identifies the heartbeat session.

Raises

TpGeneralException,TpFWException

Method

changeTimePeriod()

Allows the administrative change of the heartbeat period.

Parameters

duration : in TpDuration

The time interval in milliseconds between the heartbeats.

session : in TpSessionID

Identifies the heartbeat session.

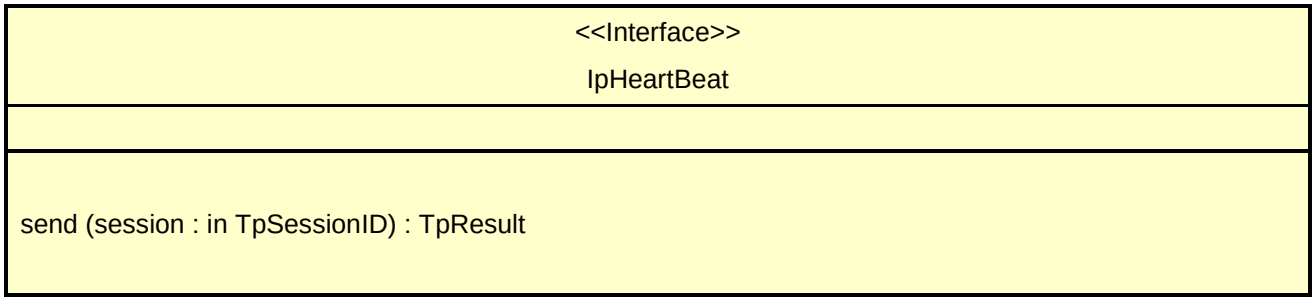
Raises

TpGeneralException,TpFWException

8.3.6 Interface Class IpHeartBeat

Inherits from: IpInterface.

The Heartbeat Framework interface is used by the client application to supervise the Framework.



Method

send()

This is the method the client application uses in case it supervises the framework. The sender must raise an exception if no result comes back after a certain, user-defined time.

Parameters

session : in TpSessionID

Identifies the heartbeat session. In general, the application has only one session.

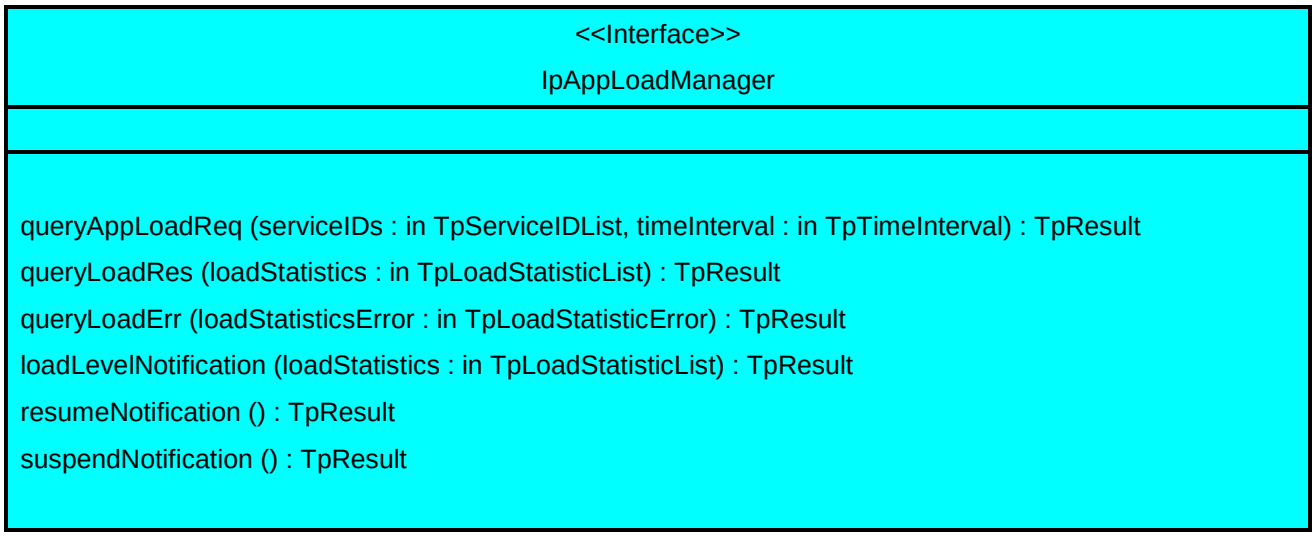
Raises

TpGeneralException,TpFWException

8.3.7 Interface Class IpAppLoadManager

Inherits from: IpInterface.

The client application developer supplies the load manager application interface to handle requests, reports and other responses from the framework load manager function. The application supplies the identity of this callback interface at the time it obtains the framework's load manager interface, by use of the obtainInterfaceWithCallback() method on the IpAccess interface.



Method

queryAppLoadReq()

The framework uses this method to request the application to provide load statistic records for the application and/or for individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

Specifies the application and/or the services for which load statistic records should be reported. The application is designated by a null value.

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistic records should be reported.

Raises

TpGeneralException,TpFWException

Method

queryLoadRes()

The framework uses this method to send load statistic records back to the application that requested the information; i.e. in response to an invocation of the queryLoadReq method on the IpLoadManager interface.

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the framework-supplied load statistics

Raises

TpGeneralException,TpFWException

Method

queryLoadErr()

The framework uses this method to return an error response to the application that requested the framework's load statistics information, when the framework is unsuccessful in obtaining any load statistic records; i.e. in response to an invocation of the queryLoadReq method on the IpLoadManager interface.

Parameters

loadStatisticsError : in TpLoadStatisticError

Specifies the error code associated with the failed attempt to retrieve the framework's load statistics.

Raises

TpGeneralException,TpFWException

Method

loadLevelNotification()

Upon detecting load condition change, (e.g. load level changing from 0 to 1, 0 to 2, 1 to 0, for the SCFs or framework which have been registered for load level notifications) this method is invoked on the application.

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the framework-supplied load statistics, which include the load level change(s).

Raises

TpGeneralException,TpFWException

Method

resumeNotification()

The framework uses this method to request the application to resume sending it notifications: e.g. after a period of suspension during which the framework handled a temporary overload condition.

Parameters
No Parameters were identified for this method

Raises

TpGeneralException,TpFWException

Method
suspendNotification()

The framework uses this method to request the application to suspend sending it any notifications: e.g. while the framework handles a temporary overload condition.

Parameters
No Parameters were identified for this method

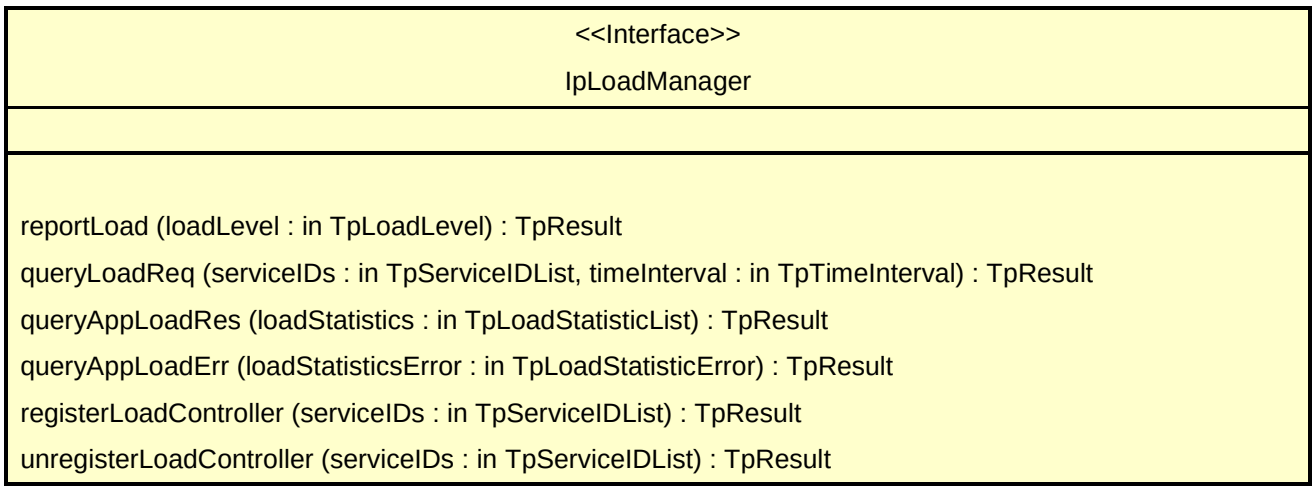
Raises

TpGeneralException,TpFWException

8.3.8 Interface Class IpLoadManager

Inherits from: IpInterface.

The framework API should allow the load to be distributed across multiple machines and across multiple component processes, according to a load management policy. The separation of the load management mechanism and load management policy ensures the flexibility of the load management services. The load management policy identifies what load management rules the framework should follow for the specific client application. It might specify what action the framework should take as the congestion level changes. For example, some real-time critical applications will want to make sure continuous service is maintained, below a given congestion level, at all costs, whereas other services will be satisfied with disconnecting and trying again later if the congestion level rises. Clearly, the load management policy is related to the QoS level to which the application is subscribed. The framework load management function is represented by the IpLoadManager interface. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. To handle responses and reports, the client application developer must implement the IpAppLoadManager interface to provide the callback mechanism. The application supplies the identity of this callback interface at the time it obtains the framework's load manager interface, by use of the obtainInterfaceWithCallback operation on the IpAccess interface.



resumeNotification (serviceIDs : in TpServiceIDList) : TpResult

suspendNotification (serviceIDs : in TpServiceIDList) : TpResult

Method

reportLoad()

The client application uses this method to report its current load level (0,1, or 2) to the framework: e.g. when the load level on the application has changed.

At level 0 load, the application is performing within its load specifications (i.e. it is not congested or overloaded). At level 1 load, the application is overloaded. At level 2 load, the application is severely overloaded.

Parameters

loadLevel : in TpLoadLevel

Specifies the application's load level.

Raises

TpGeneralException,TpFWException

Method

queryLoadReq()

The client application uses this method to request the framework to provide load statistic records for the framework and/or for individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and/or the services for which load statistic records should be reported. The framework is designated by a null value.

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistic records should be reported.

Raises

TpGeneralException,TpFWException

Method

queryAppLoadRes()

The client application uses this method to send load statistic records back to the framework that requested the information; i.e. in response to an invocation of the queryAppLoadReq method on the IpAppLoadManager interface.

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the application-supplied load statistics.

Raises

TpGeneralException,TpFWException

Method

queryAppLoadErr()

The client application uses this method to return an error response to the framework that requested the application's load statistics information, when the application is unsuccessful in obtaining any load statistic records; i.e. in response to an invocation of the queryAppLoadReq method on the IpAppLoadManager interface.

Parameters

loadStatisticsError : in TpLoadStatisticError

Specifies the error code associated with the failed attempt to retrieve the application's load statistics.

Raises

TpGeneralException,TpFWException

Method

registerLoadController()

The client application uses this method to register to receive notifications of load level changes associated with the framework and/or with individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and SCFs to be registered for load control. To register for framework load control only, the serviceIDs is null.

Raises

TpGeneralException,TpFWException

Method

unregisterLoadController()

The client application uses this method to unregister for notifications of load level changes associated with the framework and/or with individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and/or the services for which load level changes should no longer be reported. The framework is designated by a null value.

Raises

TpGeneralException,TpFWException

Method

resumeNotification()

The client application uses this method to request the framework to resume sending it load management notifications associated with the framework and/or with individual services used by the application; e.g. after a period of suspension during which the application handled a temporary overload condition.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and/or the services for which the sending of notifications of load level changes by the framework should be resumed. The framework is designated by a null value.

Raises

TpGeneralException,TpFWException

Method

suspendNotification()

The client application uses this method to request the framework to suspend sending it load management notifications associated with the framework and/or with individual services used by the application; e.g. while the application handles a temporary overload condition.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and/or the services for which the sending of notifications by the framework should be suspended. The framework is designated by a null value

Raises

TpGeneralException,TpFWException

8.3.9 Interface Class IpOAM

Inherits from: IpInterface.

The OAM interface is used to query the system date and time. The application and the framework can synchronise the date and time to a certain extent. Accurate time synchronisation is outside the scope of of the OSA APIs.

<<Interface>> IpOAM
systemDateTimeQuery (clientDateAndTime : in TpDateAndTime, systemDateAndTime : out TpDateAndTimeRef) : TpResult

Method

`systemDateTimeQuery()`

This method is used to query the system date and time. The client application passes in its own date and time to the framework. The framework responds with the system date and time.

Parameters

`clientDateAndTime : in TpDateAndTime`

This is the date and time of the client (application). The error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

`systemDateAndTime : out TpDateAndTimeRef`

This is the system date and time of the framework.

Raises

`TpGeneralException,TpFWException`

8.3.10 Interface Class IpAppOAM

Inherits from: IpInterface.

The OAM client application interface is used by the Framework to query the application date and time, for synchronisation purposes.This method is invoked by the Framework to interchange the framework and client application date and time.

<<Interface>> IpAppOAM
systemDateTimeQuery (systemDateAndTime : in TpDateAndTime, clientDateAndTime : out TpDateAndTimeRef) : TpResult

Method

`systemDateTimeQuery()`

This method is used to query the system date and time. The framework passes in its own date and time to the application. The application responds with its own date and time.

Parameters

systemDateAndTime : in TpDateAndTime

This is the system date and time of the framework.

clientDateAndTime : out TpDateAndTimeRef

This is the date and time of the client (application). The error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

Raises

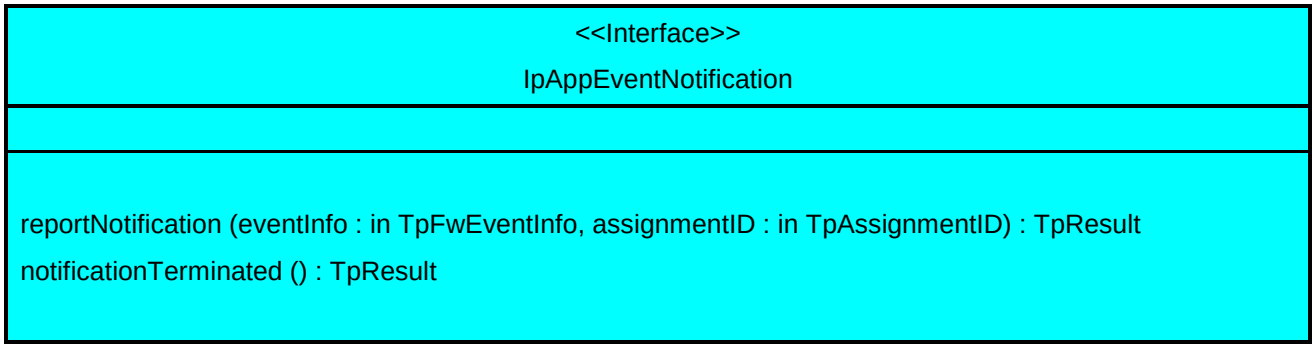
TpGeneralException,TpFWException

8.4 Event Notification Interface Classes

8.4.1 Interface Class IpAppEventNotification

Inherits from: IpInterface.

This interface is used by the services to inform the application of a generic service-related event. The Event Notification Framework will invoke methods on the Event Notification Application Interface that is specified when the Event Notification interface is obtained.



Method

reportNotification()

This method notifies the application of the arrival of a generic event.

Parameters

eventInfo : in TpFwEventInfo

Specifies specific data associated with this event.

assignmentID : in TpAssignmentID

Specifies the assignment id which was returned by the framework during the createNotification() method. The application can use assignment id to associate events with event specific criteria and to act accordingly.

Raises

TpGeneralException,TpFWException

Method
notificationTerminated()

This method indicates to the application that all generic event notifications have been terminated (for example, due to faults detected).

Parameters
No Parameters were identified for this method

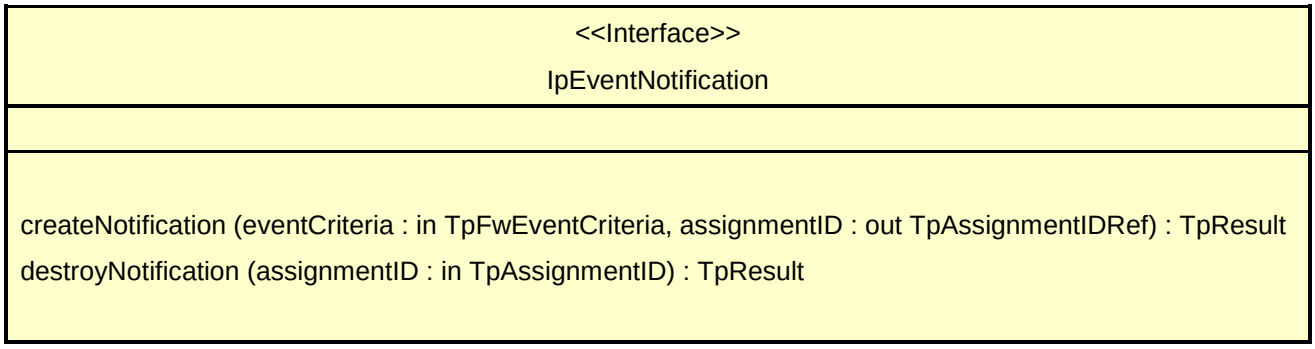
Raises

TpGeneralException,TpFWException

8.4.2 Interface Class IpEventNotification

Inherits from: IpInterface.

The event notification mechanism is used to notify the application of generic service related events that have occurred.



Method
createNotification()

This method is used to enable generic notifications so that events can be sent to the application.

Parameters

eventCriteria : in TpFwEventCriteria

Specifies the event specific criteria used by the application to define the event required.

assignmentID : out TpAssignmentIDRef

Specifies the ID assigned by the framework for this newly installed notification.

Raises

TpGeneralException,TpFWException

Method
destroyNotification()

This method is used by the application to delete generic notifications from the framework.

Parameters

`assignmentID : in TpAssignmentID`

Specifies the assignment ID given by the framework when the previous createNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will return the error code P_INVALID_ASSIGNMENTID.

Raises

`TpGeneralException,TpFWException`

9 Framework-to-Application State Transition Diagrams

This section contains the State Transition Diagrams for the objects that implement the Framework interfaces on the gateway side. The State Transition Diagrams show the behaviour of these objects. For each state the methods that can be invoked by the application are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the application also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

9.1 Trust and Security Management State Transition Diagrams

9.1.1 State Transition Diagrams for IpInitial

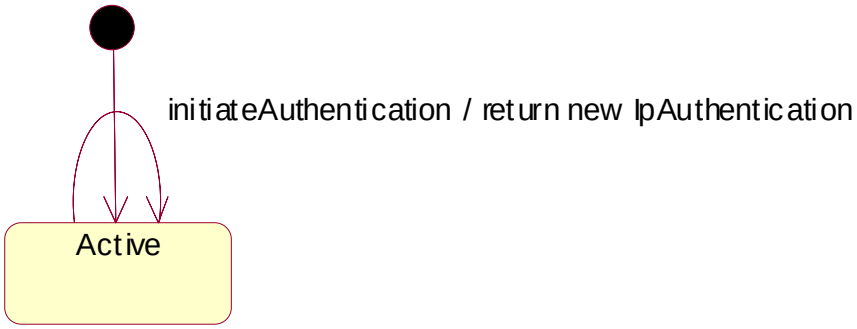


Figure : State Transition Diagram for IpInitial

9.1.1.1 Active State

9.1.2 State Transition Diagrams for IpAPILevelAuthentication

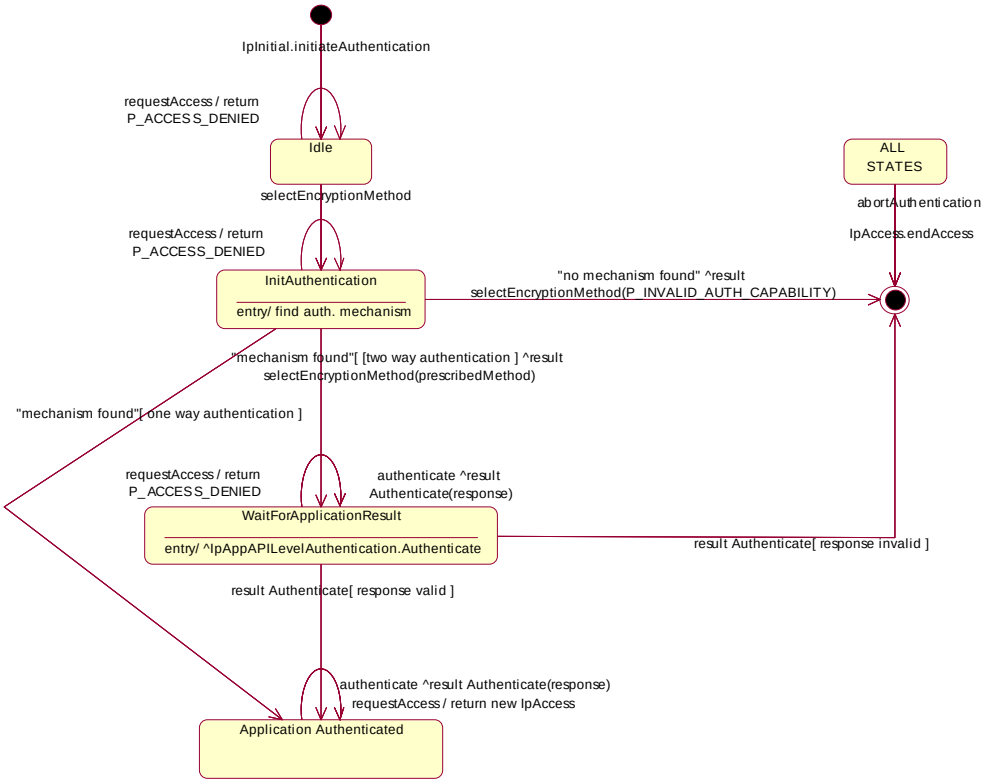


Figure : State Transition Diagram for IpAPILevelAuthentication

9.1.2.1 Idle State

When the application has requested the IpInitial interface for initiateAuthentication, an object implementing the IpAPILevelAuthentication interface is created. The application now has to provide its authentication capabilities by invoking the SelectEncryptionMethod method.

9.1.2.2 InitAuthentication State

In this state the Framework selects the preferred authentication mechanism within the capability of the application. When a proper mechanism is found, the Framework can decide that the application doesn't have to be authenticated (one way authentication) or that the application has to be authenticated. In case no mechanism can be found the error code P_INVALID_AUTH_CAPABILITY is returned and the Authentication object is destroyed. This implies that the application has to re-initiate the authentication by calling once more the initiateAuthentication method on the IpInitial interface.

9.1.2.3 WaitForApplicationResult State

When entering this state, the Framework requests the application to authenticate itself by invoking the Authenticate method on the application. In case the application requests the Framework to authenticate itself by invoking Authenticate on the IpAPILevelAuthentication interface, the Framework provides the correct response to the challenge of the application. When the Framework responds to the Authenticate request, the response is analysed and in case the response is valid a transition to the state Application Authenticated is made. In case the response is not valid, the Authentication object is destroyed. This implicates that the application has to re-initiate the authentication by calling once more the initiateAuthentication method on the IpInitial interface.

9.1.2.4 Application Authenticated State

In this state the application is considered authenticated and is now allowed to request access to the IpAccess interface. In case the application requests the Framework to authenticate itself by invoking Authenticate on the IpAPILevelAuthentication interface, the Framework provides the correct response to the challenge of the application.

9.1.3 State Transition Diagrams for IpAccess

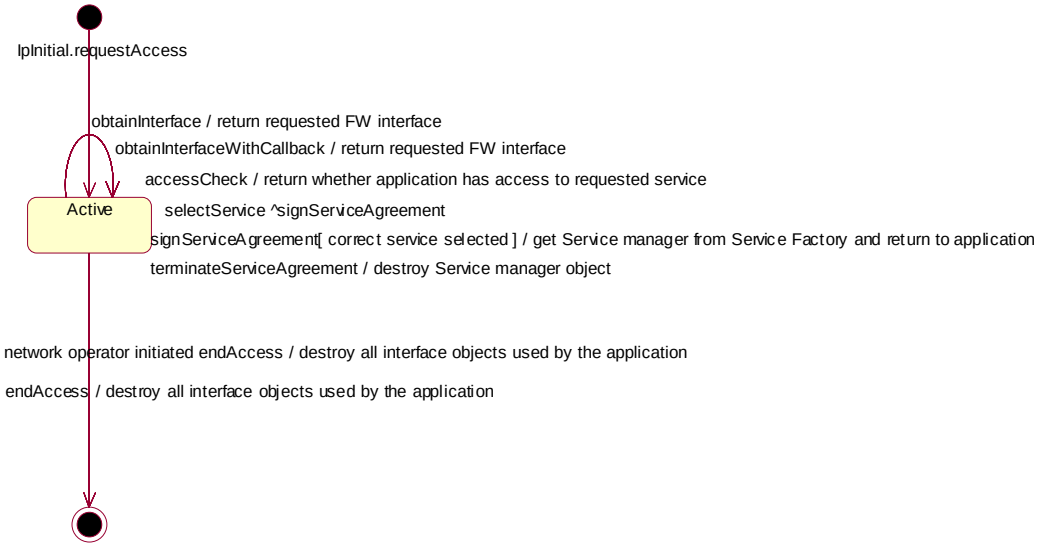


Figure : State Transition Diagram for IpAccess

9.1.3.1 Active State

When the application requestes access to the Framework on the IpInitial interface, an object implementing the IpAccess interface is created. The application can now request other Framework interfaces, including Service Discovery. When the application is no longer interested in using the interfaces it calls the endAccess method. This results in the destruction of all interface objects used by the application. In case the network operator decides that the application has no longer access to the interfaces the same will happen.

9.2 Service Discovery State Transition Diagrams

9.2.1 State Transition Diagrams for IpServiceDiscovery

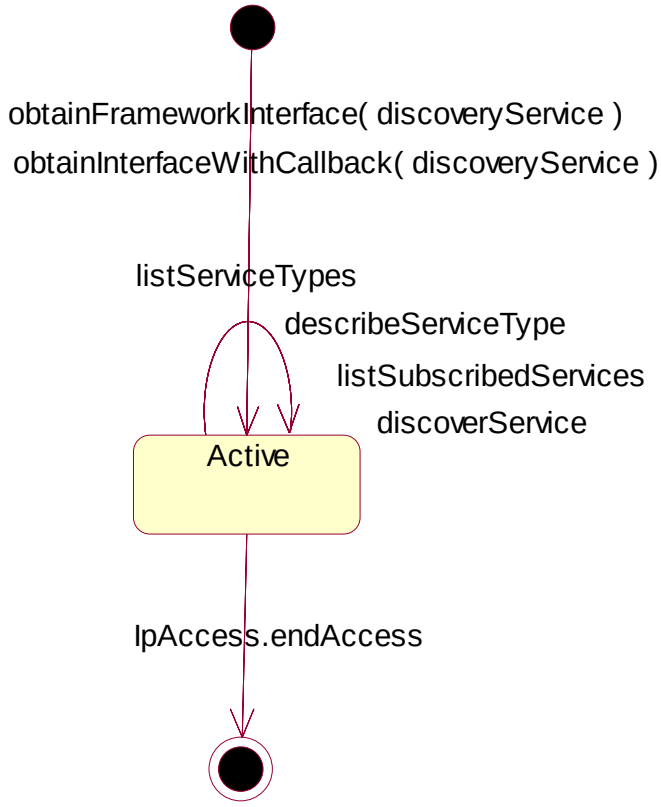


Figure : State Transition Diagram for IpServiceDiscovery

9.2.1.1 Active State

When the application requests Service Discovery by invoking the obtainInterface or the obtainInterfaceWithCallback methods on the IpAccess interface, an instance of the IpServiceDiscovery will be created. Next the application is allowed to request a list of the provided SCFs and to obtain a reference to interfaces of SCFs.

9.3 Integrity Management State Transition Diagrams

9.3.1 State Transition Diagrams for IpHeartBeatMgmt

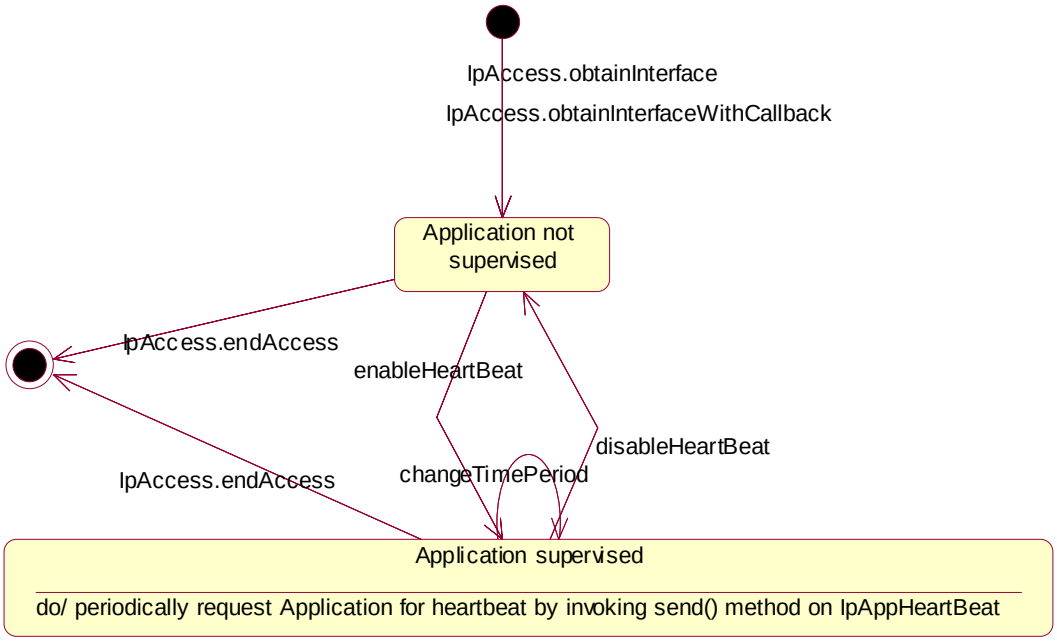


Figure : State Transition Diagram for IpHeartBeatMgmg

9.3.1.1 Application not supervised State

In this state the application has not registered for heartbeat supervision by the Framework.

9.3.1.2 Application supervised State

In this state the application has registered for heartbeat supervision by the Framework. Periodically the Framework will request for the application heartbeat by calling the send method on the IpAppHeartBeat interface.

9.3.2 State Transition Diagrams for IpHeartBeat

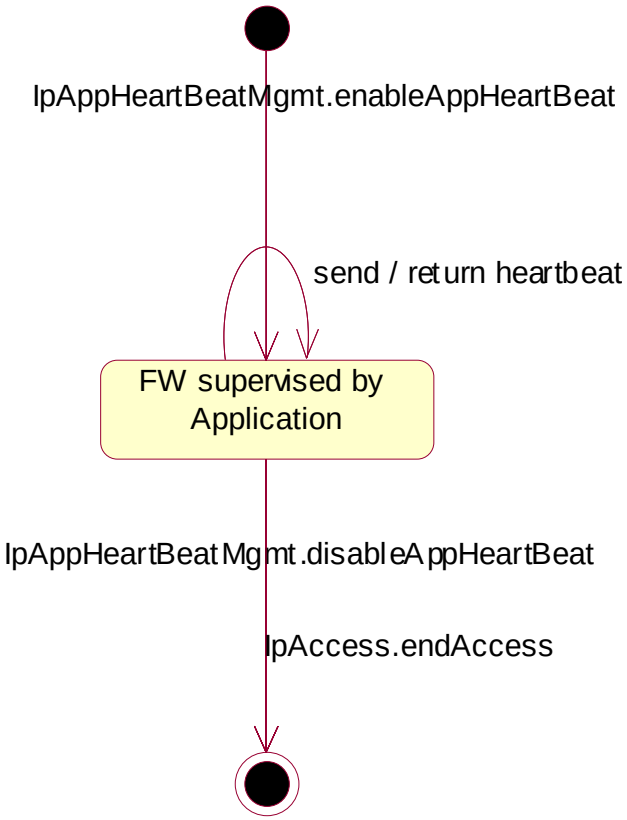


Figure : State Transition Diagram for IpHeatBeat

9.3.2.1 FW supervised by Application State

In this state the Framework has requested the application for heartbeat supervision on itself. Periodically the application calls the send() method and the Framework returns it's heartbeat result.

9.3.3 State Transition Diagrams for IpLoadManager

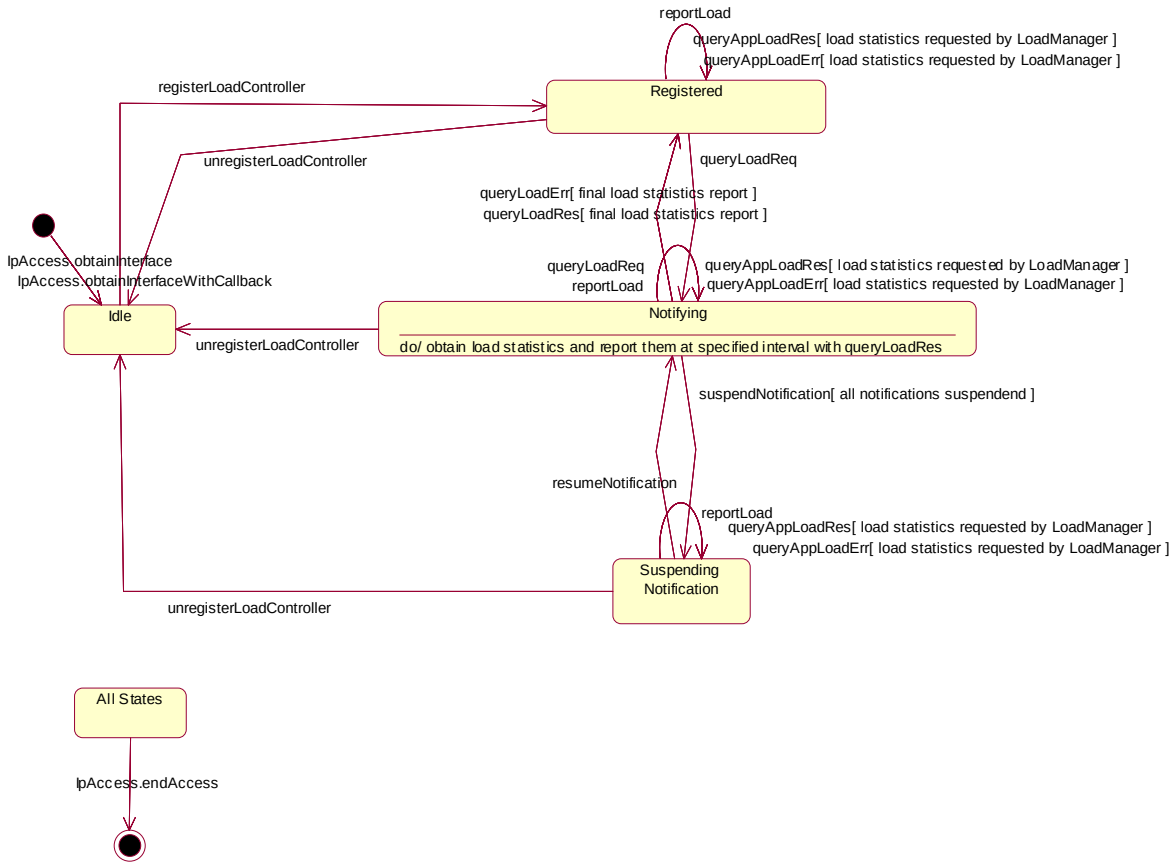


Figure : State Transition Diagram for IpLoadManager

9.3.3.1 Idle State

In this state the application has obtained an interface reference of the LoadManager from the IpAccess interface.

9.3.3.2 Notifying State

In the Notifying state the application has requested for load statistics. The Loadmanager gathers the requested information and (periodically) reports them to the application.

9.3.3.3 Suspending Notification State

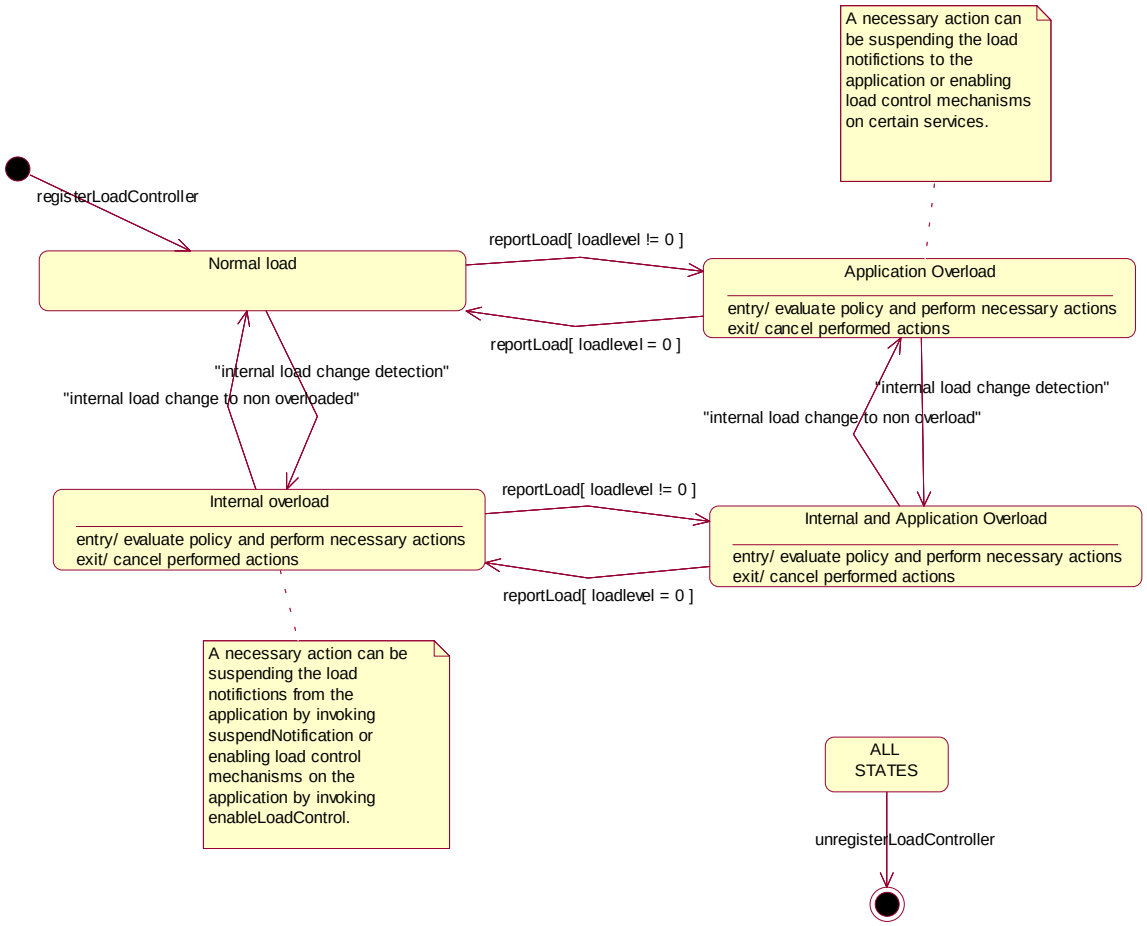
Due to e.g. a temporary load condition, the application has requested the LoadManager to suspend sending the load statistics information.

9.3.3.4 Registered State

In this state the application has registered for load control with the method RegisterLoadController(). The LoadManager can now request the application to supply load statistics information (by invoking queryAppLoadReq()). Furthermore the LoadManager can request the application to control its load (by invoking loadLevelNotification() or suspendNotification() on the application side of interface). In case the application detects a change in load level, it reports this to the LoadManager by calling the method reportLoad().

When entering this state, an object called LoadManagerInternal is created that has an internal state machine encapsulating the internal behaviour of the LoadManager. The State Transition Diagram of LoadManagerInternal is shown in Figure .

9.3.4 State Transition Diagrams for IpLoadManagerInternal



igure : State Transition Diagram for IpLoadManagerInternal

9.3.4.1 Normal load State

In this state the none of the entities defined in the load balancing policy between the application and the framework / SCFs is overloaded.

9.3.4.2 Application Overload State

In this state the application has indicated it is overloaded. When entering this state the load policy is consulted and the appropriate actions are taken by the LoadManager.

9.3.4.3 Internal overload State

In this state the Framework or one or more of the SCFs within the specific load policy is overloaded. When entering this state the load policy is consulted and the appropriate actions are taken by the LoadManager.

9.3.4.4 Internal and Application Overload State

In this state the application is overloaded as well as the Framework or one or more of the SCFs within the specific load policy. When entering this state the load policy is consulted and the appropriate actions are taken by the LoadManager.

9.3.5 State Transition Diagrams for IpOAM

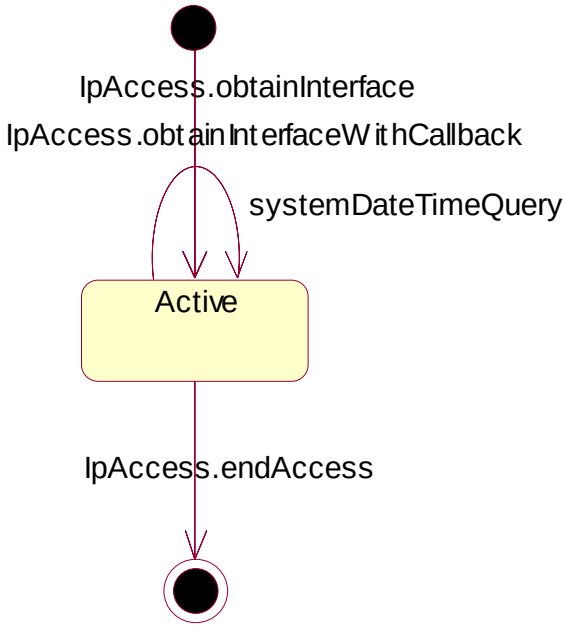


Figure : State Transition Diagram for IpOAM

9.3.5.1 Active State

In this state the application has obtained a reference to the IpOAM interface. The application is now able to request the date / time of the Framework.

9.3.6 State Transition Diagrams for IpFaultManager

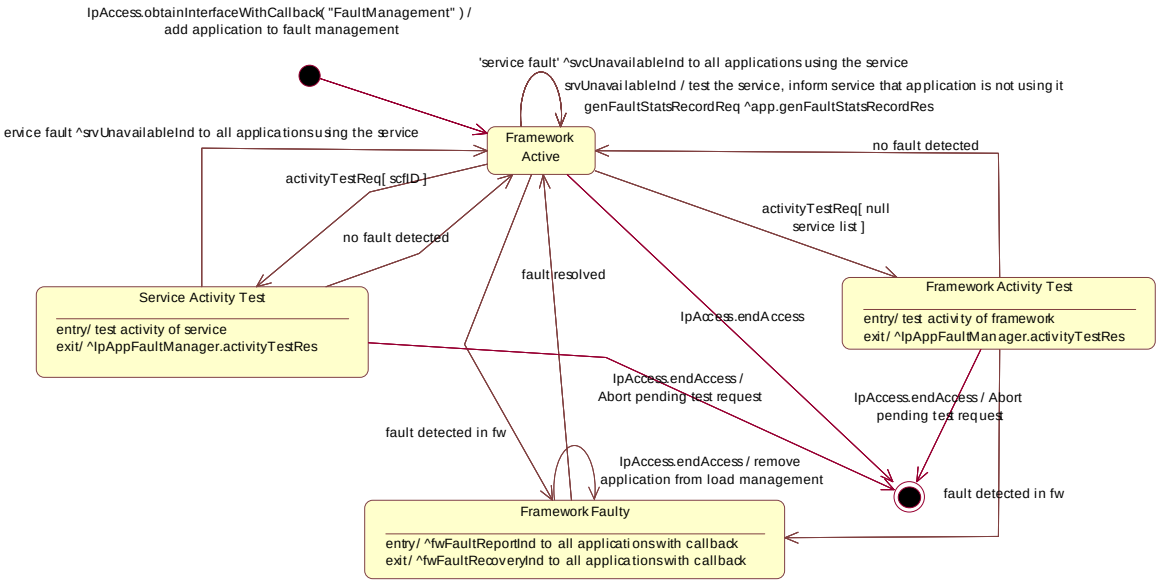


Figure : State Transition Diagram for IpFaultManager

9.3.6.1 Framework Active State

This is the normal state of the framework, which is fully functional and able to handle requests from both applications and services capability features.

9.3.6.2 Framework Faulty State

In this state, the framework has detected an internal problem with itself such that application and services capability features cannot communicate with it anymore; attempts to invoke any methods that belong to any SCFs of the framework return an error. If the framework ever recovers, applications with fault management callbacks will be notified via a fwFaultRecoveryInd message.

9.3.6.3 Framework Activity Test State

In this state, the framework is performing self-diagnostic test. If a problem is diagnosed, all applications with fault management callbacks are notified through a fwFaultReportInd message.

9.3.6.4 Service Activity Test State

In this state, the framework is performing a test on one service capability feature. If the SCF is faulty, applications with fault management callbacks are notified accordingly through a svcUnavailableInd message.

9.4 Event Notification State Transition Diagrams

9.4.1 State Transition Diagrams for IpEventNotification

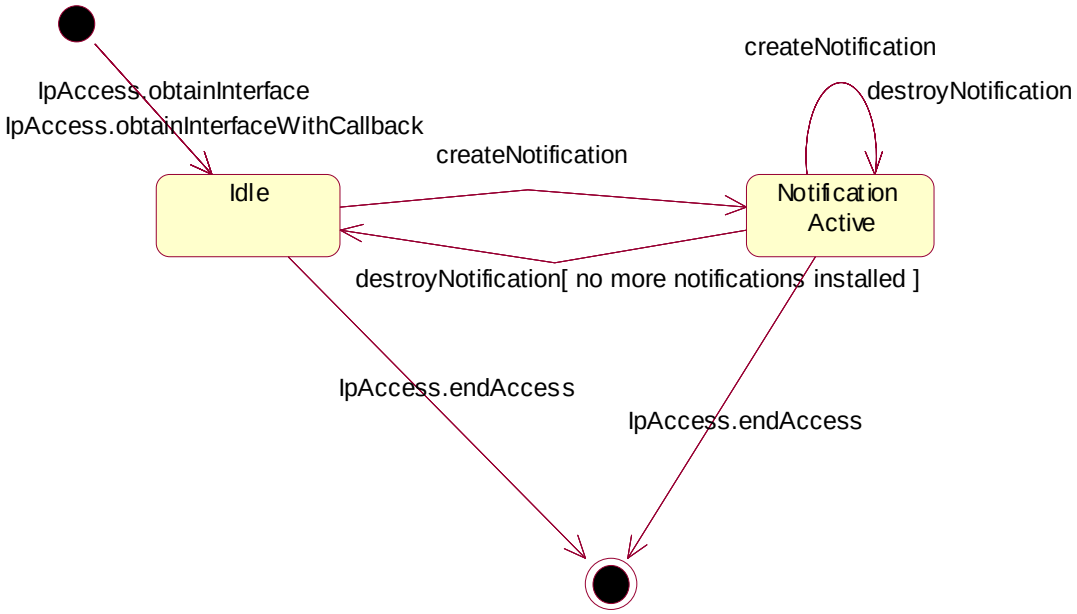


Figure : State Transition Diagram for IpEventNotification

9.4.1.1 Idle State

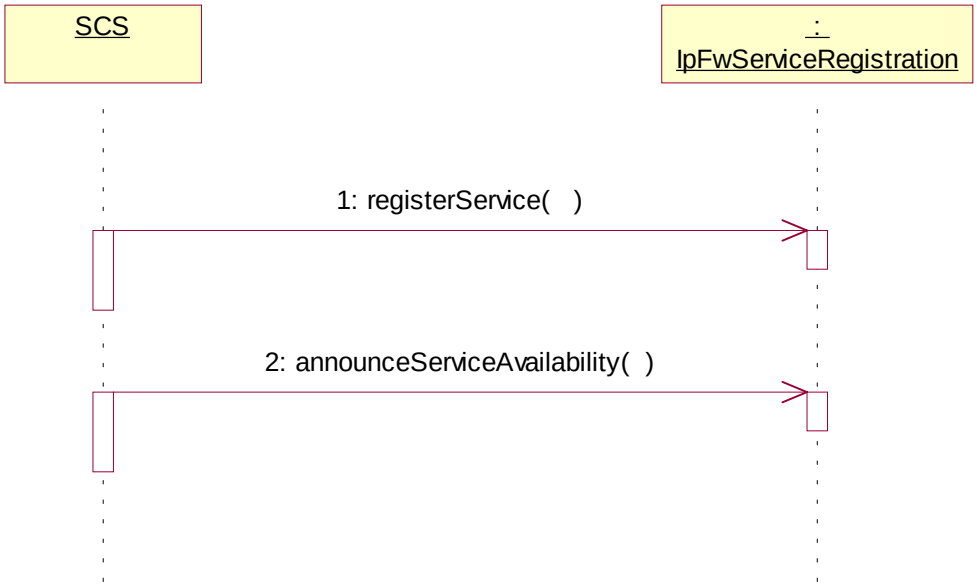
9.4.1.2 Notification Active State

10 Framework-to-Service Sequence Diagrams

10.1 Service Registration Sequence Diagrams

10.1.1 New SCF Registration

The following figure shows the process of registering a new Service Capability Feature in the Framework. Service Registration is a two step process:



1: Registration: first step - register service

The purpose of this first step in the process of registration is to agree, within the network, on a name to call, internally, a newly installed SCF version. It is necessary because the OSA Framework and SCF in the same network may come from different vendors. The goal is to make an association between the new SCF version, as characterized by a list of properties, and an identifier called serviceID.

This service ID will be the name used in that network (that is, between that network's Framework and its SCSs), whenever it is necessary to refer to this newly installed version of SCF (for example for announcing its availability, or for withdrawing it later).

The following input parameters are given from the SCS to the Framework in this first registration step:

· in serviceTypeName

This is a string with the name of the SCF, among a list of standard names (e.g. "P_MPCC").

· in servicePropertyList

This is a list of types TpServiceProperty; each TpServiceProperty is a triplet (ServicePropertyName, ServicePropertyValueList, ServicePropertyMode).

· ServicePropertyName is a string that defines a valid SFC property name (valid SCF property names are listed in the SCF data definition).

· ServicePropertyValueList is a numbered set of types TpServicePropertyValue; TpServicePropertyValue is a string that describes a valid value of a SCF property (valid SCF property values are listed in the SCF data definition).

· ServicePropertyMode is the value of the property modes (e.g. "mandatory", meaning that all properties of this SCF must be given values at service registration time).

The following output parameter results from service registration:

· out serviceID

This is a string, automatically generated by the Framework of this network, based on the following:

· a string that contains a unique number, generated by the Framework;

· a string that identifies the SCF name (e.g. "P_MPCC");

· a concatenation of strings that identify the SCF specialization, if any.

This is the name by which the newly installed version of SCF, described by the list of properties above, is going to be identified internally in this network.

2: Registration: second step - announce service availability

At this point the network's Framework is aware of the existence of a new SCF, and could let applications know - but they would have no way to use it. Installing the SCS logic and assigning a name to it does not make this SCF available. In CORBA an "entry point", called service factory, is used. The role of the service factory is to control the life cycle of a CORBA interface, or set of interfaces, and provide clients with the references that are necessary to invoke the methods offered by these interfaces. Some times service factories instantiate new interfaces for different clients, sometime they give the same interface reference to more than one client. But the starting point for a client to use an SCF is to obtain an interface reference to a factory of the desired SCF.

A Network Operator, upon completion of the first registration phase, and once it has an identifier to the new SCF version, will instantiate a factory for it that will allow client to use it. Then it will inform the Framework of the value of the interface associated to the new SCF. After the receipt of this information, the Framework makes the new SCF (identified by the pair [serviceID, serviceFactoryRef]) discoverable.

The following input parameters are given from the SCS to the Framework in this second registration step:

· in serviceID

This is the identifier that has been agreed in the network for the new SCF; any interaction related to the SCF needs to include the serviceID, to know which SCF it is.

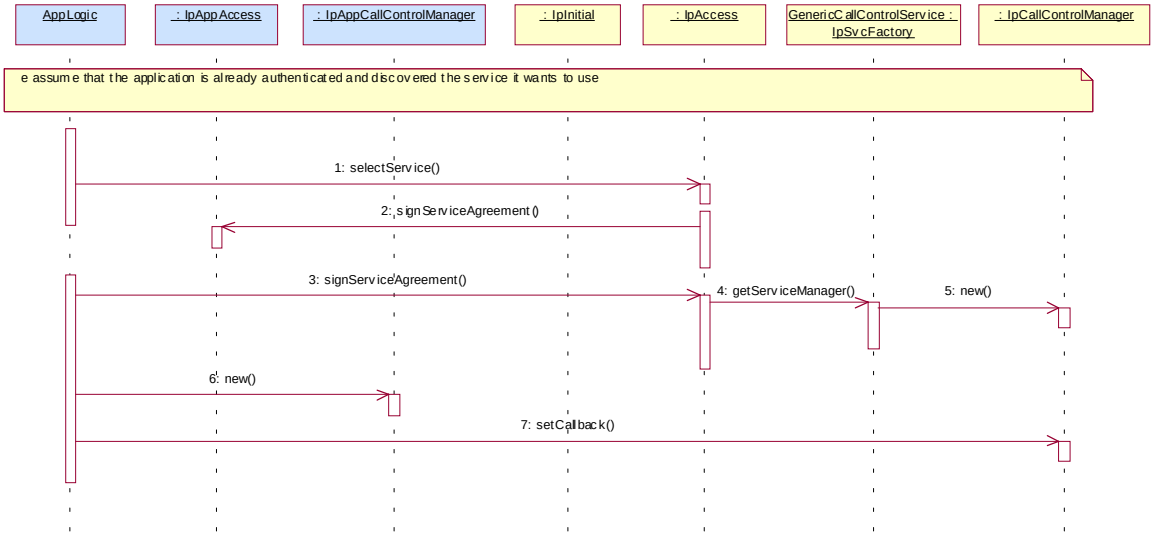
· in serviceFactoryRef

This is the interface reference at which the service factory of the new SCF is available. Note that the Framework will have to invoke the method getServiceManager() in this interface, any time between now and when it accepts the first application requests for discovery, so that it can get the service manager interface necessary for applications as an entry point to any SCF.

10.2 Service Factory Sequence Diagrams

10.2.1 Sign Service Agreement

This sequence illustrates how the application can get access to a specified service. It only illustrates the last part: the signing of the service agreement and the corresponding actions towards the service. For more information on accessing the framework, authentication and discovery of services, see the corresponding sections.



1: The application selects the service, using a serviceID for the generic call control service. The serviceID could have been obtained via the discovery interface. A ServiceToken is returned to the application.

2: The framework signs the service agreement.

3: The client application signs the service agreement. As a result a service manager interface reference (in this case of type IpCallControlManager) is returned to the application.

4: Provided the signature information is correct and all conditions have been fulfilled, the framework will request the service identified by the serviceID to return a service manager interface reference. The service manager is the initial point of contact to the service.

5: The service factory creates a new manager interface instance (a call control manager) for the specified application. It should be noted that this is an implementation detail. The service implementation may use other mechanism to get a service manager interface instance.

6: The application creates a new IpAppCallControlManager interface to be used for callbacks.

7: The Application sets the callback interface to the interface created with the previous message.

11 Framework-to-Service Class Diagrams

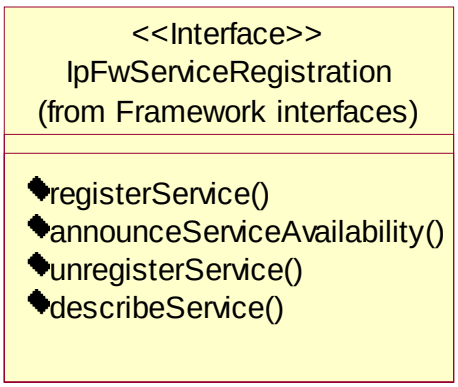


Figure: Service Registration Package Overview

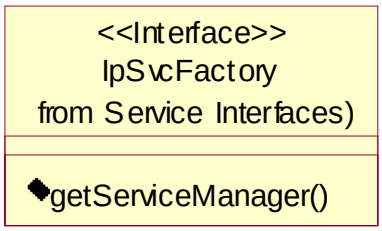


Figure: Service Factory Package Overview

12 Framework-to-Service Interface Classes

12.1 Service Registration Interface Classes

12.1.1 Interface Class IpFwServiceRegistration

Inherits from: IpInterface.

The Service Registration interface provides the methods used for the registration of network SCFs at the framework.

<<Interface>> IpFwServiceRegistration
registerService (serviceTypeName : in TpServiceTypeName, servicePropertyList : in TpServicePropertyList, serviceID : out TpServiceIDRef) : TpResult announceServiceAvailability (serviceID : in TpServiceID, serviceFactoryRef : in IpServiceRef) : TpResult unregisterService (serviceID : in TpServiceID) : TpResult describeService (serviceID : in TpServiceID, serviceDescription : out TpServiceDescriptionRef) : TpResult

Method

registerService()

The registerService() operation is the means by which a service is registered in the Framework, for subsequent discovery by the enterprise applications . A service-ID is returned to the service supplier when a service is registered in the Framework. The service-ID is the handle with which the service supplier can identify the registered service when needed (e.g. for withdrawing it). The service-ID is only meaningful in the context of the Framework that generated it.

Parameters

serviceTypeName : in TpServiceTypeName

The "serviceTypeName" parameter identifies the service type and a set of named property types that may be used in further describing this service (i.e., it restricts what is acceptable in the servicePropertyList parameter). If the string representation of the "type" does not obey the rules for identifiers, then an P_ILLEGAL_SERVICE_TYPE exception is raised. If the "type" is correct syntactically but the Framework is able to unambiguously determine that it is not a recognised service type, then a P_UNKNOWN_SERVICE_TYPE exception is raised.

servicePropertyList : in TpServicePropertyList

The "servicePropertyList" parameter is a list of property name and property value pairs. They describe the service being registered. This description typically covers behavioral, non-functional and non-computational aspects of the service. Service properties are marked "mandatory" or "readonly". These property mode attributes have the following semantics:

a. mandatory - a service associated with this service type must provide an appropriate value for this property when registering.

b. readonly - this modifier indicates that the property is optional, but that once given a value, subsequently it may not be modified.

Specifying both modifiers indicates that a value must be provided and that subsequently it may not be modified. An example of such properties are those which form part of a service agreement and hence cannot be modified by service suppliers during the life time of service.

If the type of any of the property values is not the same as the declared type (declared in the service type), then a P_PROPERTY_TYPE_MISMATCH exception is raised. If an attempt is made to assign a dynamic property value to a readonly property, then the P_READONLY_DYNAMIC_PROPERTY exception is raised. If the "servicePropertyList" parameter omits any property declared in the service type with a mode of mandatory, then a P_MISSING_MANDATORY_PROPERTY exception is raised. If two or more properties with the same property name are included in this parameter, the P_DUPLICATE_PROPERTY_NAME exception is raised.

serviceID : out TpServiceIDRef

This is the unique handle that is returned as a result of the successful completion of this operation. The Service Supplier can identify the registered service when attempting to access it via other operations such as unregisterService(), etc. Enterprise client applications are also returned this service-ID when attempting to discover a service of this type.

Raises

TpGeneralException,TpFWException

Method

announceServiceAvailability()

The registerService() method described previously does not make the service discoverable. The announceServiceAvailability() method is invoked after the service is authenticated and its service factory is instantiated at a particular interface. This method informs the framework of the availability of "service factory" of the previously registered service, identified by its service ID, at a specific interface. After the receipt of this method, the framework makes the corresponding service discoverable.

There exists a "service manager"instance per service instance. Each service implements the IpSvcFactory interface. The IpSvcFactory interface supports a method called the getServiceManager(application: in TpClientAppID, serviceManager: out IpServiceRefRef). When the service agreement is signed for some serviceID (using signServiceAgreement()), the framework calls the getServiceManager() for this service, gets a serviceManager and returns this to the client application.

Parameters

serviceID : in TpServiceID

The service ID of the service that is being announced. If the string representation of the "serviceID" does not obey the rules for service identifiers, then an P_ILLEGAL_SERVICE_ID exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then an P_UNKNOWN_SERVICE_ID exception is raised.

serviceFactoryRef : in IpServiceRef

The interface reference at which the service factory of the previously registered service is available.

Raises

TpGeneralException,TpFWException

Method

unregisterService()

The unregisterService() operation is used by the service suppliers to remove a registered service from the Framework. The service is identified by the "service-ID" which was originally returned by the Framework in response to the registerService() operation. After the unregisterService(), the service can no longer be discovered by the enterprise client application.

Parameters

serviceID : in TpServiceID

The service to be withdrawn is identified by the "serviceID" parameter which was originally returned by the registerService() operation. If the string representation of the "serviceID" does not obey the rules for service identifiers, then an P_ILLEGAL_SERVICE_ID exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then an P_UNKNOWN_SERVICE_ID exception is raised.

Raises

TpGeneralException,TpFWException

Method

describeService()

The describeService() operation returns the information about a service that is registered in the framework. It comprises, the "type" of the service , and the "properties" that describe this service. The service is identified by the "service-ID" parameter which was originally returned by the registerService() operation.

This operation is intended to be used between a certain framework and the SCS that registered the SCF, since it is only between them that the serviceID is valid. The SCS may register various versions of the same SCF, each with a different description (more or less restrictive, for example), and each getting a different serviceID assigned. Getting the description of these SCFs from the framework where they have been registered helps the SCS internal maintenance.

Parameters

serviceID : in TpServiceID

The service to be described is identified by the "serviceID" parameter which was originally returned by the registerService() operation. If the string representation of the "serviceID" does not obey the rules for object identifiers, then an P_ILLEGAL_SERVICE_ID exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then a P_UNKNOWN_SERVICE_ID exception is raised.

serviceDescription : out TpServiceDescriptionRef

This consists of the information about an offered service that is held by the Framework. It comprises the "type" of the service , and the properties that describe this service.

Raises

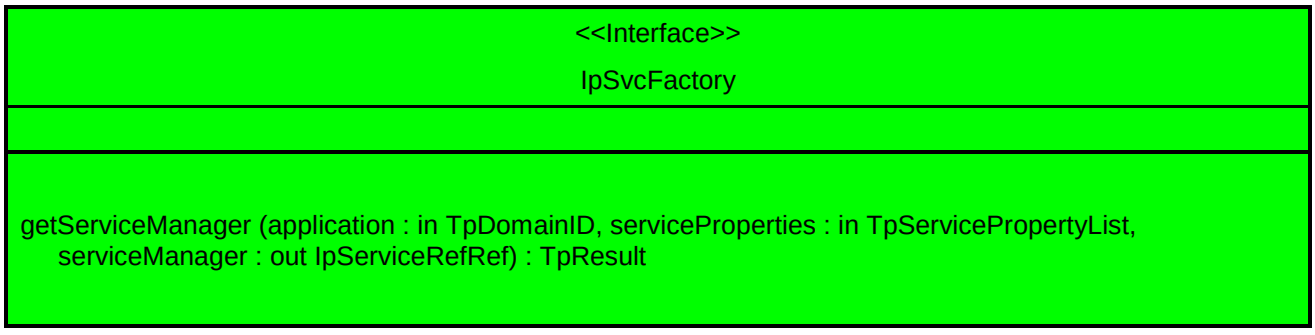
TpGeneralException,TpFWException

12.2 Service Factory Interface Classes

12.2.1 Interface Class IpSvcFactory

Inherits from: IpInterface.

The IpSvcFactory interface allows the framework to get access to a service manager interface of a service. It is used during the signServiceAgreement, in order to return a service manager interface reference to the application. Each service has a service manager interface that is the initial point of contact for the service. E.g., the generic call control service uses the IpCallControlManager interface.



Method

getServiceManager()

This method returns a service manager interface reference for the specified application. Usually, but not necessarily, this involves the instantiation of a new service manager interface.

Parameters

application : in TpDomainID

Specifies the application for which the service manager interface is requested.

serviceProperties : in TpServicePropertyList

serviceManager : out IpServiceRefRef

Specifies the service manager interface reference for the specified application ID.

Raises

TpGeneralException,TpFWException

13 Framework-to-Service State Transition Diagrams

13.1 Service Registration State Transition Diagrams

13.1.1 State Transition Diagrams for IpFwServiceRegistration

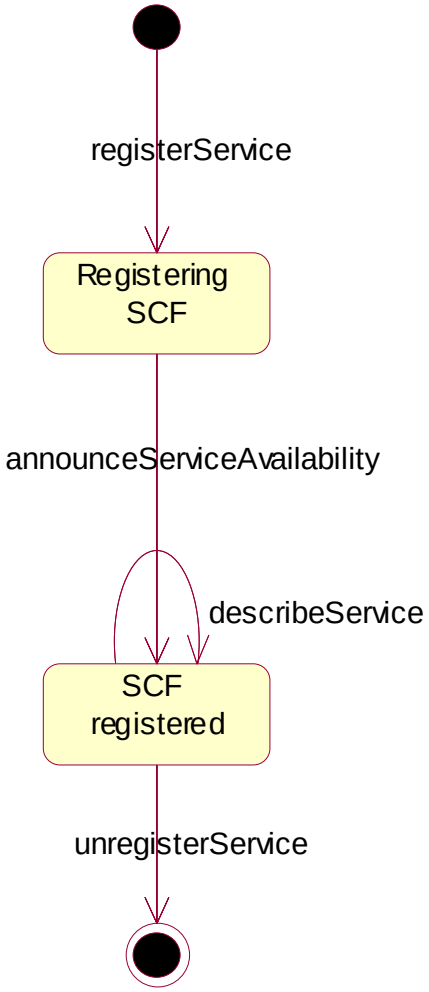


Figure : State Transition Diagram for IpFwServiceRegistration

13.1.1.1 Registering SCF State

This is the state entered when a Service Capability Server (SCS) starts the registration of its SCF in the Framework, by informing it of the existence of an SCF characterised by a service type and a set of service properties. As a result the Framework associates a service ID to this SCF, that will be used to identify it by both sides. When receiving this ID, the SCS instantiates a manager interface for this SCF, which will be the entry point for applications that want to use it.

13.1.1.2 SCF registered State

This is the state entered when, the service manager interface having been instantiated, the SCS informs the Framework of the availability of the SCF, and makes it actually available by providing the Framework with the manager interfaces to be used by applications. Anytime the SCF availability may be withdrawn by un-registering it.

13.2 Service Factory State Transition Diagrams

There are no State Transition Diagrams defined for Service Factory

14 Service Properties

14.1 Service Property Types

The service type defines which properties the supplier of an SCF supllier must provide when he registers an SCF.

At Service Registration the properties of a type must be interpreted as the set of values that can be supported by the service. If a service type has a certain property (e.g. "CAN_DO_SOMETHING"), a service registers with a proprety value of {"true", "false"}. This means that the SCS is able to support Service instances where this property is used or allowed and instances where this property is not used or allowed. This clarifies why sets of values must be used for the property values in stead of primitive types.

At establishment of the Service Level Agreement the property can then be set to the value of the specific agreement. The context of the Service Level Agreement thus restricts the set of property values of the SCS and will thus lead to a sub-set of the service property values. When the correct SCF is instantiated during the discovery and selection procedure[1], the Service Properties must thus be interpreted as the requested property values.

All property values are represented by an array of strings. The following table shows all supported property types.

Property type name	Description	Example value (array of strings)	Interpretation of example value
BOOLEAN_SET	set of booleans	{"FALSE"}	The set of booleans consisting of the boolean "false".
INTEGER_SET	set of integers	{"1", "2", "5", "7"}	The set of integers consisting of the integers 1, 2, 5 and 7.
STRING_SET	set of strings	{"Sophia", "Rijen"}	The set of strings consisting of the string “Sophia" and the string "Rijen"
ADDRESSRANGE_SET	set of address ranges	{"123??*", "*.ericsson.se"}	The set of address ranges consisting of ranges 123??* and *.ericsson.se.
INTEGER_INTERVAL	interval of integers	{"5", "100"}	The integers that are between or equal to 5 and 100.
STRING_INTERVAL	interval of strings	{"Rijen", "Sophia"}	The strings that are between or equal to the strings "Rijen" and "Sophia", in lexicographical order.
INTEGER_INTEGER_MAP	map from integers to integers	{"1", "10", "2", "20", "3", "30"}	The map that maps 1 to 10, 2 to 20 and 3 to 30.

The bounds of the string interval and the integer interval types may hold the reserved value "UNBOUNDED". If the left bound of the interval holds the value "UNBOUNDED", the lower bound of the interval is the smallest value supported by the type. If the right bound of the interval holds the value "UNBOUNDED", the upper bound of the interval is the largest value supported by the type.

14.2 General Service Properties

Each service instance has the following general properties:

- Service Name

[1] This is achieved through the getServiceManager() operation in the Service Factory interface.

- Service Version
- Service Instance ID
- Service Instance Description
- Product Name
- Product Version
- Supported Interfaces

14.2.1 Service Name

This property contains the name of the service, e.g. “UserLocation”, “UserLocationCamel”, “UserLocationEmergency” or “UserStatus”.

14.2.2 Service Version

This property contains the version of the APIs, to which the service is compliant, e.g. “2.1".

14.2.3 Service Instance ID

This property uniquely identifies a specific instance of the service. The Framework generates this property.

14.2.4 Service Instance Description

This property contains a textual description of the service.

14.2.5 Product Name

This property contains the name of the product that provides the service, e.g. “Find It”, “Locate.com”.

14.2.6 Product Version

This property contains the version of the product that provides the service, e.g. “3.1.11”.

14.2.7 Supported Interfaces

This property contains a list of strings with interface names that the service supports, e.g. “IpUserLocation”, “IpUserStatus”.

14.2.8 Operation Set

Property	Type	Description
P_OPERATION_SET	STRING_SET	Specifies set of the operations the SCS supports. The notation to be used is : {“Interface1.operation1”,”Interface1.operation2”, “Interface2.operation1”}, e.g.: {“IpCall.createCall”,”IpCall.routeReq”}.

15 Data Definitions

This section provides the framework specific data definitions necessary to support the OSA interface specification.

The general format of a data definition specification is the following:

- Data type, that shows the name of the data type.
- Description, that describes the data type.
- Tabular specification, that specifies the data types and values of the data type.
- Example, if relevant, shown to illustrate the data type.

15.1 Common Framework Data Definitions

15.1.1 TpClientAppID

This is an identifier for the client application. It is used to identify the client to the framework. This data type is identical to TpString and is defined as a string of characters that uniquely identifies the application. The content of this string shall be unique for each OSA API implementation (or unique for a network operator's domain). This unique identifier shall be negotiated with the OSA operator and the application shall use it to identify itself.

15.1.2 TpClientAppIDList

This data type defines a Numbered Set of Data Elements of type TpClientAppID.

15.1.3 TpDomainID

Defines the `Tagged Choice of Data Elements` that specify either the framework or the type of entity attempting to access the framework.

	Tag Element Type	
	TpDomainIDType	

Tag Element Value	Choice Element Type	Choice Element Name
P_FW	TpFwID	`FwID`
P_CLIENT_APPLICATION	TpClientAppID	`ClientAppID`
P_ENT_OP	TpEntOpID	`EntOpID`
P_REGISTERED_SERVICE	TpServiceID	`ServiceID`
P_SERVICE_SUPPLIER	TpServiceSupplierID	`ServiceSupplierID`

15.1.4 TpDomainIDType

Defines either the framework or the type of entity attempting to access the framework

Name	Value	Description
`P_FW`	0	The framework
`P_CLIENT_APPLICATION`	1	A client application
`P_ENT_OP`	2	An enterprise operator
`P_REGISTERED_SERVICE`	3	A registered service
`P_SERVICE_SUPPLIER`	4	A service supplier

15.1.5 TpEntOpID

This data type is identical to TpString and is defined as a string of characters that identifies an enterprise operator. In conjunction with the application it uniquely identifies the enterprise operator which uses a particular OSA Service Capability Feature.

15.1.6 TpPropertyName

This data type is identical to `TpString`. It is the name of a generic "property".

15.1.7 TpPropertyValue

This data type is identical to `TpString`. It is the value (or the list of values) associated with a generic "property".

15.1.8 TpProperty

This data type is a `Sequence of Data Elements` which describes a generic "property". It is a structured data type consisting of the following {name,value} pair:

Sequence Element Name	Sequence Element Type
PropertyName	TpPropertyName
PropertyValue	TpPropertyValue

15.1.9 TpPropertyList

This data type defines a `Numbered List of Data Elements` of type TpProperty.

15.1.10 TpEntOpIDList

This data type defines a Numbered Set of Data Elements of type TpEntOpID.

15.1.11 TpFwID

This data type is identical to `TpString` and identifies the Framework to a client application (or Service Capability Feature)

15.1.12 TpService

This data type is a Sequence of Data Elements which describes a registered SCFs. It is a structured type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
`ServiceID`	TpServiceID	
`ServicePropertyList`	TpServicePropertyList	

15.1.13 TpServiceList

This data type defines a Numbered Set of Data Elements of type TpService.

15.1.14 TpServiceDescription

This data type is a Sequence of Data Elements which describes a registered SCF. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceTypeName	TpServiceTypeName	
ServicePropertyList	TpServicePropertyList	

15.1.15 TpServiceID

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies an instance of a SCF interface. The string is automatically generated by the Framework, and comprises a TpUniqueServiceNumber, TpServiceTypeName, and a number of relevant TpServiceSpecString, which are concatenated using a forward separator (/) as the separation character.

15.1.16 TpServiceIDList

This data type defines a Numbered Set of Data Elements of type TpServiceID.

15.1.17 TpServiceIDRef

Defines a Reference to type TpServiceId.

15.1.18 TpServiceSpecString

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the name of an SCF specialization interface. Other network operator specific capabilities may also be used, but should be preceded by the string "SP_".The following values are defined.

Character String Value	Description
NULL	An empty (NULL) string indicates no SCF specialization
P_CALL	The Call specialization of the of the User Interaction SCF

15.1.19 TpUniqueServiceNumber

This data type is identical to a TpString, and is defined as a string of characters that represents a unique number that is used to build the service ID (refer to TpServiceID).

15.1.20 TpServiceTypeProperty

This data type is a Sequence of Data Elements which describes a service property associated with a service type. It defines the name and mode of the service property, and also the service property type: e.g. boolean, integer. It is similar to, but distinct from, TpServiceProperty. The latter is associated with an actual service: it defines the service property's name and mode, but also defines the list of values assigned to it.

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServicePropertyMode	TpServicePropertyMode	
ServicePropertyTypeName	TpServicePropertyTypeName	

15.1.21 TpServiceTypePropertyList

This data type defines a Numbered Set of Data Elements of type TpServiceTypeProperty.

15.1.22 TpServicePropertyMode

This type defines SCF property modes.

Name	Value	Documentation
NORMAL	0	The value of the corresponding SCF property type may optionally be provided
MANDATORY	1	The value of the corresponding SCF property type must be provided at service registration time
READONLY	2	The value of the corresponding SCF property type is optional, but once given a value it may not be modified
MANDATORY_READONLY	3	The value of the corresponding SCF property type must be provided and subsequently it may not be modified.

15.1.23 TpServicePropertyTypeName

This data type is identical to TpString and describes a valid SCF property name. The valid SCF property names are listed in the SCF data definition.

15.1.24 TpServicePropertyName

This data type is identical to TpString. It defines a valid SCF property name.

15.1.25 TpServicePropertyNameList

This data type defines a Numbered Set of Data Elements of type TpServicePropertyName.

15.1.26 TpServicePropertyValue

This data type is identical to TpString and describes a valid value of a SCF property.

15.1.27 TpServicePropertyValueList

This data type defines a Numbered Set of Data Elements of type TpServicePropertyValue

15.1.28 TpServiceProperty

This data type is a Sequence of Data Elements which describes an “SCF property”. It is a structured data type which consists of:

Sequence Element	Sequence Element	Documentation

Name	Type	
ServicePropertyName	TpServicePropertyName	
ServicePropertyValueList	TpServicePropertyValueList	
ServicePropertyMode	TpServicePropertyMode	

15.1.29 TpServicePropertyList

This data type defines a Numbered Set of Data Elements of type TpServiceProperty.

15.1.30 TpServiceSupplierID

This is an identifier for a service supplier. It is used to identify the supplier to the framework. This data type is identical to TpString.

15.1.31 TpServiceTypeDescription

This data type is a Sequence_of_Data_Elements which describes an SCF type. It is a structured data type. It consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceTypePropertyList	TpServiceTypePropertyList	a sequence of property name and property mode tuples associated with the SCF type
ServiceTypeNameList	TpServiceTypeNameList	the names of the super types of the associated SCF type
EnabledOrDisabled	TpBoolean	an indication whether the SCF type is enabled (true) or disabled (false)

15.1.32 TpServiceTypeName

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the type of an SCF interface. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_".The following values are defined.

Character String Value	Description
NULL	An empty (NULL) string indicates no SCF name
P_CALL_CONTROL	The name of the Call Control SCF
P_USER_INTERACTION	The name of the User Interaction SCFs
P_TERMINAL_CAPABILITIES	The name of the Terminal Capabilities SCF
P_USER_LOCATION_CAMEL	The name of the Network User Location SCF
P_USER_STATUS	The name of the User Status SCF
P_DATA_SESSION_CONTROL	The name of the Data Session Control SCF

15.1.33 TpServiceTypeNameList

This data type defines a Numbered Set of Data Elements of type TpServiceTypeName.

15.2 Event Notification Data Definitions

15.2.1 TpFwEventName

Defines the name of event being notified..

Name	Value	Description
P_EVENT_FW_NAME_UNDEFINED	0	Undefined
P_EVENT_FW_NEW_SERVICE_AVAILABLE	1	Notification of a new SCS available

15.2.2 TpFwEventCriteria

Defines the Tagged Choice of Data Elements that specify the criteria for an event notification to be generated.

	Tag Element Type	
	TpFwEventName	

Tag Element Value	Choice Element Type	Choice Element Name
P_EVENT_FW_NAME_UNDEFINED	TpString	EventNameUndefined
P_EVENT_FW_NEW_SERVICE_AVAILABLE	TpServiceTypeNameList	ServiceTypeNameList

15.2.3 TpFwEventInfo

Defines the Tagged Choice of Data Elements that specify the information returned to the application in an event notification.

	Tag Element Type	
	TpFwEventName	

Tag Element Value	Choice Element Type	Choice Element Name
P_EVENT_FW_NAME_UNDEFINED	TpString	EventNameUndefined

15.3 Trust and Security Management Data Definitions

15.3.1 TpAccessType

This data type is identical to a TpString. This identifies the type of access interface requested by the client application. If they request P_OSA_ACCESS, then a reference to the IpAccess interface is returned. (Network operators can define their own access interfaces to satisfy client requirements for different types of access. These can be selected using the TpAccessType, but should be preceded by the string "SP_". The following value is defined:

String Value	Description
P_OSA_ACCESS	Access using the OSA Access Interfaces: IpAccess and IpAppAccess

15.3.2 TpAuthType

This data type is identical to a TpString. It identifies the type of authentication mechanism requested by the client. It provides Network operators and client's with the opportunity to use an alternative to the OSA API Level Authentication interface. This can for example be an implementation specific authentication mechanism, e.g. CORBA Security, or a proprietary Authentication interface supported by the Network Operator. OSA API Level Authentication is the default authentication method. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined:

String Value	Description
P_OSA_AUTHENTICATION	Authenticate using the OSA API Level Authentication Interfaces: IpAPILevelAuthentication and IpAppAPILevelAuthentication
P_AUTHENTICATION	Authenticate using the implementation specific authentication mechanism, e.g. CORBA Security.

15.3.3 TpAuthCapability

This data type is identical to a TpString, and is defined as a string of characters that identify the authentication capabilities that could be supported by the OSA. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". Capabilities may be concatenated, using commas (,) as the separation character. The following values are defined.

String Value	Description
NULL	An empty (NULL) string indicates no client capabilities.
P_DES_56	A simple transfer of secret information that is shared between the client application and the framework with protection against interception on the link provided by the DES algorithm with a 56bit shared secret key
P_DES_128	A simple transfer of secret information that is shared between the client entity and the framework with protection against interception on the link provided by the DES algorithm with a 128bit shared secret key
P_RSA_512	A public-key cryptography system providing authentication without prior exchange of secrets using 512 bit keys
P_RSA_1024	A public-key cryptography system providing authentication without prior exchange of secrets using 1024bit keys

15.3.4 TpAuthCapabilityList

This data type is identical to a TpString. It is a string of multiple TpAuthCapability concatenated using a comma (,)as the separation character.

15.3.5 TpEndAccessProperties

This data type is of type TpPropertyList. It identifies the actions that the framework should perform when an application or service capability feature entity ends its access session (e.g. existing service capability or application sessions may be stopped, or left running).

15.3.6 TpAuthDomain

This is Sequence of Data Elements containing all the data necessary to identify a domain: the domain identifier, and a reference to the authentication interface of the domain

Sequence Element Name	Sequence Element Type	Description

DomainID	TpDomainID	Identifies the domain for authentication. This identifier is assigned to the domain during the initial contractual agreements, and is valid during the lifetime of the contract.
AuthInterface	IpInterfaceRef	Identifies the authentication interface of the specific entity. This data element has the same lifetime as the domain authentication process, i.e. in principle a new interface reference can be provided each time a domain intents to access another.

15.3.7 TpInterfaceName

This data type is identical to a TpString, and is defined as a string of characters that identify the names of the framework SCFs that are to be supported by the OSA API. Other Network operator specific SCFs may also be used, but should be preceded by the string "SP_".The following values are defined.

Character String Value	Description
P_DISCOVERY	The name for the Discovery interface.
P_EVENT_NOTIFICATION	The name for the Event Notification interface.
P_OAM	The name for the OA&M interface.
P_LOAD_MANAGER	The name for the Load Manager interface.
P_FAULT_MANAGER	The name for the Fault Manager interface.
P_HEARTBEAT_MANAGEMENT	The name for the Heartbeat Management interface.
P_REGISTRATION	The name for the Service Registration interface.
P_ENT_OP_ACCOUNT_MANAGEMENT	The name for the Service Subscription: Enterprise Operator Account Management interface.
P_ENT_OP_ACCOUNT_INFO_QUERY	The name for the Service Subscription: Enterprise Operator Account Information Query interface.
P_SVC_CONTRACT_MANAGEMENT	The name for the Service Subscription: Service Contract Management interface.
P_SVC_CONTRACT_INFO_QUERY	The name for the Service Subscription: Service Contract Information Query interface.
P_CLIENT_APP_MANAGEMENT	The name for the Service Subscription: Client Application Management interface.
P_CLIENT_APP_INFO_QUERY	The name for the Service Subscription: Client Application Information Query interface.
P_SVC_PROFILE_MANAGEMENT	The name for the Service Subscription: Service Profile Management interface.
P_SVC_PROFILE_INFO_QUERY	The name for the Service Subscription: Service Profile Information Query interface.

15.3.8 TpServiceAccessControl

This is Sequence of Data Elements containing the access control policy information controlling access to the service capability feature, and the trustLevel that the Network operator has assigned to the client application.

Sequence Element Name	Sequence Element Type
Policy	TpString
TrustLevel	TpString

The policy parameter indicates whether access has been granted or denied. If granted then the parameter trustLevel must also have a value.

The trustLevel parameter indicates the trust level that the Network operator has assigned to the client application.

15.3.9 TpSecurityContext

This data type is identical to a TpString and contains a group of security relevant attributes.

15.3.10 TpSecurityDomain

This data type is identical to a TpString and contains the security domain in which the client application is operating.

15.3.11 TpSecurityGroup

This data type is identical to a TpString and contains a definition of the access rights associated with all clients that belong to that group.

15.3.12 TpServiceAccessType

This data type is identical to a TpString and contains a definition of the specific security model in use.

15.3.13 TpServiceToken

This data type is identical to a TpString, and identifies a selected SCF. This is a free format text token returned by the framework, which can be signed as part of a service agreement. This will contain Network operator specific information relating to the service level agreement. The serviceToken has a limited lifetime, which is the same as the lifetime of the service agreement in normal conditions. If something goes wrong the serviceToken expires, and any method accepting the serviceToken will return an error code (P_INVALID_SERVICE_TOKEN). Service Tokens will automatically expire if the client or framework invokes the endAccess method on the other's corresponding access interface.

15.3.14 TpSignatureAndServiceMgr

This is a Sequence of Data Elements containing the digital signature of the framework for the service agreement, and a reference to the SCF manager interface of the SCF.

Sequence Element Name	Sequence Element Type
DigitalSignature	TpString
ServiceMgrInterface	IpServiceRef

The digitalSignature is the signed version of a hash of the service token and agreement text given by the client application.

The ServiceMgrInterface is a reference to the SCF manager interface for the selected SCF.

15.3.15 TpSigningAlgorithm

This data type is identical to a TpString, and is defined as a string of characters that identify the signing algorithm that must be used. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined.

String Value	Description
NULL	An empty (NULL) string indicates no signing algorithm is required
P_MD5_RSA_512	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 512 bit key.
P_MD5_RSA_1024	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public- key cryptography system using a 1024 bit key

15.4 Integrity Management Data Definitions

15.4.1 TpActivityTestRes

This type is identical to TpString and is an implementation specific result. The values in this data type are "Available" or "Unavailable".

15.4.2 TpFaultStatsRecord

This defines the set of records to be returned giving fault information for the requested time period.

Sequence Element Name	Sequence Element Type
Period	TpTimeInterval
FaultStatsSet	TpFaultStatsSet

15.4.3 TpFaultStats

This defines the sequence of data elements which provide the statistics on a per fault type basis.

Sequence Element Name	Sequence Element Type	Description
Fault	TpInterfaceFault	
Occurrences	TpInt32	The number of separate instances of this fault
MaxDuration	TpInt32	The number of seconds duration of the longest fault
TotalDuration	TpInt32	The cumulative duration (all occurrences)
NumberOfClientsAffected	TpInt32	The number of clients informed of the fault by the Fw

Occurrences is the number of separate instances of this fault during the period. MaxDuration and TotalDuration are the number of seconds duration of the longest fault and the cumulative total during the period. NumberOfClientsAffected is the number of clients informed of the fault by the framework.

15.4.4 TpFaultStatsSet

This data type defines a `Numbered Set of Data Elements` of type TpFaultStats

15.4.5 TpActivityTestID

This data type is identical to a TpInt32, and is used as a token to match activity test requests with their results..

15.4.6 TpInterfaceFault

Defines the cause of the interface fault detected.

Name	Value	Description
INTERFACE_FAULT_UNDEFINED	0	Undefined
INTERFACE_FAULT_LOCAL_FAILURE	1	A fault in the local API software or hardware has been detected
INTERFACE_FAULT_GATEWAY_FAILURE	2	A fault in the gateway API software or hardware has been detected
INTERFACE_FAULT_PROTOCOL_ERROR	3	An error in the protocol used on the client-gateway link has been detected

15.4.7 TpSvcUnavailReason

Defines the reason why a SCF is unavailable.

Name	Value	Description
SERVICE_UNAVAILABLE_UNDEFINED	0	Undefined
SERVICE_UNAVAILABLE_LOCAL_FAILURE	1	The Local API software or hardware has failed
SERVICE_UNAVAILABLE_GATEWAY_FAILURE	2	The gateway API software or hardware has failed
SERVICE_UNAVAILABLE_OVERLOADED	3	The SCF is fully overloaded
SERVICE_UNAVAILABLE_CLOSED	4	The SCF has closed itself (e.g. to protect from fraud or malicious attack)

15.4.8 TpFWUnavailReason

Defines the reason why the Framework is unavailable.

Name	Value	Description
FW_UNAVAILABLE_UNDEFINED	0	Undefined
FW_UNAVAILABLE_LOCAL_FAILURE	1	The Local API software or hardware has failed
FW_UNAVAILABLE_GATEWAY_FAILURE	2	The gateway API software or hardware has failed
FW_UNAVAILABLE_OVERLOADED	3	The framework is fully overloaded
FW_UNAVAILABLE_CLOSED	4	The framework has closed itself (e.g. to protect from fraud or malicious attack)
FW_UNAVAILABLE_PROTOCOL_FAILURE	5	The protocol used on the client-gateway link has failed

15.4.9 TpLoadLevel

Defines the Sequence of Data Elements that specify load level values.

Name	Value	Description
LOAD_LEVEL_NORMAL	0	Normal load
LOAD_LEVEL_OVERLOAD	1	Overload
LOAD_LEVEL_SEVERE_OVERLOAD	2	Severe Overload

15.4.10 TpLoadThreshold

Defines the Sequence of Data Elements that specify the load threshold value. The actual load threshold value is application and SCF dependent, so is their relationship with load level.

Sequence Element Name	Sequence Element Type
LoadThreshold	TpFloat

15.4.11 TpLoadInitVal

Defines the Sequence of Data Elements that specify the pair of load level and associated load threshold value.

Sequence Element Name	Sequence Element Type

LoadLevel	TpLoadLevel
LoadThreshold	TpLoadThreshold

15.4.12 TpTimeInterval

Defines the Sequence of Data Elements that specify a time interval.

Sequence Element Name	Sequence Element Type
StartTime	TpDateAndTime
StopTime	TpDateAndTime

15.4.13 TpLoadPolicy

Defines the load balancing policy.

Sequence Element Name	Sequence Element Type
LoadPolicy	TpString

15.4.14 TpLoadStatistic

Defines the Sequence of Data Elements that represents a load statistic record for a specific entity (i.e. framework, service or application) at a specific date and time.

Sequence Element Name	Sequence Element Type
LoadStatisticEntityID	TpLoadStatisticEntityID
TimeStamp	TpDateAndTime
LoadStatisticInfo	TpLoadStatisticInfo

15.4.15 TpLoadStatisticList

Defines a Numbered List of Data Elements of type TpLoadStatistic.

15.4.16 TpLoadStatisticData

Defines the Sequence of Data Elements that represents load statistic information

Sequence Element Name	Sequence Element Type
LoadValue	TpFloat
LoadLevel	TpLoadLevel

Note: LoadValue is expressed as a percentage.

15.4.17 TpLoadStatisticEntityID

Defines the Tagged Choice of Data Elements that specify the type of entity (i.e. service, application or framework) providing load statistics.

	Tag Element Type	

	TpLoadStatisticEntityType	

Tag Element Value	Choice Element Type	Choice Element Name
P_LOAD_STATISTICS_FW_TYPE	TpFwID	FrameworkID
P_LOAD_STATISTICS_SVC_TYPE	TpServiceID	ServiceID
P_LOAD_STATISTICS_APP_TYPE	TpClientAppID	ClientAppID

15.4.18 TpLoadStatisticEntityType

Defines the type of entity (i.e. service, application or framework) supplying load statistics.

Name	Value	Description
P_LOAD_STATISTICS_FW_TYPE	0	Framework-type load statistics
P_LOAD_STATISTICS_SVC_TYPE	1	Service-type load statistics
P_LOAD_STATISTICS_APP_TYPE	2	Application-type load statistics

15.4.19 TpLoadStatisticInfo

Defines the Tagged Choice of Data Elements that specify the type of load statistic information (i.e. valid or invalid).

	Tag Element Type	
	TpLoadStatisticInfoType	

Tag Element Value	Choice Element Type	Choice Element Name
P_LOAD_STATISTICS_VALID	TpLoadStatisticData	LoadStatisticData
P_LOAD_STATISTICS_INVALID	TpLoadStatisticError	LoadStatisticError

15.4.20 TpLoadStatisticInfoType

Defines the type of load statistic information (i.e. valid or invalid).

Name	Value	Description
P_LOAD_STATISTICS_VALID	0	Valid load statistics
P_LOAD_STATISTICS_INVALID	1	Invalid load statistics

15.4.21 TpLoadStatisticError

Defines the error code associated with a failed attempt to retrieve any load statistics information.

Name	Value	Description
P_LOAD_INFO_ERROR_UNDEFINED	0	Undefined error
P_LOAD_INFO_UNAVAILABLE	1	Load statistics unavailable

15.5 Service Subscription Data Definitions

15.5.1 TpPropertyName

This data type is identical to `TpString`. It is the name of a generic "property".

15.5.2 TpPropertyValue

This data type is identical to `TpString`. It is the value (or the list of values) associated with a generic "property".

15.5.3 TpProperty

This data type is a `Sequence of Data Elements` which describes a generic "property". It is a structured data type consisting of the following {name,value} pair:

Sequence Element Name	Sequence Element Type
`PropertyName`	TpPropertyName
`PropertyValue`	TpPropertyValue

15.5.4 TpPropertyList

This data type defines a `Numbered List of Data Elements` of type TpProperty.

15.5.5 TpEntOpProperties

This data type is of type TpPropertyList. It identifies the list of properties associated with an enterprise operator: e.g. name, organisation, address, phone, e-mail, fax, payment method (credit card, bank account).

15.5.6 TpEntOp

This data type is a `Sequence of Data Elements` which describes an enterprise operator. It is a structured data type, consisting of a unique "enterprise operator ID" and a list of "enterprise operator properties", as follows:

Sequence Element Name	Sequence Element Type
`EntOpID`	TpEntOpID
`EntOpProperties`	TpEntOpProperties

15.5.7 TpServiceContractID

This data type is identical to `TpString`. It uniquely identifies the contract, between an enterprise operator and the framework, for the use of a Parlay service by the enterprise.

15.5.8 TpPersonName

This data type is identical to `TpString`. It is the name of a generic "person".

15.5.9 TpPostalAddress

This data type is identical to `TpString`. It is the mailing address of a generic "person".

15.5.10 TpTelephoneNumber

This data type is identical to `TpString`. It is the telephone number of a generic "person".

15.5.11 TpEmail

This data type is identical to `TpString`. It is the email address of a generic "person".

15.5.12 TpHomePage

This data type is identical to `TpString`. It is the web address of a generic "person".

15.5.13 TpPersonProperties

This data type is of type TpPropertyList. It identifies the list of additional properties, other than those listed above, that can be associated with a generic "person".

15.5.14 TpPerson

This data type is a `Sequence of Data Elements` which describes a generic "person": e.g. a billing contact, a service requestor. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
`PersonName`	TpPersonName
`PostalAddress`	TpPostalAddress
`TelephoneNumber`	TpTelephoneNumber
`Email`	TpEmail
`HomePage`	TpHomePage
`PersonProperties`	TpPersonProperties

15.5.15 TpServiceStartDate

This is of type `TpDateAndTime`. It identifies the contractual start date and time for the use of a Parlay service by an enterprise or an enterprise SAG.

15.5.16 TpServiceEndDate

This is of type `TpDateAndTime`. It identifies the contractual end date and time for the use of a Parlay service by an enterprise or an enterprise SAG.

15.5.17 TpServiceRequestor

This is of type TpPerson. It identifies the enterprise person requesting use of a Parlay service: e.g. the enterprise operator.

15.5.18 TpBillingContact

This is of type TpPerson. It identifies the enterprise person responsible for billing issues associated with an enterprise's use of a Parlay service.

15.5.19 TpServiceSubscriptionProperties

This is of type TpPropertyList. It specifies a subset of all available service properties and service property values that apply to an enterprise's use of a Parlay service.

15.5.20 TpServiceContract

This data type is a `Sequence of Data Elements` which describes a service contract. This contract should conform to a previously negotiated high-level agreement (regarding Parlay services, their usage and the price, etc.), if any, between the enterprise operator and the framework operator. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
`ServiceContractID`	TpServiceContractID
`ServiceRequestor`	TpServiceRequestor
`BillingContact`	TpBillingContact
`ServiceStartDate`	TpServiceStartDate
`ServiceEndDate`	TpServiceEndDate
`ServiceTypeName`	TpServiceTypeName
`ServiceID`	TpServiceID
`ServiceSubscriptionProperties`	TpServiceSubscriptionProperties

15.5.21 TpPassword

This data type is identical to `TpString`. It is a password assigned to a client application for authentication purposes.

15.5.22 TpClientAppProperties

This is of type TpPropertyList. The client application properties is a list of {name,value} pairs, for bilateral agreement between the enterprise operator and the framework.

15.5.23 TpClientAppDescription

This data type is a `Sequence of Data Elements` which describes an enterprise client application. It is a structured data type, consisting of a unique "client application ID", password and a list of "client application properties:

Sequence Element Name	Sequence Element Type
`ClientAppID`	TpClientAppID
`Password`	TpPassword
`ClientAppProperties`	TpClientAppProperties

15.5.24 TpSagID

This data type is identical to `TpString`. It uniquely identifies a Subscription Assignment Group (SAG) of client applications within an enterprise.

15.5.25 TpSagIDList

This data type defines a `Numbered List of Data Elements` of type TpSagID.

15.5.26 TpSagDescription

This data type is identical to `TpString`. It describes a SAG: e.g. a list of identifiers of the constituent client applications, the purpose of the "grouping".

15.5.27 TpSag

This data type is a `Sequence of Data Elements` which describes a Subscription Assignment Group (SAG) of client applications within an enterprise. It is a structured data type consisting of a unique SAG ID and a description:

Sequence Element Name	Sequence Element Type
`SagID`	TpSagID
`SagDescription`	TpSagDescription

15.5.28 TpServiceProfileID

This data type is identical to `TpString`. It uniquely identifies the service profile, which further constrains how an enterprise SAG uses a Parlay service.

15.5.29 TpServiceProfileIDList

This data type defines a `Numbered List of Data Elements` of type TpServiceProfileID.

15.5.30 TpServiceProfile

This data type is a `Sequence of Data Elements` which describes a Service Profile. A service contract contains one or more Service Profiles, one for each SAG in the enterprise operator domain. A service profile is a restriction of the service contract in order to provide restricted service features to a SAG. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
`ServiceProfileID`	TpServiceProfileID
`ServiceContractID`	TpServiceContractID
`ServiceStartDate`	TpServiceStartDate
`ServiceEndDate`	TpServiceEndDate
`ServiceTypeName`	TpServiceTypeName
`ServiceSubscriptionProperties`	TpServiceSubscriptionProperties

Annex A (normative): OMG IDL Description of Framework

The OMG IDL representation of this interface specification is contained in a text file (fw.idl contained in archive2919803IDL.ZIP) which accompanies the present document.

Annex B (informative): Differences between this draft and 3GPP 29.198 R99

The following is a list of the differences between this draft and 3GPP 29.198 R99, for those items which are common to both documents. Any new interfaces/methods with respect to Release 99 are not listed.

B.1 IpService Registration

Interface Class IpServiceRegistration in R99 renamed IpFwServiceRegistration

B.2 IDL Namespace

IDL namespace has been extended. Instead of all interfaces being under org::open-service-access::fw, now all interfaces except IpFwServiceRegistratin and IpSvcFactory are under fw::fw_client, and IpFwServiceRegistration and IpSvcFactory are under fw::fw_service

B.3 IpAccess

accessCheck(serviceToken: in TpServiceToken,securityContext: in ~~TpString~~TpSecurityContext, securityDomain: in ~~TpString~~TpSecurityDomain, group : in ~~TpString~~TpSecurityGroup, serviceAccessTypes: in ~~TpString~~TpServiceAccessType, serviceAccessControl: out TpServiceAccessControlRef): TpResult

B.4 IpAPILevelAuthentication, IpAppAPILevelAuthentication

Interfaces IpAuthentication and IpAppAuthentication renamed as IpAPILevelAuthentication and IpAppAPILevelAuthentication. New interface IpAuthentication added. IpAPILevelAuthentication inherits from IpAuthentication.

selectEncryptionMethod~~selectAuthMethod~~ (authCaps : in TpAuthCapabilityList, prescribedMethod : out TpAuthCapabilityRef) : TpResult

B.5 New IpAuthentication

requestAccess (accessType : in TpAccessType, appAccessInterface : in IpInterfaceRef, fwAccessInterface : out IpInterfaceRefRef) : TpResult added.

B.6 IpInitial

~~requestAccess (accessType : in TpAccessType, appAccessInterface : in IpInterfaceRef, fwAccessInterface : out IpInterfaceRefRef) : TpResult~~ deleted from interface.

B.7 IpAppLoadManager

~~disableLoadControl (serviceIDs : in TpServiceIDList) : TpResult~~

~~enableLoadControl (loadStatistics : in TpLoadStatisticList) : TpResult~~

loadLevelNotification(loadStatistics : in TpLoadStatisticList) : TpResult

B.8 Data Type Changes

TpServiceID

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies an instance of a SCF interface. The string is automatically generated by the Framework, and comprises a TpUniqueServiceNumber, ~~TpServiceNameString~~ TpServiceTypeName, and a number of relevant TpServiceSpecString, which are concatenated using a forward separator (/) as the separation character.

TpServiceIDList

This data type defines a Numbered Set of Data Elements of type TpServiceID.

TpServiceIDRef

Defines a Reference to type TpServiceId.

~~**TpServiceNameString**~~

~~This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the name of an SCF interface. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_".The following values are defined for OSA release 99.~~

~~Character String Value~~	~~Description~~
~~*NULL*~~	~~An empty (NULL) string indicates no SCF name~~
~~P_CALL_CONTROL~~	~~The name of the Call Control SCF~~
~~P_USER_INTERACTION~~	~~The name of the User Interaction SCFs~~
~~P_TERMINAL_CAPABILITIES~~	~~The name of the Terminal Capabilities SCF~~
~~P_USER_LOCATION_CAMEL~~	~~The name of the Network User Location SCF~~
~~P_USER_STATUS~~	~~The name of the User Status SCF~~
~~P_DATA_SESSION_CONTROL~~	~~The name of the Data Session Control SCF~~

TpServiceSpecString

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the name of an SCF specialization interface. Other network operator specific capabilities may also be used, but should be preceded by the string "SP_".The following values are defined for OSA release 99.

Character String Value	Description
NULL	An empty (NULL) string indicates no SCF specialization
P_CALL	The Call specialization of the of the User Interaction SCF

TpUniqueServiceNumber

This data type is identical to a TpString, and is defined as a string of characters that represents a unique number that is used to build the service ID (refer to TpServiceID).

TpServiceTypeProperty

This data type is a Sequence of Data Elements which describes a service property associated with a service type. It defines the name and mode of the service property, and also the service property type: e.g. boolean, integer. It is similar to, but distinct from, TpServiceProperty. The latter is associated with an actual service: it defines the service property's name and mode, but also defines the list of values assigned to it.

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServicePropertyMode	TpServicePropertyMode	
ServicePropertyTypeName	TpServicePropertyTypeName	

TpServiceTypePropertyList

This data type defines a Numbered Set of Data Elements of type TpServiceTypeProperty.

TpServicePropertyMode

This type is left as a placeholder but is not used in release 99.This defines SCF property modes.

Name	Value	Documentation
NORMAL	0	The value of the corresponding SCF property type may optionally be provided
MANDATORY	1	The value of the corresponding SCF property type must be provided at service registration time
READONLY	2	The value of the corresponding SCF property type is optional, but once given a value it may not be modified
MANDATORY_READONLY	3	The value of the corresponding SCF property type must be provided and subsequently it may not be modified.

TpServicePropertyTypeName

This data type is identical to TpString and describes a valid SCF property name. The valid SCF property names are listed in the SCF data definition.

TpServicePropertyName

This data type is identical to TpString. It defines a valid S~~F~~CF property name. ~~Valid SCF property names are listed in the SCF data definition.~~

TpServicePropertyNameList

This data type defines a Numbered Set of Data Elements of type TpServicePropertyName.

TpServicePropertyValue

This data type is identical to TpString and describes a valid value of a SCF property. ~~The valid SCF property values are given in the SCF data definition.~~

TpServicePropertyValueList

This data type defines a Numbered Set of Data Elements of type TpServicePropertyValue

TpServiceProperty

This data type is a Sequence of Data Elements which describes an “SCF property”. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServicePropertyValueList	TpServicePropertyValueList	
ServicePropertyMode	TpServicePropertyMode	

TpServicePropertyList

This data type defines a Numbered Set of Data Elements of type TpServiceProperty.

TpServiceSupplierID

This is an identifier for a service supplier. It is used to identify the supplier to the framework. This data type is identical to TpString.

TpServiceTypeDescription

This type is left as a placeholder but is not used in release 99.

This data type is a Sequence_of_Data_Elements which describes an SCF type. It is a structured data type. It consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceTypePropertyList	TpServiceTypePropertyList	a sequence of property name and property mode tuples associated with the SCF type
ServiceTypeNameList	TpServiceTypeNameList	the names of the super types of the associated SCF type
EnabledOrDisabled	TpBoolean	an indication whether the SCF type is enabled or disabled

TpServiceTypeName

~~This data type is identical to TpString~~ and describes a valid SCF type name.This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the type of an SCF interface. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_".The following values are defined for OSA release 99.

Character String Value	Description
NULL	An empty (NULL) string indicates no SCF name
P_CALL_CONTROL	The name of the Call Control SCF
P_USER_INTERACTION	The name of the User Interaction SCFs
P_TERMINAL_CAPABILITIES	The name of the Terminal Capabilities SCF
P_USER_LOCATION_CAMEL	The name of the Network User Location SCF
P_USER_STATUS	The name of the User Status SCF
P_DATA_SESSION_CONTROL	The name of the Data Session Control SCF

TpServiceTypeNameList

This data type defines a Numbered Set of Data Elements of type TpServiceTypeName.

TpSecurityContext

This data type is identical to a TpString and contains a group of security relevant attributes.

TpSecurityDomain

This data type is identical to a TpString and contains the security domain in which the client application is operating.

TpSecurityGroup

This data type is identical to a TpString and contains a definition of the access rights associated with all clients that belong to that group.

TpServiceAccessType

This data type is identical to a TpString and contains a definition of the specific security model in use.

TpAccessType

This data type is identical to a TpString. This identifies the type of access interface requested by the client application. If they request P_OSA_ACCESS, then a reference to the IpAccess interface is returned. (Network operators can define their own access interfaces to satisfy client requirements for different types of access. These can be selected using the TpAccessType, but should be preceded by the string "SP_". The following value is defined :

String Value	Description
P_OSA_ACCESS	Access using the OSA Access Interfaces: IpAccess and IpAppAccess

TpAuthType

This data type is identical to a TpString. It identifies the type of authentication mechanism requested by the client. It provides Network operators and client's with the opportunity to use an alternative to the OSA API Level Authentication interface. This can for example be an implementation specific authentication mechanism, e.g. CORBA Security, or a proprietary Authentication interface supported by the Network Operator. OSA API Level Authentication is the default authentication method. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined :

String Value	Description
P_OSA_AUTHENTICATION	Authenticate using the OSA API Level Authentication Interfaces: IpAPILevelAuthentication and IpAppAPILevelAuthentication
P_AUTHENTICATION	Authenticate using the implementation specific authentication mechanism, e.g. CORBA Security.

TpFaultStatsRecord

This defines the set of records to be returned giving fault information for the requested time period.

Sequence Element Name	Sequence Element Type
Period	TpTimeInterval
FaultStatsSet~~FaultRecords~~	TpFaultStatsSet

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for approval and placement under Change Control

3GPP TS 29.198-4 V1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface
Part 4: Call Control
(Release 4)

Keywords
API, OSA, IDL, GCC, MPCC, Call Control

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

This document is part of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA). The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA API's. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

This document specifies the Call Control Service Capability Feature (SCF) aspects of the interface. All aspects of the Call Control SCF are defined here, these being:

- Sequence Diagrams
- Class Diagrams
- Interface specification plus detailed method descriptions
- State Transition diagrams
- Data definitions
- IDL Description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modelling Language (UML).

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

[1] 3GPP TS 29.198-1 "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in TS 29.198-1 [1] apply.

3.2 Symbols

For the purposes of the present document, the symbols in TS 29.198-1 [1] apply.

3.3 Abbreviations

For the purposes of the present document, the abbreviations in TS 29.198-1 [1] apply.

4 Call Control SCF

Two flavours of call control APIs have been included in Rel.4. These are the generic call control and the multi-party call control. The generic call control is the same API as was already present in the previous specification for Rel.99 (TS 29.198 v3.2.0) and is in principle able to satisfy the requirements on Call Control APIs for Rel.4.

However, the joint work between 3GPP CN5, ETSI SPAN12 and the Parlay Call Control Working group with collaboration from JAIN has been focussed on the Multi-party call control API. A number of improvements on call control functionality have been made and are reflected in this API. For this it was necessary to break the inheritance that previously existed between Generic and Multi-party call control.

The joint call control group has furthermore decided that the multi-party call control is to be considered as the future base call control family and the technical work will not be continued on Generic Call control. Errors or technical flaws will of course be corrected.

The following sections describe each aspect of the Call Control Service Capability Feature (SCF).

The order is as follows:

- The Sequence diagrams give the reader a practical idea of how each of the service capability feature is implemented.
- The Class relationships section show how each of the interfaces applicable to the SCF, relate to one another
- The Interface specification section describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the progression of internal processes either in the application, or Gateway.
- The Data definitions section show a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.

5 The Service Interface Specifications

5.1 Interface Specification Format

This section defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

5.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for enterprise-based client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>

5.1.2 Method descriptions

Each method (API method “call”) is described. All methods in the API return a value of type `TpResult`, indicating, amongst other things, if the method invocation was sucessfully executed or not.

Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a '`Req`' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a '`Res`' or '`Err`' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name> or IpSvc<name>` interfaces to provide the callback mechanism.

5.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.

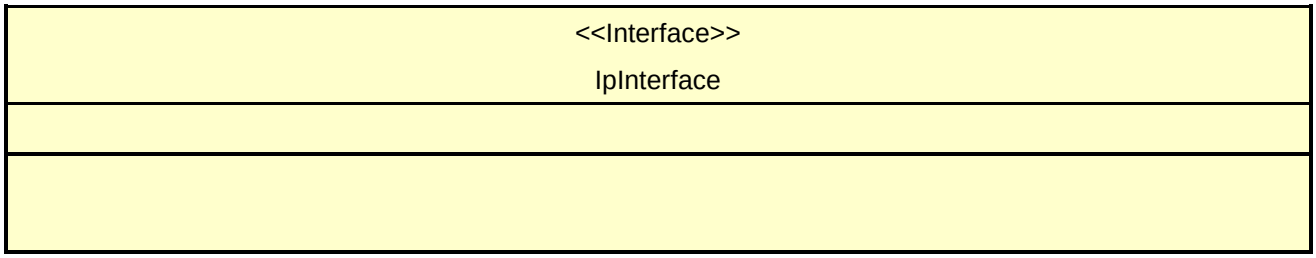
5.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

5.2 Base Interface

5.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.



5.3 Service Interfaces

5.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

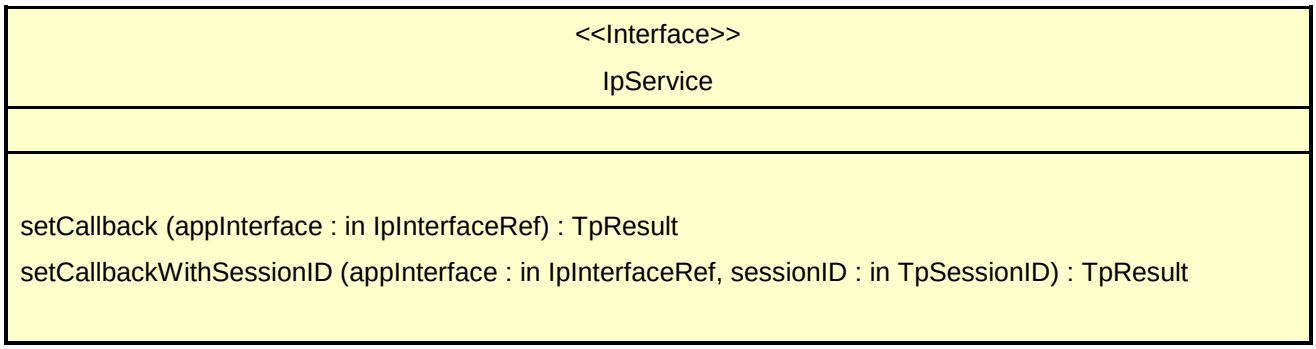
The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

5.4 Generic Service Interface

5.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.



Method

setCallback()

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

Raises

TpGeneralException

Method

setCallbackWithSessionID()

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

sessionID : in TpSessionID

Specifies the session for which the service can invoke the application's callback interface.

Raises

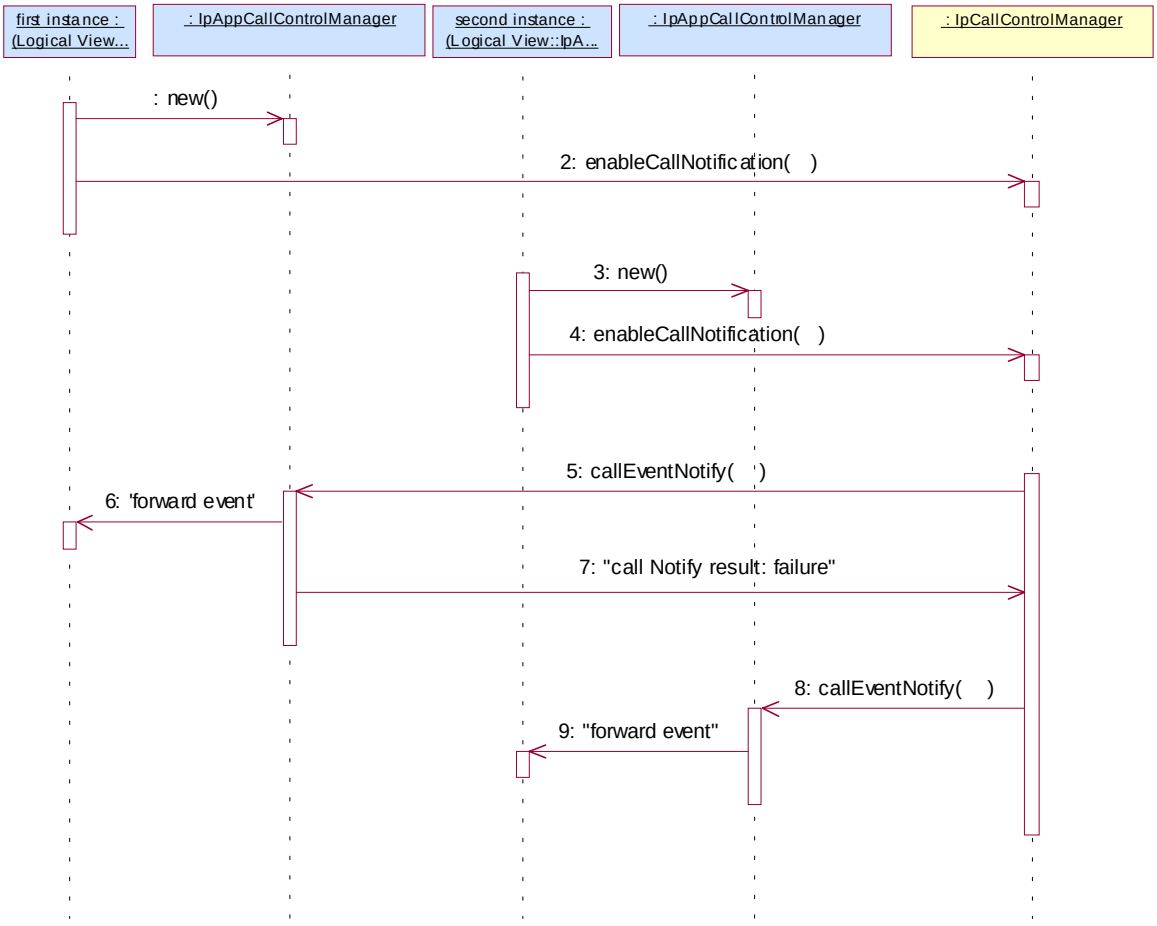
`TpGeneralException`

6 Generic Call Control Service

6.1 Sequence Diagrams

6.1.1 Additional Callbacks

The following sequence diagram shows how an application can register two call back interfaces for the same set of events. If one of the call backs can not be used, e.g., because the application crashed, the other call back interface is used instead.



1: The first instance of the application is started on node 1. The application creates a new IpAppCallControlManager to handle callbacks for this first instance of the logic.

2: The enableCallNotfication is associated with an applicationID. The call control manager uses the applicationID to decide whether this is the same application.

3: The second instance of the application is started on node 2. The application creates a new IpAppCallControlManager to handle callbacks for this second instance of the logic.

4: The same enableCallNotfication request is sent as for the first instance of the logic. Because both requests are associated with the same application, the second request is not rejected, but the specified callback object is stored as an additional callback.

5: When the trigger occurs one of the first instance of the application is notified. The gateway may have different policies on how to handle additional callbacks, e.g., always first try the first registered or use some kind of round robin scheme.

6: The event is forwarded to the first instance of the logic.

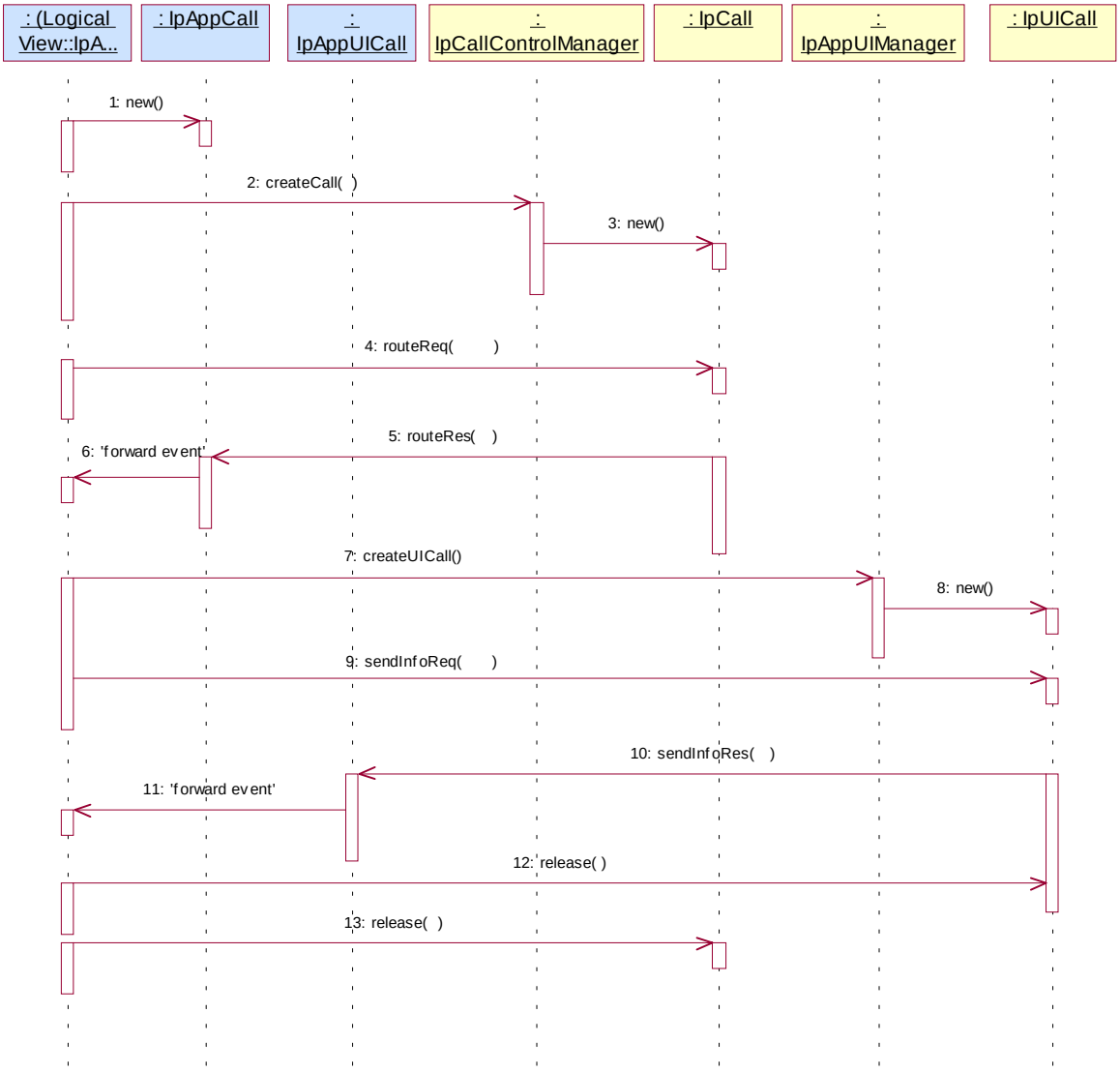
7: When the first instance of the application is overloaded or unavailable this is communicated with an exception to the call control manager.

8: Based on this exception the call control manager will notify another instance of the application (if available).

9: The event is forwarded to the second instance of the logic.

6.1.2 Alarm Call

The following sequence diagram shows a 'reminder message', in the form of an alarm, being delivered to a customer as a result of a trigger from an application. Typically, the application would be set to trigger at a certain time, however, the application could also trigger on events.



1: This message is used to create an object implementing the IpAppCall interface.

2: This message requests the object implementing the IpCallControlManager interface to create an object implementing the IpCall interface.

3: Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met it is created.

4: This message instructs the object implementing the IpCall interface to route the call to the customer destined to receive the 'reminder message'

5: This message passes the result of the call being answered to its callback object.

6: This message is used to forward the previous message to the IpAppLogic.

7: The application requests a new UICall object that is associated with the call object.

8: Assuming all criteria are met, a new UICall object is created by the service.

9: This message instructs the object implementing the IpUICall interface to send the alarm to the customer's call.

10: When the announcement ends this is reported to the call back interface.

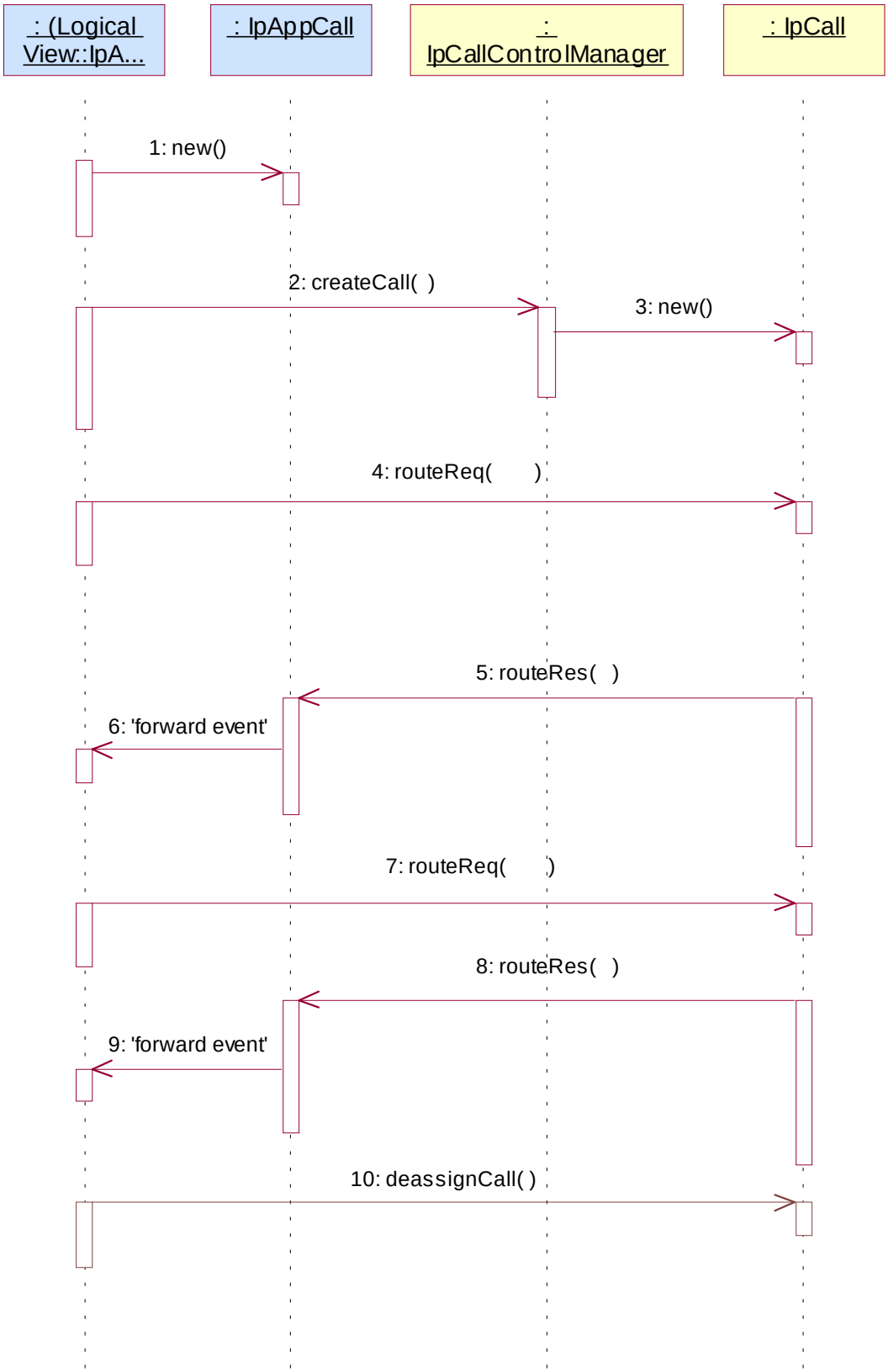
11: The event is forwarded to the application logic.

12: The application releases the UICall object, since no further announcements are required. Alternatively, the application could have indicated P_FINAL_REQUEST in the sendInfoReq in which case the UICall object would have been implicitly released after the announcement was played.

13: The application releases the call and all associated parties.

6.1.3 Application Initiated Call

The following sequence diagram shows an application creating a call between party A and party B. This sequence could be done after a customer has accessed a Web page and selected a name on the page of a person or organisation to talk to.


: (Logical View::IpA...
: IpAppCall
: IpCallControlManager
: IpCall
1: new()
2: createCall( )
3: new()
4: routeReq( )
5: routeRes( )
6: 'forward event'
7: routeReq( )
8: routeRes( )
9: 'forward event'
10: deassignCall( )

1: This message is used to create an object implementing the IpAppCall interface.

2: This message requests the object implementing the IpCallControlManager interface to create an object implementing the IpCall interface.

3: Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, it is created.

4: This message is used to route the call to the A subscriber (origination). In the message the application request response when the A party answers.

5: This message indicates that the A party answered the call.

6: This message forwards the previous message to the application logic.

7: This message is used to route the call to the B-party. Also in this case a response is requested for call answer or failure.

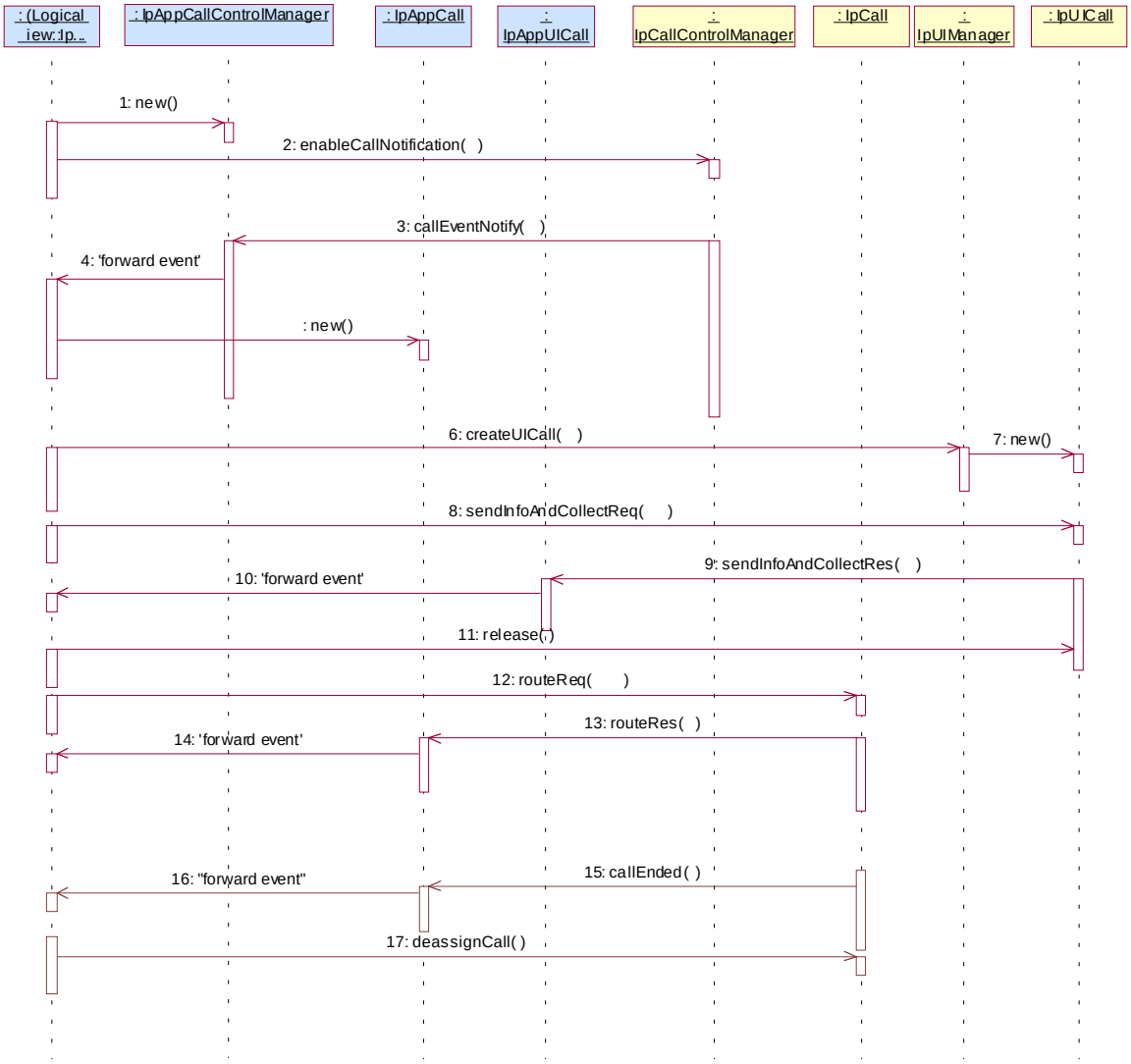
8: This message indicates that the B-party answered the call. The call now has two parties and a speech connection is automatically established between them.

9: This message is used to forward the previous message to the IpAppLogic.

10: Since the application is no longer interested in controlling the call, the application deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

6.1.4 Call Barring 1

The following sequence diagram shows a call barring service, initiated as a result of a prearranged event being received by the framework. Before the call is routed to the destination number, the calling party is asked for a PIN code. The code is accepted and the call is routed to the original called party.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria set, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward the previous message to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.

6: This message is used to create a new UICall object. The reference to the call object is given when creating the UICall.

7: Provided all the criteria are fulfilled, a new UICall object is created.

8: The call barring service dialogue is invoked.

9: The result of the dialogue, which in this case is the PIN code, is returned to its callback object.

10: This message is used to forward the previous message to the IpAppLogic.

11: This message releases the UICall object.

12: Assuming the correct PIN is entered, the call is forward routed to the destination party.

13: This message passes the result of the call being answered to its callback object.

14: This message is used to forward the previous message to the IpAppLogic

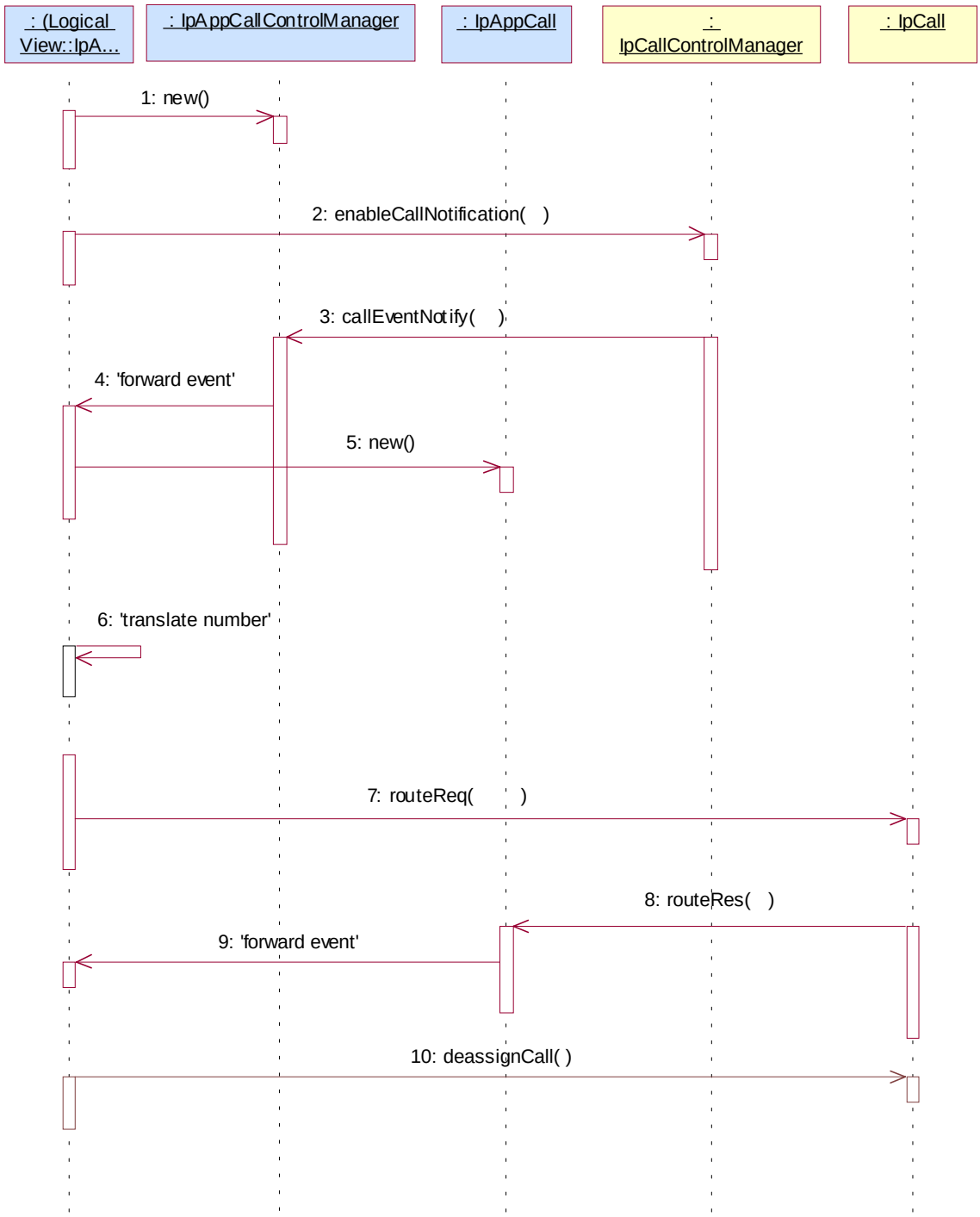
15: When the call is terminated in the network, the application will receive a notification. This notification will always be received when the call is terminated by the network in a normal way, the application does not have to request this event explicitly.

16: The event is forwarded to the application.

17: The application must free the call related resources in the gateway by calling deassignCall.

6.1.5 Number Translation 1

The following sequence diagram shows a simple number translation service, initiated as a result of a prearranged event being received by the framework.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward message 3 to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of message 3.

6: This message invokes the number translation function.

7: The returned translated number is used in message 7 to route the call towards the destination.

8: This message passes the result of the call being answered to its callback object

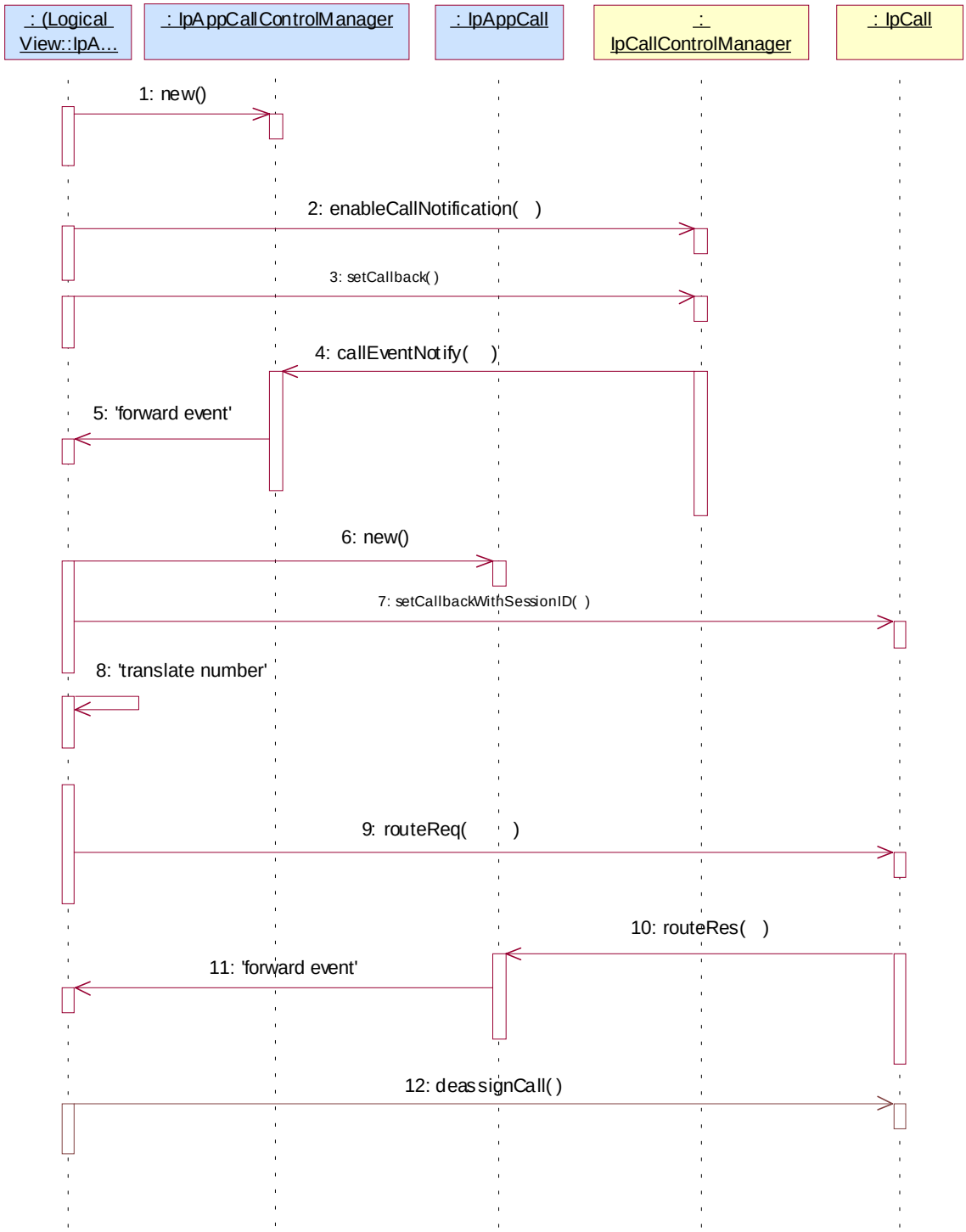
9: This message is used to forward the previous message to the IpAppLogic.

10: The application is no longer interested in controlling the call and therefore deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

6.1.6 Number Translation 1 (with callbacks)

The following sequence diagram shows a simple number translation service, initiated as a result of a prearranged event being received by the framework.

For illustation, in this sequence the callback references are set explictly. This is optional. All the callbacks references can also be passed in other methods. From an efficiency point of view that is also the preferred method. The rest of the sequences use that mechanism.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall

interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message sets the reference of the IpAppCallControlManager object in the CallControlManager. The CallControlManager reports the callEventNotify to referenced object only for enableCallNotification's that do not have a explicit IpAppCallControlManager reference specified in the enableCallNotification.

4: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

5: This message is used to forward message 4 to the IpAppLogic.

6: This message is used by the application to create an object implementing the IpAppCall interface.

7: This message is used to set the reference to the IpAppCall for this call.

8: This message invokes the number translation function.

9: The returned translated number is used in message 7 to route the call towards the destination.

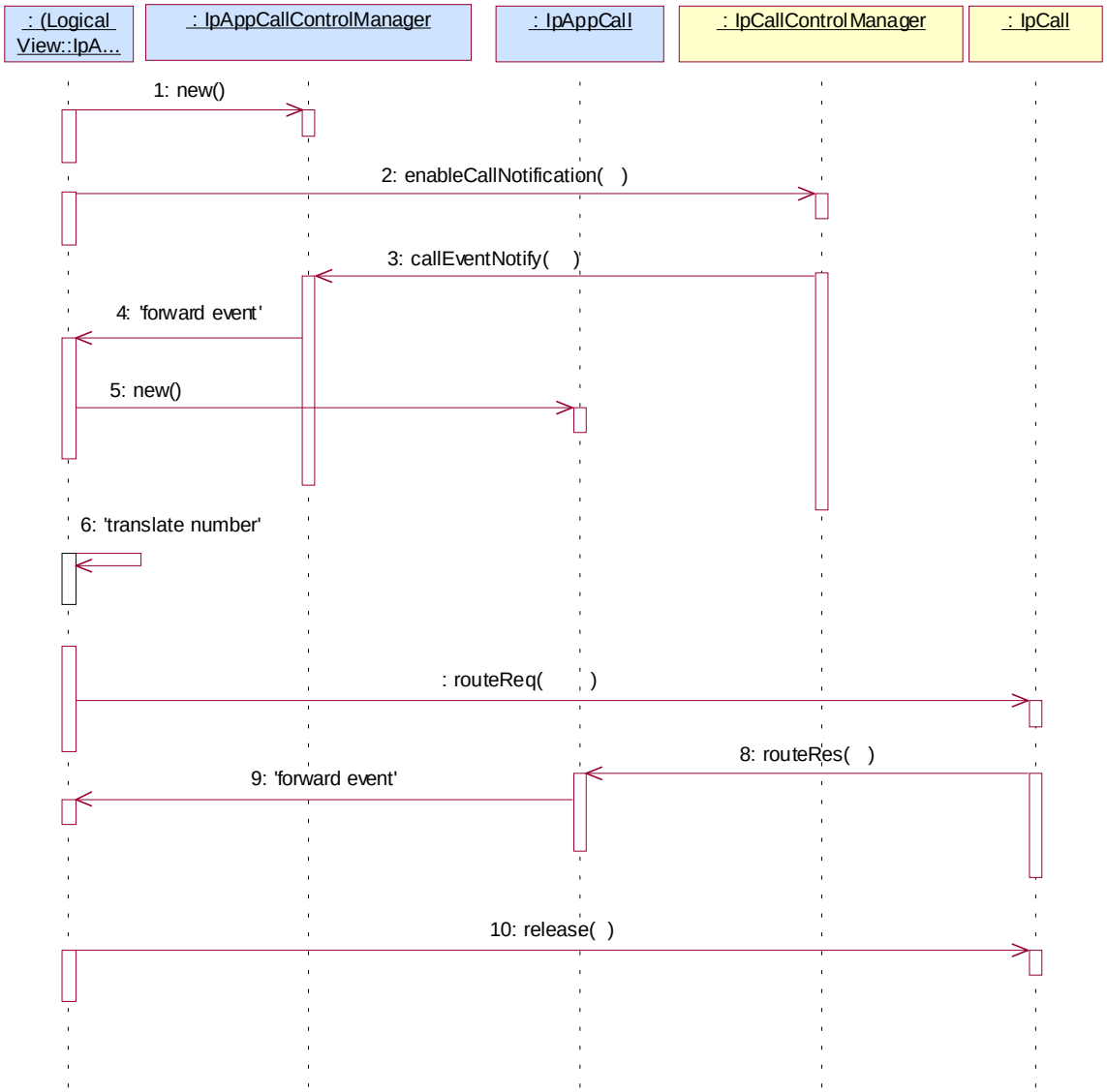
10: This message passes the result of the call being answered to its callback object

11: This message is used to forward the previous message to the IpAppLogic.

12: The application is no longer interested in controlling the call and therefore deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

6.1.7 Number Translation 2

The following sequence diagram shows a number translation service, initiated as a result of a prearranged event being received by the framework. If the translated number being routed to does not answer or is busy then the call is automatically released.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward the previous message to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.

6: This message invokes the number translation function.

7: The returned translated number is used to route the call towards the destination.

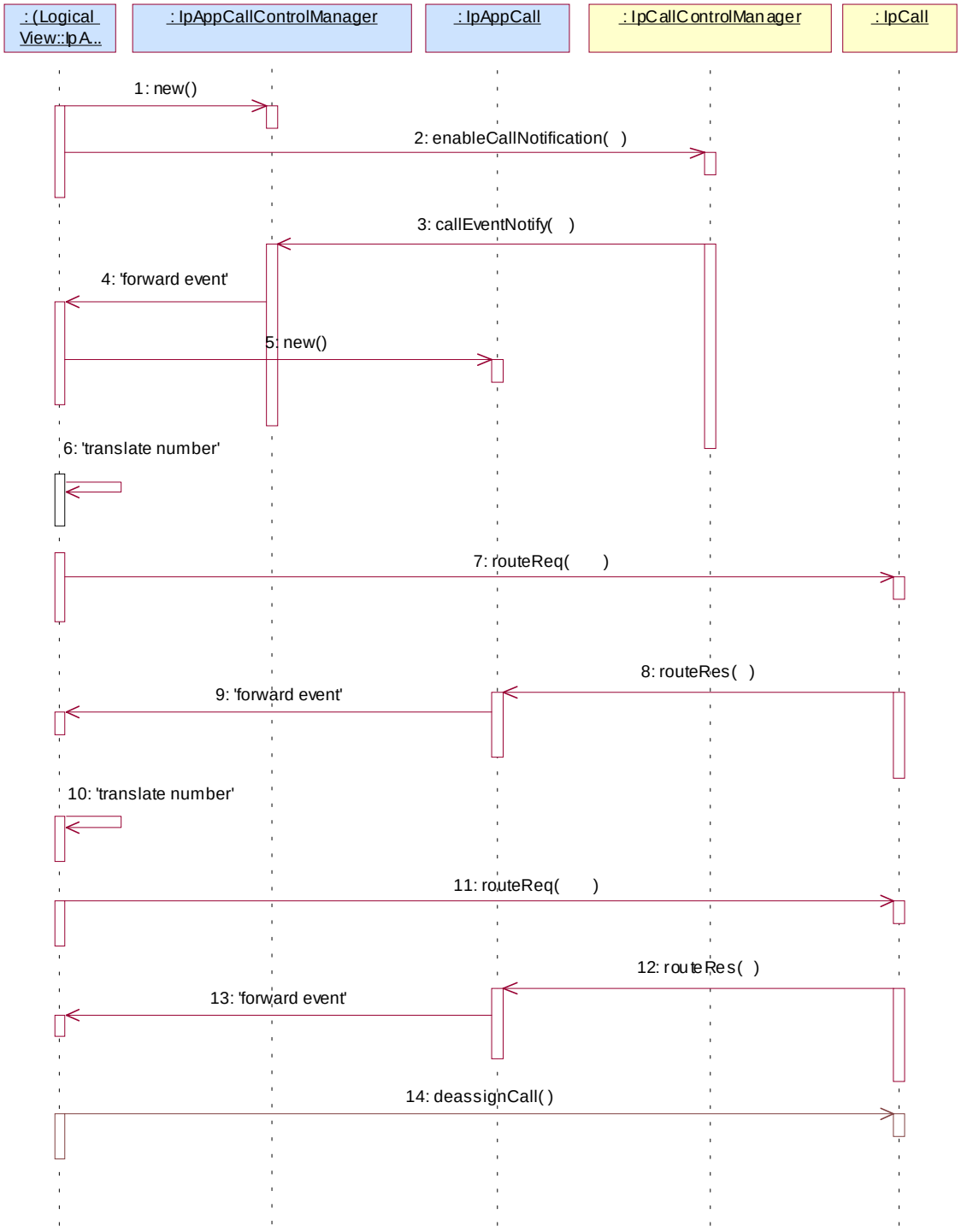
8: Assuming the called party is busy or does not answer, the object implementing the IpCall interface sends a callback in this message, indicating the unavailability of the called party.

9: This message is used to forward the previous message to the IpAppLogic.

10: The application takes the decision to release the call.

6.1.8 Number Translation 3

The following sequence diagram shows a number translation service, initiated as a result of a prearranged event being received by the framework. If the translated number being routed to does not answer or is busy then the call is automatically routed to a voice mailbox.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load

control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward the previous message to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.

6: This message invokes the number translation function.

7: The returned translated number is used to route the call towards the destination.

8: Assuming the called party is busy or does not answer, the object implementing the IpCall interface sends a callback, indicating the unavailability of the called party.

9: This message is used to forward the previous message to the IpAppLogic.

10: The application takes the decision to translate the number, but this time the number is translated to a number belonging to a voice mailbox system.

11: This message routes the call towards the voice mailbox.

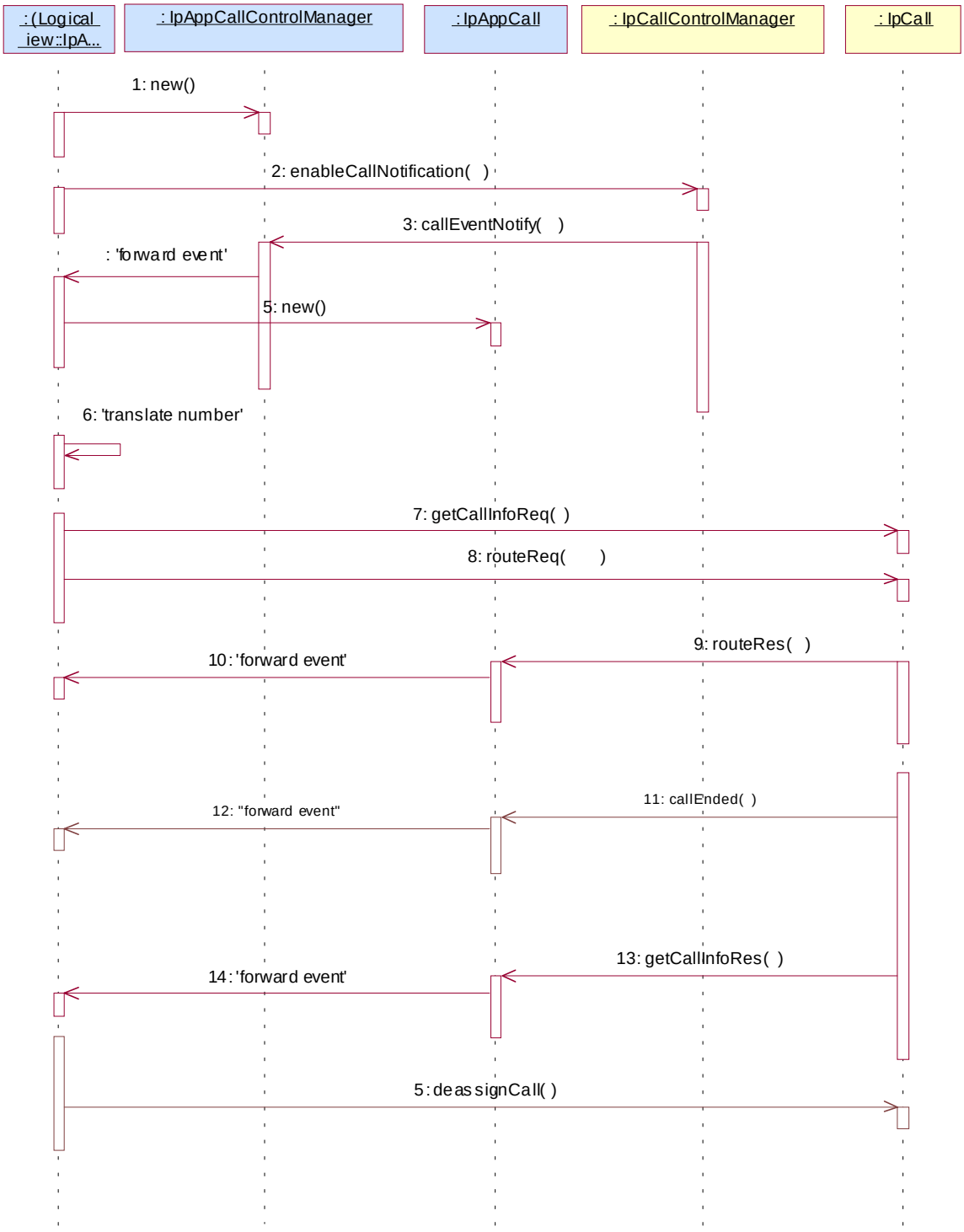
12: This message passes the result of the call being answered to its callback object.

13: This message is used to forward the previous message to the IpAppLogic.

14: The application is no longer interested in controlling the call and therefore deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

6.1.9 Number Translation 4

The following sequence diagram shows a number translation service, initiated as a result of a prearranged event being received by the framework. Before the call is routed to the translated number, the application requests for all call related information to be delivered back to the application on completion of the call.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load

control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward the previous message to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.

6: This message invokes the number translation function.

7: The application instructs the object implementing the IpCall interface to return all call related information once the call has been released.

8: The returned translated number is used to route the call towards the destination.

9: This message passes the result of the call being answered to its callback object.

10: This message is used to forward the previous message to the IpAppLogic.

11: Towards the end of the call, when one of the parties disconnects, a message (not shown) is directed to the object implementing the IpCall. This causes an event, to be passed to the object implementing the IpAppCall object.

12: This message is used to forward the previous message to the IpAppLogic.

13: The application now waits for the call information to be sent. Now that the call has completed, the object implementing the IpCall interface passes the call information to its callback object.

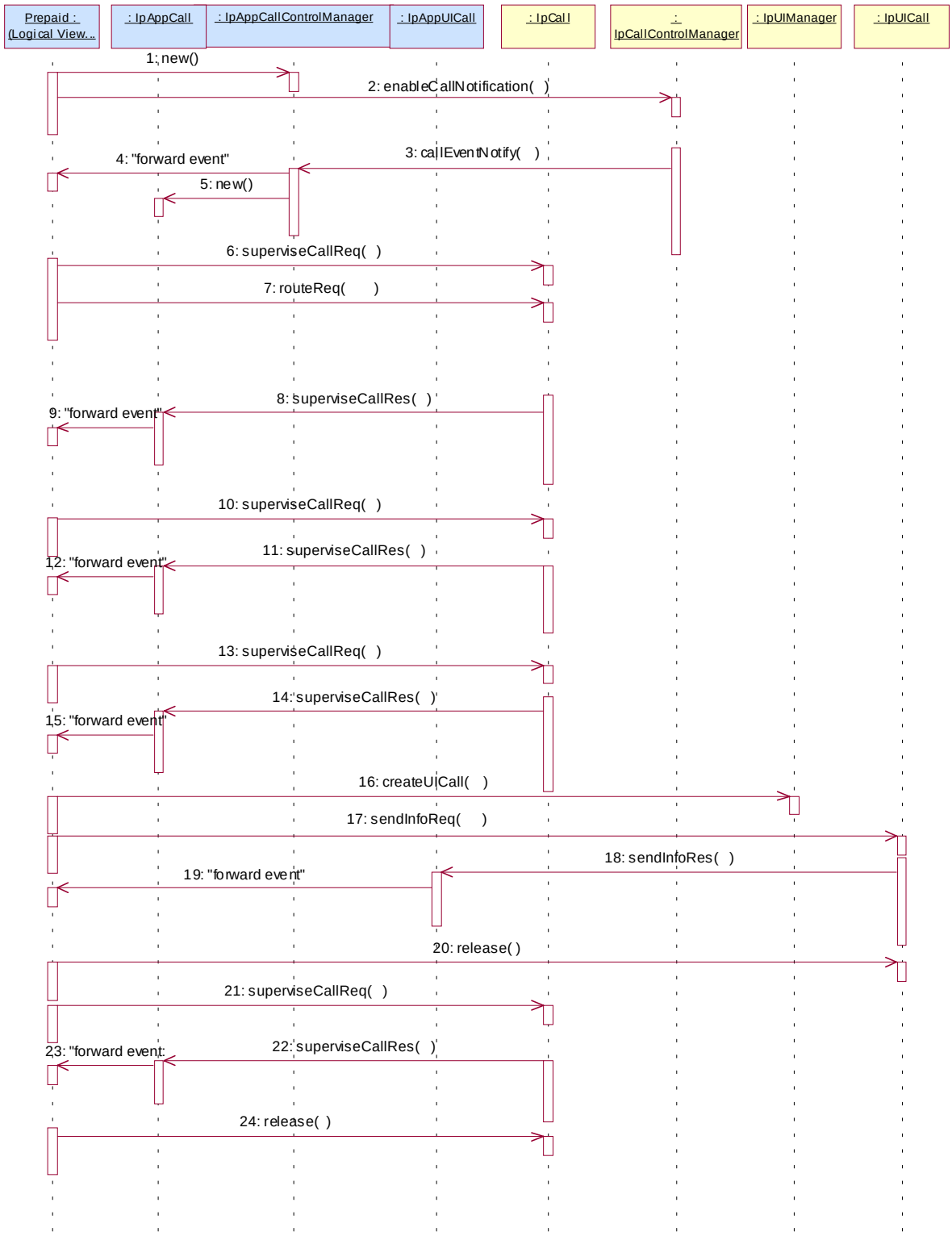
14: This message is used to forward the previous message to the IpAppLogic

15: After the last information is received, the application deassigns the call. This will free the resources related to this call in the gateway.

6.1.10 Prepaid

This sequence shows a Pre-paid application.

The subscriber is using a pre-paid card or credit card to pay for the call. The application each time allows a certain timeslice for the call. After the timeslice, a new timeslice can be started or the application can terminate the call. In the following sequence the end-user will received an announcement before his final timeslice.



1: This message is used by the application to create an object implementing the IpAppGenericCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a pre-paid service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: The incoming call triggers the Pre-Paid Application (PPA).

4: The message is forwarded to the application.

5: A new object on the application side for the Generic Call object is created

6: The Pre-Paid Application (PPA) requests to supervise the call. The application will be informed after the period indicated in the message. This period is related to the credits left on the account of the pre-paid subscriber.

7: Before continuation of the call, PPA sends all charging information, a possible tariff switch time and the call duration supervision period, towards the GW which forwards it to the network.

8: At the end of each supervision period the application is informed and a new period is started.

9: The message is forwarded to the application.

10: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

11: At the end of each supervision period the application is informed and a new period is started.

12: The message is forwarded to the application.

13: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

14: When the user is almost out of credit an announcement is played to inform about this. The announcement is played only to the leg of the A-party, the B-party will not hear the announcement.

15: The message is forwarded to the application.

16: A new UICall object is created and associated with the controlling leg.

17: An announcement is played to the controlling leg informing the user about the near-expiration of his credit limit. The B-subscriber will not hear the announcement.

18: When the announcement is completed the application is informed.

19: The message is forwarded to the application.

20: The application releases the UICall object.

21: The user does not terminate so the application terminates the call after the next supervision period.

22: The supervision period ends

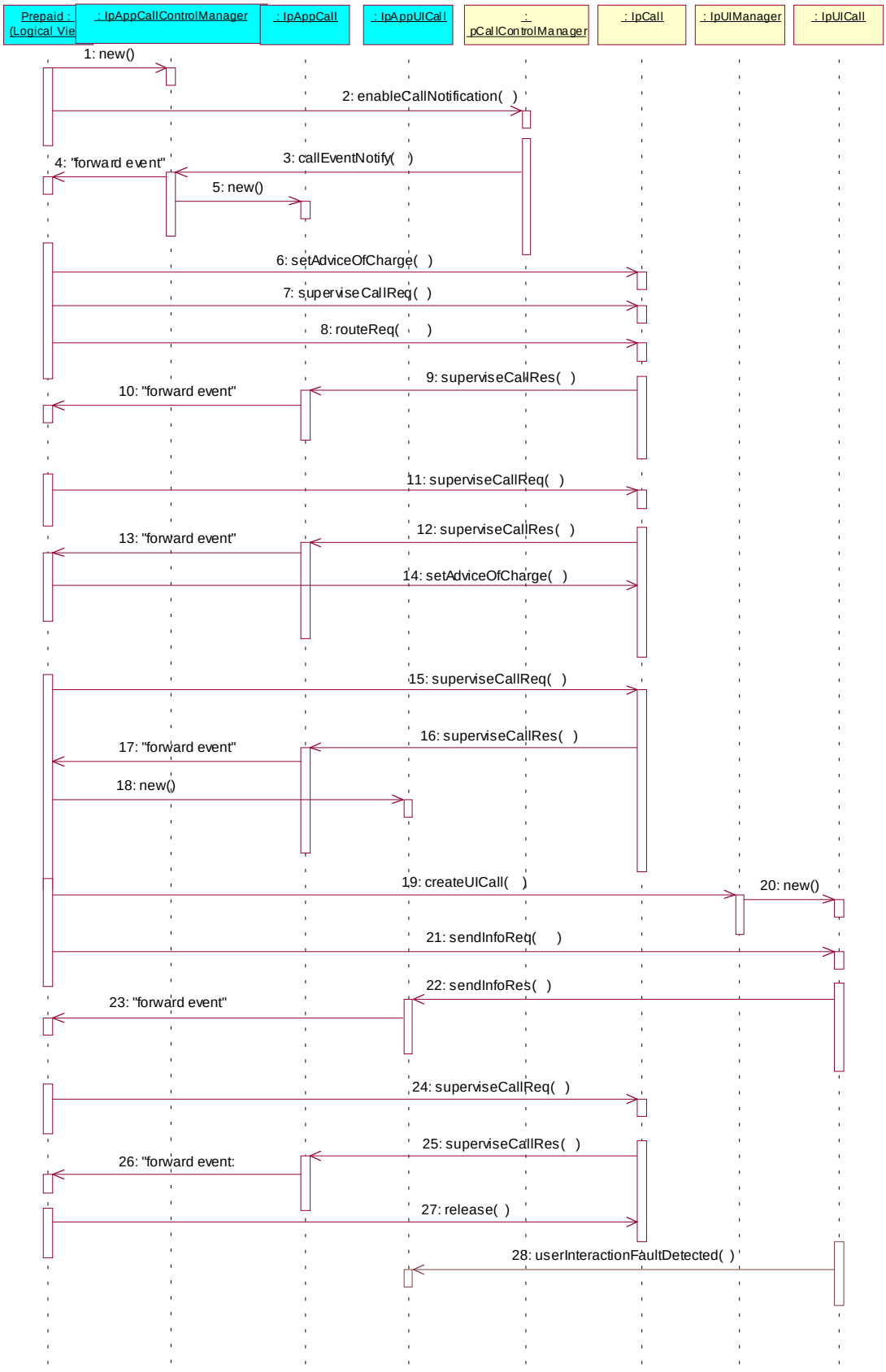
23: The event is forwarded to the logic.

24: The application terminates the call. Since the user interaction is already explicitly terminated no userInteractionFaultDetected is sent to the application.

6.1.11 Pre-Paid with Advice of Charge (AoC)

This sequence shows a Pre-paid application that uses the Advice of Charge feature.

The application will send the charging information before the actual call setup and when during the call the charging changes new information is sent in order to update the end-user. Note: the Advice of Charge feature requires an application in the end-user terminal to display the charges for the call, depending on the information received from the application.


Prepaid : (Logical Vie
: IpAppCallControlManager
: IpAppCall
: IpAppUICall
: pCallControlManager
: IpCall
: IpUIManager
: IpUICall
1: new()
2: enableCallNotification( )
3: callEventNotify( )
4: "forward event"
5: new()
6: setAdviceOfCharge( )
7: superviseCallReq( )
8: routeReq( )
9: superviseCallRes( )
10: "forward event"
11: superviseCallReq( )
12: superviseCallRes( )
13: "forward event"
14: setAdviceOfCharge( )
15: superviseCallReq( )
16: superviseCallRes( )
17: "forward event"
18: new()
19: createUICall( )
20: new()
21: sendInfoReq( )
22: sendInfoRes( )
23: "forward event"
24: superviseCallReq( )
25: superviseCallRes( )
26: "forward event:
27: release( )
28: userInteractionFaultDetected( )

1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a pre-paid service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: The incoming call triggers the Pre-Paid Application (PPA).

4: The message is forwarded to the application.

5: A new object on the application side for the Call object is created

6: The Pre-Paid Application (PPA) sends the AoC information (e.g the tariff switch time). (it shall be noted the PPA contains ALL the tariff information and knows how to charge the user).

During this call sequence 2 tariff changes take place. The call starts with tariff 1, and at the tariff switch time (e.g., 18:00 hours) switches to tariff 2. The application is not informed about this (but the end-user is!)

7: The Pre-Paid Application (PPA) requests to supervise the call. The application will be informed after the period indicated in the message. This period is related to the credits left on the account of the pre-paid subscriber.

8: The application requests to route the call to the destination address.

9: At the end of each supervision period the application is informed and a new period is started.

10: The message is forwarded to the application.

11: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

12: At the end of each supervision period the application is informed and a new period is started.

13: The message is forwarded to the application.

14: Before the next tariff switch (e.g., 19:00 hours) the application sends a new AOC with the tarif switch time. Again, at the tariff switch time,the network will send AoC information to the end-user.

15: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

16: When the user is almost out of credit an announcement is played to inform about this (19-21). The announcement is played only to the leg of the A-party, the B-party will not hear the announcement.

17: The message is forwarded to the application.

18: The application creates a new call back interface for the User interaction messages.

19: A new UI Call object that will handle playing of the announcement needs to be created

20: The Gateway creates a new UI call object that will handle playing of the announcement.

21: With this message the announcement is played to the calling party.

22: The user indicates that the call should continue.

23: The message is forwarded to the application.

24: The user does not terminate so the application terminates the call after the next supervision period.

25: The user is out of credit and the application is informed.

26: The message is forwarded to the application.

27: With this message the application requests to release the call.

28: Terminating the call which has still a UICall object associated will result in a userInteractionFaultDetected. The UICall object is terminated in the gateway and no further communication is possible between the UICall and the application.

6.2 Class Diagrams

The generic call control service consists of two packages, one for the interfaces on the application side and one for interfaces on the service side.

The class diagrams in the following figures show the interfaces that make up the generic call control application package and the generic call control service package. Communication between these packages is indicated with the <<uses>> associations; e.g., the IpCallControlManager interface uses the IpAppGenericCallControlManager , by means of calling callback methods.

This class diagram shows the interfaces of the generic call control application package and their relations to the interfaces of the generic call control service package.

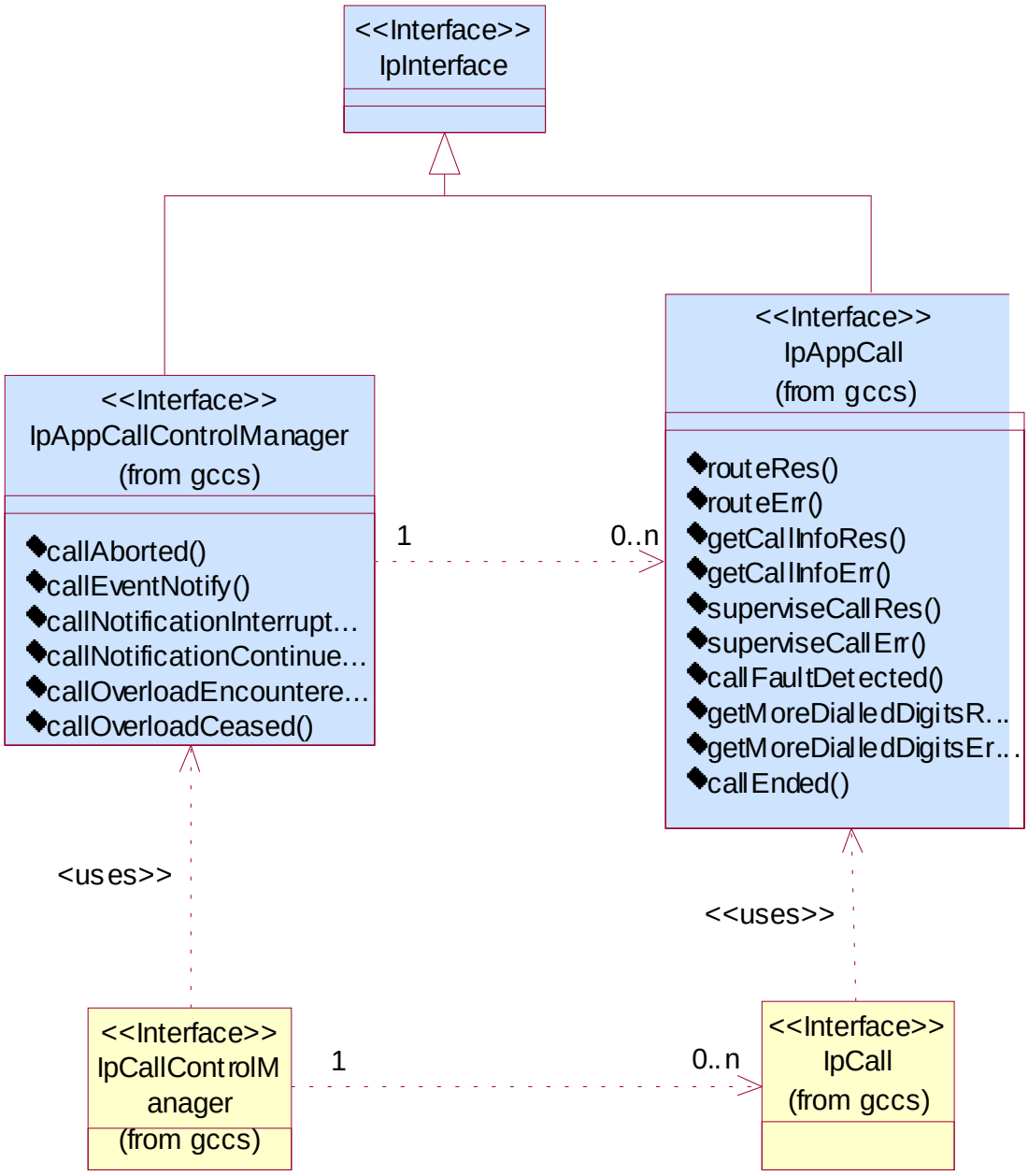


Figure: Application Interfaces

This class diagram shows the interfaces of the generic call control service package.

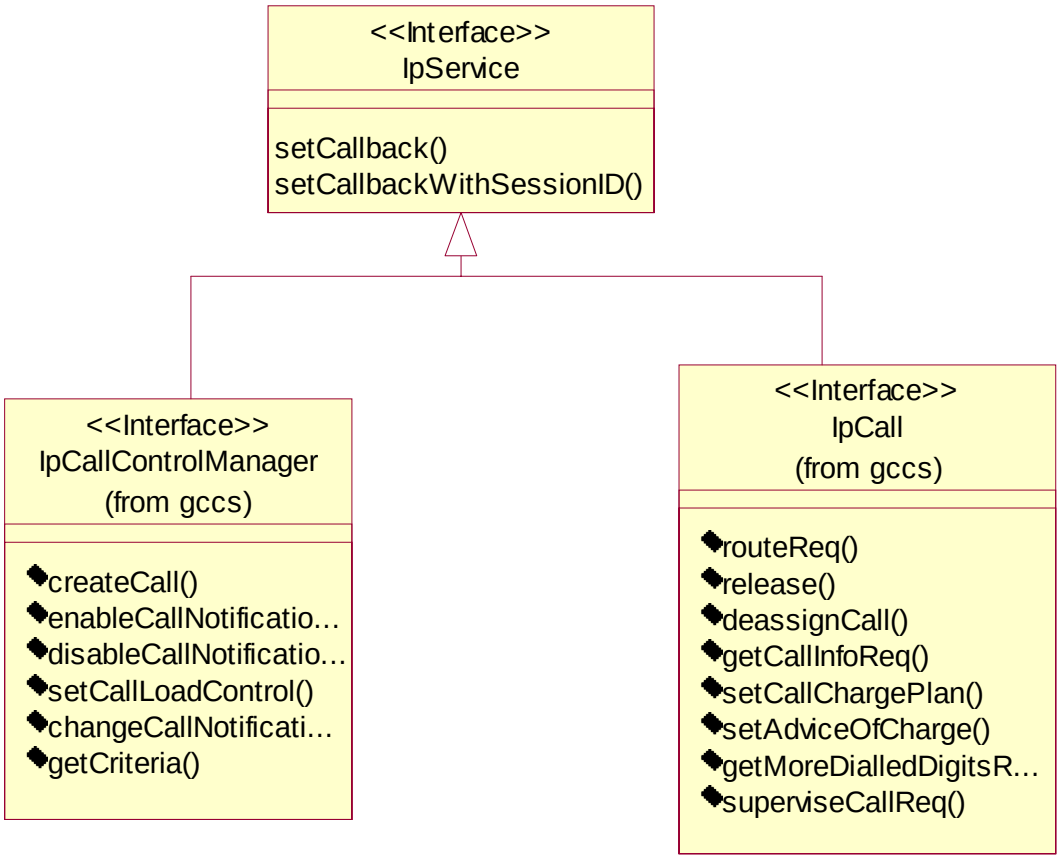


Figure: Service Interfaces

6.3 Generic Call Control Service Interface Classes

The Generic Call Control Service (GCCS) provides the basic call control service for the API. It is based around a third party model, which allows calls to be instantiated from the network and routed through the network.

The GCCS supports enough functionality to allow call routing and call management for today's Intelligent Network (IN) services in the case of a switched telephony network, or equivalent for packet based networks.

It is the intention of the GCCS that it could be readily specialised into call control specifications, for example, ITU-T recommendations H.323, ISUP, Q.931 and Q.2931, ATM Forum specification UNI3.1 and the IETF Session Initiation Protocol, or any other call control technology.

The adopted call model has the following objects. Note that not all of these concepts are used in the generic call.

* a call object. A call is a relation between a number of parties. The call object relates to the entire call view from the application. E.g., the entire call will be released when a release is called on the call. Note that different applications can have different views on the same physical call, e.g., one application for the originating side and another application for the terminating side. The applications will not be aware of each other, all 'communication' between the applications will be by means of network signalling. The API currently does not specify any feature interaction mechanisms.

* a call leg object. The leg object represents a logical association between a call and an address. The relationship includes at least the signalling relation with the party. The relation with the address is only made when the leg is routed. Before that the leg object is IDLE and not yet associated with the address.

* an address. The address logically represents a party in the call.

* a terminal. A terminal is the end-point of the signalling and/or media for a party. This object type is currently not addressed.

The call object is used to establish a relation between a number of parties by creating a leg for each party within the call.

Associated with the signalling relationship represented by the call leg, there may also be a bearer connection (e.g., in the traditional voice only networks) or a number (zero or more) of media channels (in multi-media networks).

A leg can be attached to the call or detached from the call. When the leg is attached, this means that media or bearer channels related to the legs are connected to the media or bearer channels of the other legs that are attached to the same call. I.e., only legs that are attached can 'speak' to each other. A leg can have a number of states, depending on the signalling received from or sent to the party associated with the leg. Usually there is a limit to the number of legs that are in being routed (i.e., the connection is being established) or connected to the call (i.e., the connection is established). Also, there usually is a limit to the number of legs that can be simultaneously attached to the same call.

Some networks distinguish between controlling and passive legs. By definition the call will be released when the controlling leg is released. All other legs are called passive legs. There can be at most one controlling leg per call. However, there is currently no way the application can influence whether a Leg is controlling or not.

There are two ways for an application to get the control of a call. The application can request to be notified of calls that meet certain criteria. When a call occurs in the network that meets these criteria, the application is notified and can control the call. Some legs will already be associated with the call in this case. Another way is to create a new call from the application.

For the generic call control service, only a subset of the model is used; the API for generic call control does not give explicit access to the legs and the media channels. This is provided by the Multi-Party Call Control Service. Furthermore, the generic call is restricted to two party calls, i.e., only two legs are active at any given time. Active is defined here as 'being routed' or connected.

The GCCS is represented by the IpCallManager and IpCall interfaces that interface to services provided by the network. Some methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppCallManager and IpAppCall to provide the callback mechanism.

6.3.1 Interface Class IpCallControlManager

Inherits from: IpService

This interface is the 'service manager' interface for the Generic Call Control Service. The generic call control manager interface provides the management functions to the generic call control service. The application programmer can use this interface to provide overload control functionality, create call objects and to enable or disable call-related event notifications.

<<Interface>> IpCallControlManager
createCall (appCall : in IpAppCallRef, callReference : out TpCallIdentifierRef) : TpResult enableCallNotification (appCallControlManager : in IpAppCallControlManagerRef, eventCriteria : in TpCallEventCriteria, assignmentID : out TpAssignmentIDRef) : TpResult disableCallNotification (assignmentID : in TpAssignmentID) : TpResult setCallLoadControl (duration : in TpDuration, mechanism : in TpCallLoadControlMechanism, treatment : in TpCallTreatment, addressRange : in TpAddressRange, assignmentID : out TpAssignmentIDRef) : TpResult

changeCallNotification (assignmentID : in TpAssignmentID, eventCriteria : in TpCallEventCriteria) : TpResult

getCriteria (eventCriteria : out TpCallEventCriteriaResultSetRef) : TpResult

Method

createCall()

This method is used to create a new call object.

Parameters

appCall : in IpAppCallRef

Specifies the application interface for callbacks from the call created.

callReference : out TpCallIdentifierRef

Specifies the interface reference and sessionID of the call created.

Raises

TpGCCSException,TpGeneralException

Method

enableCallNotification()

This method is used to enable call notifications so that events can be sent to the application. If some application already requested notifications with criteria that overlap the specified criteria, the request is refused with P_GCCS_INVALID_CRITERIA.

The criteria are said to overlap if both originating and terminating ranges overlap and the same number plan is used and the same CallNotificationType is used.

If the same application requests two notifications with exactly the same criteria but different callback references, the second callback will be treated as an additional callback. This means that the callback will only be used in case when the first callback specified by the application is unable to handle the callEventNotify (e.g., due to overload or failure).

Parameters

appCallControlManager : in IpAppCallControlManagerRef

If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the setCallback() method.

eventCriteria : in TpCallEventCriteria

Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported. Examples of events are "incoming call attempt reported by network", "answer", "no answer", "busy". Individual addresses or address ranges may be specified for destination and/or origination.

assignmentID : out TpAssignmentIDRef

Specifies the ID assigned by the generic call control manager interface for this newly-enabled event notification.

Raises

TpGCCSException,TpGeneralException

Method

disableCallNotification()

This method is used by the application to disable call notifications.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID given by the generic call control manager interface when the previous enableNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will return the error code P_INVALID_ASSIGNMENTID.

Raises

TpGCCSException,TpGeneralException

Method

setCallLoadControl()

This method imposes or removes load control on calls made to a particular address range within the generic call control service. The address matching mechanism is similar as defined for TpCallEventCriteria.

Parameters

duration : in TpDuration

Specifies the duration for which the load control should be set.

A duration of 0 indicates that the load control should be removed.

A duration of -1 indicates an infinite duration (i.e., until disabled by the application)

A duration of -2 indicates the network default duration.

mechanism : in TpCallLoadControlMechanism

Specifies the load control mechanism to use (for example, admit one call per interval), and any necessary parameters, such as the call admission rate. The contents of this parameter are ignored if the load control duration is set to zero.

treatment : in TpCallTreatment

Specifies the treatment of calls that are not admitted. The contents of this parameter are ignored if the load control duration is set to zero.

addressRange : in TpAddressRange

Specifies the address or address range to which the overload control should be applied or removed.

assignmentID : out TpAssignmentIDRef

Specifies the assignmentID assigned by the gateway to this request. This assignementID can be used to correlate the callOverlloadEncountered and callOverloadCeased methods with the request.

Raises

TpGeneralException,TpGCCSException

Method

changeCallNotification()

This method is used by the application to change the event criteria introduced with enableCallNotification. Any stored criteria associated with the specified assignementID will be replaced with the specified criteria.

Parameters

assignmentID : in TpAssignmentID

Specifies the ID assigned by the generic call control manager interface for the event notification.

eventCriteria : in TpCallEventCriteria

Specifies the new set of event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

Raises

TpGeneralException,TpGCCSException

Method

getCriteria()

This method is used by the application to query the event criteria set with enableCallNotification or changeCallNotification.

Parameters

eventCriteria : out TpCallEventCriteriaResultSetRef

Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

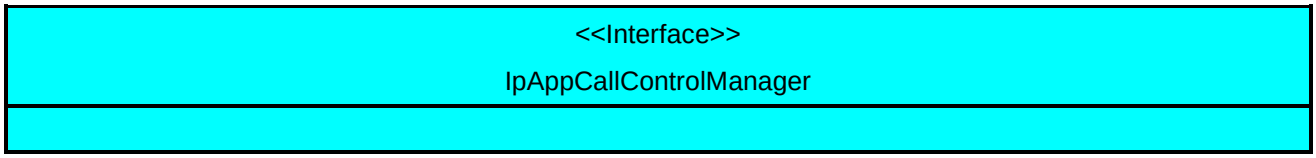
Raises

TpGeneralException,TpGCCSException

6.3.2 Interface Class IpAppCallControlManager

Inherits from: IpInterface

The generic call control manager application interface provides the application call control management functions to the generic call control service.



callAborted (callReference : in TpSessionID) : TpResult

callEventNotify (callReference : in TpCallIdentifier, eventInfo : in TpCallEventInfo, assignmentID : in TpAssignmentID, appCall : out IpAppCallRefRef) : TpResult

callNotificationInterrupted () : TpResult

callNotificationContinued () : TpResult

callOverloadEncountered (assignmentID : in TpAssignmentID) : TpResult

callOverloadCeased (assignmentID : in TpAssignmentID) : TpResult

Method

callAborted()

This method indicates to the application that the call object (at the gateway) has aborted or terminated abnormally. No further communication will be possible between the call and application.

Parameters

callReference : in TpSessionID

Specifies the sessionID of call that has aborted or terminated abnormally.

Raises

TpGCCSException,TpGeneralException

Method

callEventNotify()

This method notifies the application of the arrival of a call-related event.

Parameters

callReference : in TpCallIdentifier

Specifies the reference to the call interface to which the notification relates.

eventInfo : in TpCallEventInfo

Specifies data associated with this event.

assignmentID : in TpAssignmentID

Specifies the assignment id which was returned by the enableNotification() method. The application can use assignment id to associate events with event specific criteria and to act accordingly.

appCall : out IpAppCallRefRef

Specifies a reference to the application interface which implements the callback interface for the new call.

Raises

TpGCCSException,TpGeneralException

Method

callNotificationInterrupted()

This method indicates to the application that all event notifications have been temporary interrupted (for example, due to faults detected).

Note that more permanent failures are reported via the Framework (integrity management).

Parameters

No Parameters were identified for this method

Raises

TpGCCSException,TpGeneralException

Method

callNotificationContinued()

This method indicates to the application that event notifications will again be possible.

Parameters

No Parameters were identified for this method

Method

callOverloadEncountered()

This method indicates that the network has detected overload and may have automatically imposed load control on calls requested to a particular address range or calls made to a particular destination within the call control service.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been encountered.

Raises

TpGeneralException,TpGCCSException

Method

callOverloadCeased()

This method indicates that the network has detected that the overload has ceased and has automatically removed any load controls on calls requested to a particular address range or calls made to a particular destination within the call control service.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been ceased

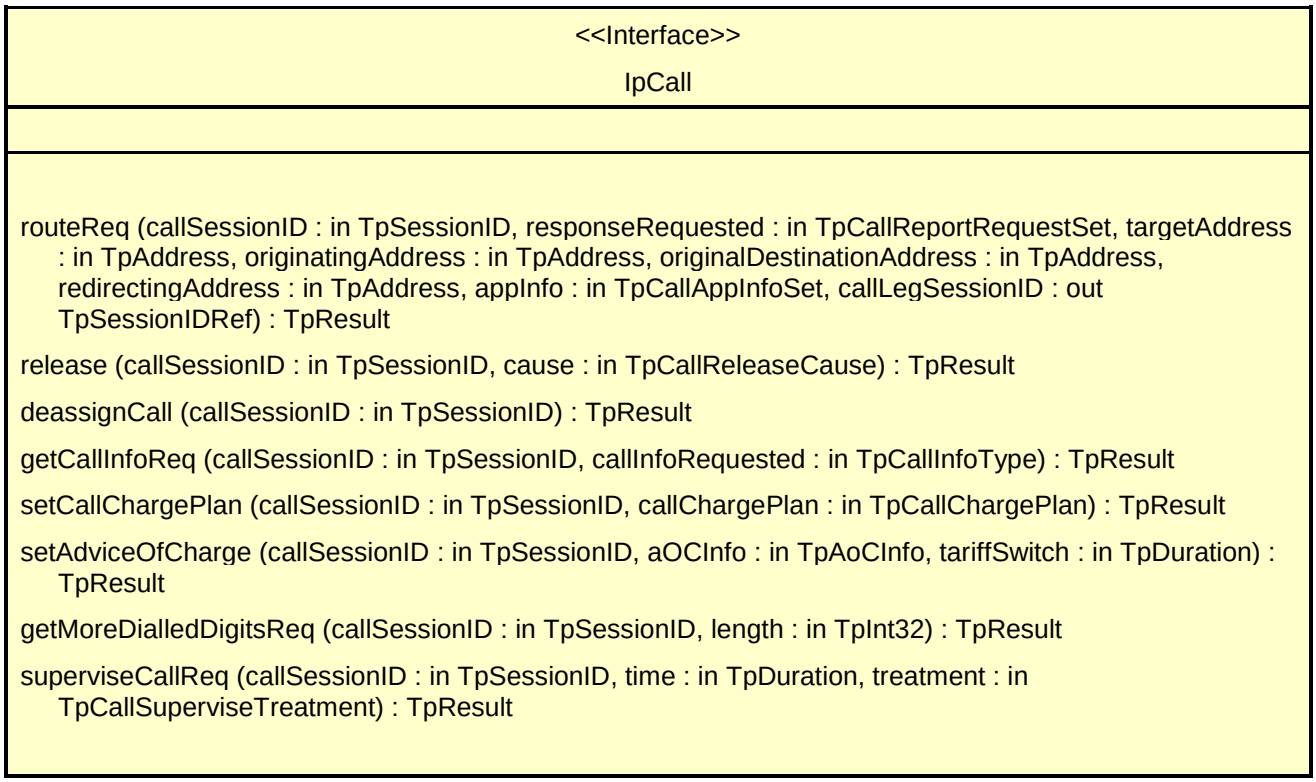
Raises

TpGeneralException,TpGCCSException

6.3.3 Interface Class IpCall

Inherits from: IpService

The generic Call provides the possibility to control the call routing, to request information from the call, control the charging of the call, to release the call and to supervise the call. It does not give the possibility to control the legs directly and it does not allow control over the media. The first capability is provided by the multi-party call and the latter as well by the multi-media call. The call is limited to two party calls, although it is possible to provide 'follow-on' calls, meaning that the call can be rerouted after the terminating party has disconnected or routing to the terminating party has failed. Basically, this means that at most two legs can be in connected or routing state at any time.



Method

routeReq()

This asynchronous method requests routing of the call (and inherently attached parties) to the destination party, via a new call leg (which is implicitly created).

The extra address information (i.e., originalDestinationAddress, redirectingAddress, originatingAddress) is optional. If not present (i.e., the plan is set to P_ADDRESS_PLAN_NOT_PRESENT), the information provided in corresponding addresses from the route is used, otherwise the network or gateway provided numbers will be used.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

responseRequested : in TpCallReportRequestSet

Specifies the set of observed events that will result in zero or more routeRes() being generated.

E.g., when both answer and disconnect is monitored the result can be received two times.

If the application wants to control the call (in whatever sense) it shall enable event reports

targetAddress : in TpAddress

Specifies the destination party to which the call should be routed.

originatingAddress : in TpAddress

Specifies the address of the originating (calling) party.

originalDestinationAddress : in TpAddress

Specifies the original destination address of the call.

redirectingAddress : in TpAddress

Specifies the address from which the call was last redirected.

appInfo : in TpCallAppInfoSet

Specifies application-related information pertinent to the call (such as alerting method, tele-service type, service identities and interaction indicators).

callLegSessionID : out TpSessionIDRef

Specifies the sessionID assigned by the gateway. This is the sessionID of the implicitly created call leg. The same ID will be returned in the routeRes or Err. This allows the application to correlate the request and the result.

This parameter is only relevant when multiple routeReq() calls are executed in parallel, e.g., in the multi-party call control service.

Raises

TpGCCSException,TpGeneralException

Method

release()

This method requests the release of the call object and associated objects. The call will also be terminated in the network. If the application requested reports to be sent at the end of the call (e.g., by means of getCallInfoReq) these reports will still be sent to the application.

The application should always either release or deassign the call when it is finished with the call, unless a callFaultDetected is received by the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

cause : in TpCallReleaseCause

Specifies the cause of the release.

Raises

TpGCCSException,TpGeneralException

Method

deassignCall()

This method requests that the relationship between the application and the call and associated objects be de-assigned. It leaves the call in progress, however, it purges the specified call object so that the application has no further control of call processing. If a call is de-assigned that has event reports, call information reports or call Leg information reports requested, then these reports will be disabled and any related information discarded.

The application should always either release or deassign the call when it is finished with the call, unless callFaultDetected is received by the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

Raises

TpGCCSException,TpGeneralException

Method

getCallInfoReq()

This asynchronous method requests information associated with the call to be provided at the appropriate time (for example, to calculate charging). This method must be invoked before the call is routed to a target address. Two types of reports can be requested; a final report or intermediate reports.

A final call report is sent when the call is ended. The call object will exist after the call is ended if information is required to be sent to the application at the end of the call. The call information will be sent after any call event reports.

Intermediate reports are received when the destination leg or party terminates or when the call ends. In case the originating party is still available the application can still initiate a follow-on call using routeReq.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callInfoRequested : in TpCallInfoType

Specifies the call information that is requested.

Raises

TpGCCSException,TpGeneralException

Method

setCallChargePlan()

Set an operator specific charge plan for the call. The charge plan must be set before the call is routed to a target address. Depending on the operator the method can also be used to change the charge plan for ongoing calls.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callChargePlan : in TpCallChargePlan

Specifies the charge plan to use.

Raises

TpGCCSException,TpGeneralException

Method

setAdviceOfCharge()

This method allows for advice of charge (AOC) information to be sent to terminals that are capable of receiving this information.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

aOCInfo : in TpAoCInfo

Specifies two sets of Advice of Charge parameter.

tariffSwitch : in TpDuration

Specifies the tariff switch interval that signifies when the second set of AoC parameters becomes valid.

Raises

TpGeneralException,TpGCCSException

Method

getMoreDialledDigitsReq()

This asynchronous method requests the call control service to collect further digits and return them to the application. Depending on the administered data, the network may indicate a new call to the gateway if a caller goes off-hook or dialled only a few digits. The application then gets a new call event which contains no digits or only the few dialled digits in the event data.

The application should use this method if it requires more dialled digits, e.g. to perform screening.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

length : in TpInt32

Specifies the maximum number of digits to collect.

Raises

TpGeneralException, TpGCCSException

Method

superviseCallReq()

The application calls this method to supervise a call. The application can set a granted connection time for this call. If an application calls this function before it calls a routeReq() or a user interaction function the time measurement will start as soon as the call is answered by the B-party or the user interaction system.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

time : in TpDuration

Specifies the granted time in milliseconds for the connection.

treatment : in TpCallSuperviseTreatment

Specifies how the network should react after the granted connection time expired.

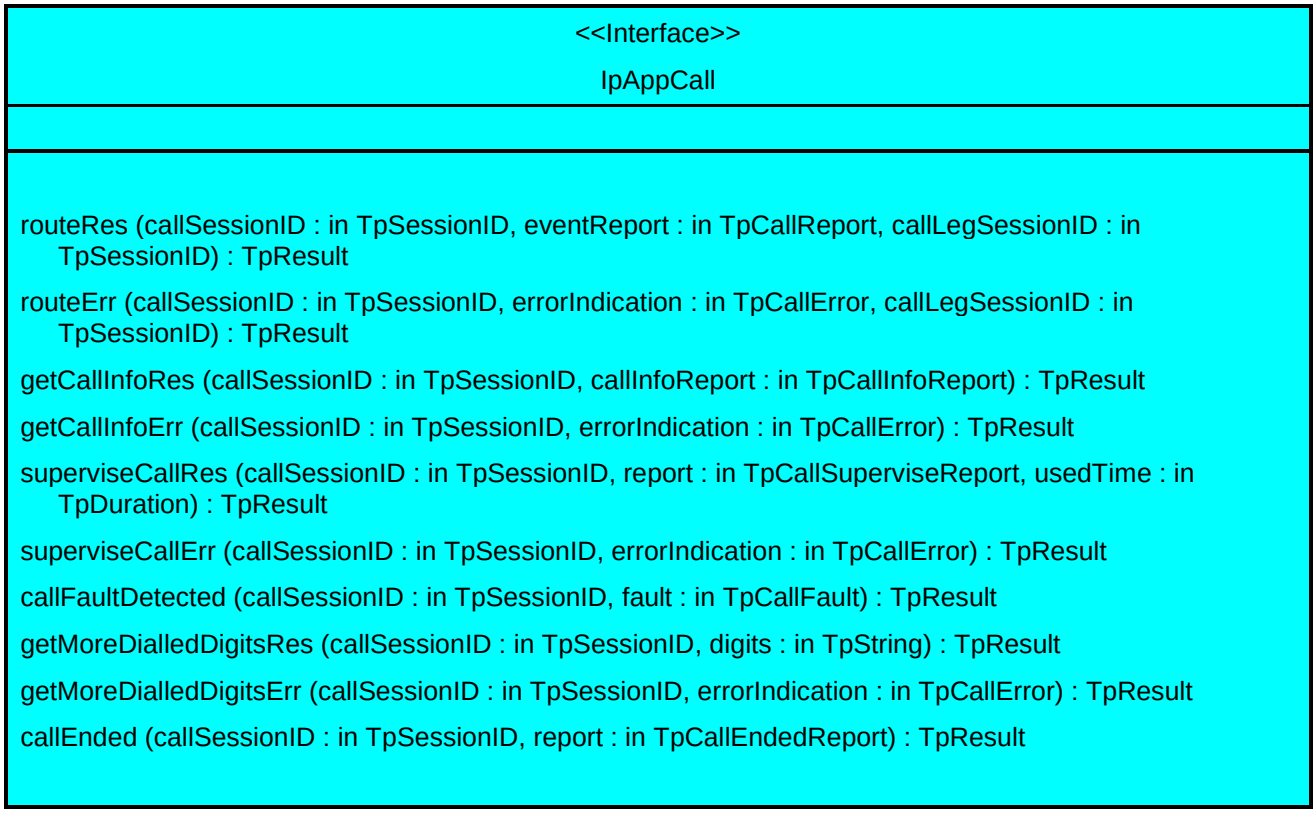
Raises

TpGCCSException,TpGeneralException

6.3.4 Interface Class IpAppCall

Inherits from: IpInterface

The generic call application interface is implemented by the client application developer and is used to handle call request responses and state reports.



Method

`routeRes()`

This asynchronous method indicates that the request to route the call to the destination was successful, and indicates the response of the destination party (for example, the call was answered, not answered, refused due to busy, etc.).

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`eventReport : in TpCallReport`

Specifies the result of the request to route the call to the destination party. It also includes the network event, date and time, monitoring mode and event specific information such as release cause.

`callLegSessionID : in TpSessionID`

Specifies the sessionID of the associated call leg. This corresponds to the sesion ID returned at the routeReq() and can be used to correlate the response with the request.

Raises

TpGCCSException,TpGeneralException

Method

routeErr()

This asynchronous method indicates that the request to route the call to the destination party was unsuccessful - the call could not be routed to the destination party (for example, the network was unable to route the call, the parameters were incorrect, the request was refused, etc.).

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

callLegSessionID : in TpSessionID

Specifies the sessionID of the associated call leg. This corresponds to the sessionID returned at the routeReq() and can be used to correlate the error with the request.

Raises

TpGCCSException,TpGeneralException

Method

getCallInfoRes()

This asynchronous method reports time information of the finished call or call attempt as well as release cause depending on which information has been requested by getCallInfoReq. This information may be used e.g. for charging purposes. The call information will possibly be sent after routeRes in all cases where the call or a leg of the call has been disconnected or a routing failure has been encountered.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callInfoReport : in TpCallInfoReport

Specifies the call information requested.

Raises

TpGCCSException,TpGeneralException

Method
getCallInfoErr()

This asynchronous method reports that the original request was erroneous, or resulted in an error condition.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGCCSException,TpGeneralException

Method
superviseCallRes()

This asynchronous method reports a call supervision event to the application when it has indicated it's interest in these kind of events.

It is also called when the connection is terminated before the supervision event occurs. Furthermore, this method is invoked as a response to the request also when a tariff switch happens in the network during an active call.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call

report : in TpCallSuperviseReport

Specifies the situation which triggered the sending of the call supervision response.

usedTime : in TpDuration

Specifies the used time for the call supervision (in milliseconds).

Raises

TpGCCSException,TpGeneralException

Method
superviseCallErr()

This asynchronous method reports a call supervision error to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGCCSException,TpGeneralException

Method

callFaultDetected()

This method indicates to the application that a fault in the network has been detected. The call may or may not have been terminated.

The system deletes the call object. Therefore, the application has no further control of call processing. No report will be forwarded to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call in which the fault has been detected.

fault : in TpCallFault

Specifies the fault that has been detected.

Raises

TpGCCSException,TpGeneralException

Method

getMoreDialledDigitsRes()

This asynchronous method returns the collected digits to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

digits : in TpString

Specifies the additional dialled digits if the string length is greater than zero.

Raises

TpGeneralException,TpGCCSException

Method

getMoreDialledDigitsErr()

This asynchronous method reports an error in collecting digits to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGeneralException,TpGCCSException

Method

callEnded()

This method indicates to the application that the call has terminated in the network. However, the application may still receive some results (e.g., getCallInfoRes) related to the call. The application is expected to deassign the call object after having received the callEnded.

Note that the event that caused the call to end might also be received separately if the application was monitoring for it.

Parameters

callSessionID : in TpSessionID

Specifies the call sessionID.

report : in TpCallEndedReport

Specifies the reason the call is terminated.

Raises

TpGeneralException,TpGCCSException

6.4 Generic Call Control Service State Transition Diagrams

6.4.1 State Transition Diagrams for IpCallControlManager

The state transition diagram shows the application view on the Call Control Manager object.

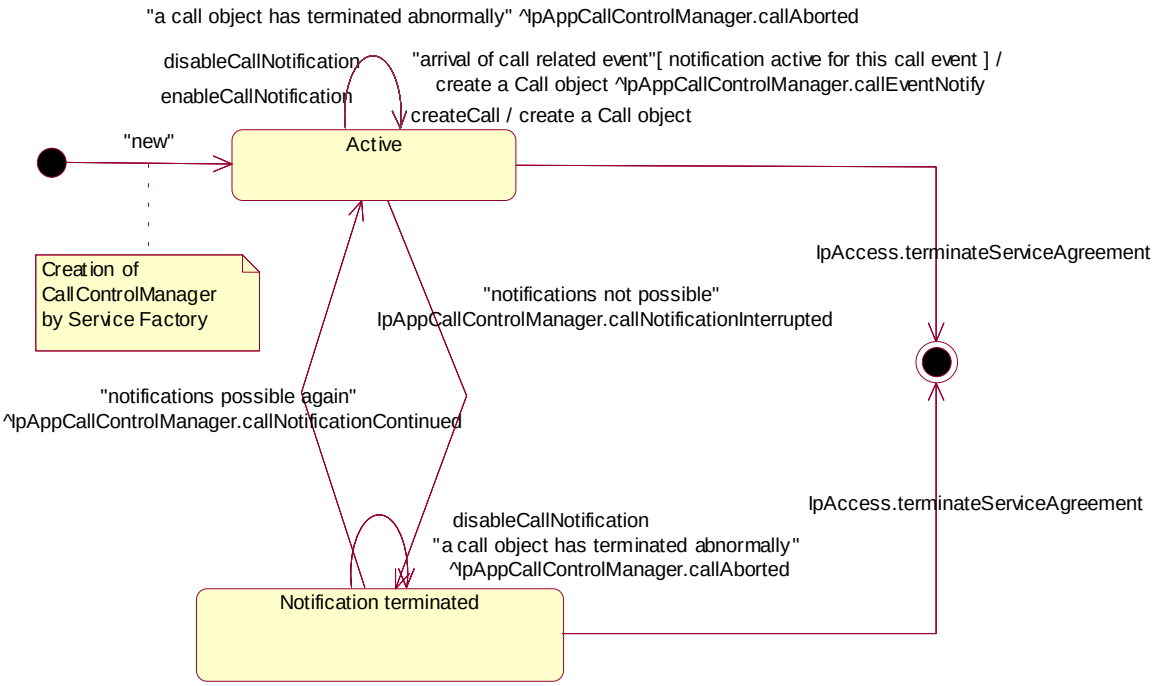


Figure : Application view on the Call Control Manager

6.4.1.1 Active State

In this state a relation between the Application and the Generic Call Control Service has been established. The state allows the applicatoin to indicate that it is interested in call related events. In case such an event occurs, the Call Control Manager will create a Call object and inform the application by invoking the operation callEventNotify() on the IpAppCallControlManager interface. The application can also indicate it is no longer interested in certain call related events by calling disableCallNotification().

6.4.1.2 Notification terminated State

When the Call Control Manager is in the Notification terminated state, events requested with enableCallNotification() will not be forwarded to the application. There can be multiple reasons for this: for instance it might be that the application receives more notifications from the network than defined in the Service Level Agreement. Another example is that the Service has detected it receives no notifications from the network due to e.g. a link failure. In this state no requests for new notifications will be accepted.

6.4.2 State Transition Diagrams for IpCall

The state transition diagram shows the application view on the Call object. This diagram shows only the part of the state transition diagram valid for 3GPP (UMTS) release 4.

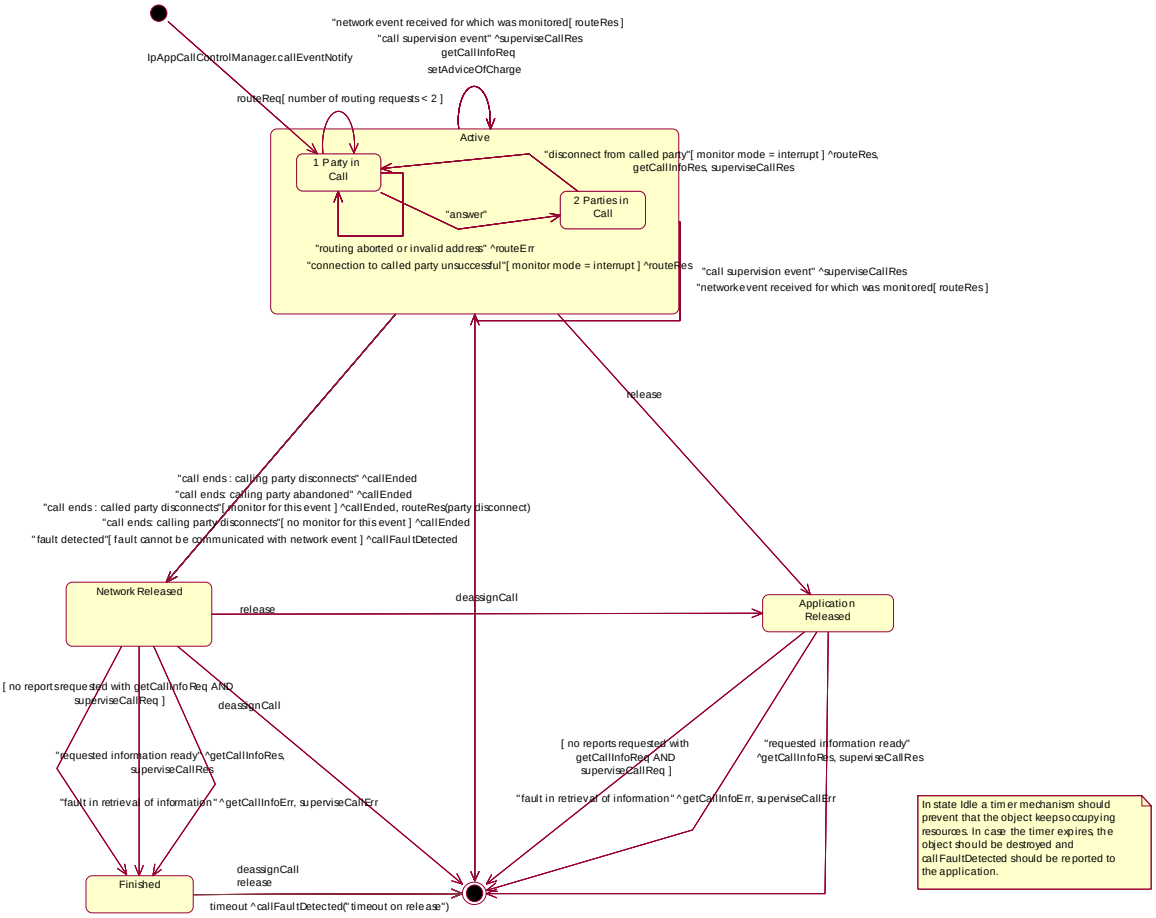


Figure : Application view on the IpCall object

6.4.2.1 Network Released State

In this state the call has ended and the Gateway collects the possible call information requested with getCallInfoReq() and / or superviseCallReq(). The information will be returned to the application by invoking the methods getCallInfoRes() and / or superviseCallRes() on the application. Also when a call was unsuccessful these methods are used.In case the application has not requested additional call related information immediately a transition is made to state Idle.

6.4.2.2 Finished State

In this state the call has ended and no call related information is to be send to the application. The application can only release the call object. Calling the deassignCall() operation has the same effect. Note that the application has to release the object itself as good OO practice requires that when an object was created on behalf of a certain entity, this entity is also responsible for destroying it when the object is no longer needed.

6.4.2.3 Application Released State

In this state the application has requested to release the Call object and the Gateway collects the possilbe call information requested with getCallInfoReq() and / or superviseCallReq(). In case the application has not requested additional call related information the Call object is destroyed immediately.

6.4.2.4 Active State

In this state a call between two parties is being setup or present. Refer to the substates for more details. The application can request the gateway for a certain type of charging of the call by calling setCallChargePlan(). The application can request for charging related information by calling getCallInfoReq(). Furthermore the application can request supervision of the call by calling superviseCallReq(). It is also allowed to send Advice of Charge information by calling setAdviceOfCharge().

6.4.2.5 1 Party in Call State

When the Call is in this state a calling party is present. The application can now request that a connection to a called party be established by calling the method routeReq(). When the calling party abandons the call before the application has invoked the routeReq() operation, the gateway informs the application by invoking callFaultDetected() and also the operation callEnded() will be invoked. When the calling party abandons the call after the application has invoked routeReq() but before the call has actually been established, the gateway informs the application by invoking callEnded().

When the calling party answers the call, a transition will be made to the 2 Parties in Call state. In case the call can not be established because the application supplied an invalid address or the connection to the called party was unsuccessful while the application was monitoring for the latter in interrupt mode, the Call object will stay in this state

In this state user interaction is possible unless there is an outstanding routing request.

6.4.2.6 2 Parties in Call State

A connection between two parties has been established.

In case the calling party disconnects, the gateway informs the application by invoking callEnded().

When the called party disconnects different situations apply:

1. the application is monitoring for this event in interrupt mode: a transition is made to the 1 Party in Call state, the application is informed with routeRes with indication that the called party has disconnected and all requested reports are sent to the application. The application now again has control of the call.

2. the application is monitoring for this event but not in interrupt mode. In this case a transition is made to the Network Released state and the gateway informs the application by invoking the operation routeRes() and callEnded().

3. the application is not monitoring for this event. In this case the application is informed by the gateway invoking the callEnded() operation and a transition is made to the Network Released state.

6.5 Generic Call Control Service Properties

The following table lists properties relevant for the Generic Call Control API.

Property	Type	Description / Interpretation
P_TRIGGERING_EVENT_TYPES	INTEGER_SET	Indicates the static event types supported by the SCS. Static events are the events by which applications are initiated.
P_DYNAMIC_EVENT_TYPES	INTEGER_SET	Indicates the dynamic event types supported by the SCS. Dynamic events are the events the application can request for during the context of a call.
P_ADDRESSPLAN	INTEGER_SET	Indicates the supported address plan (defined in TpAddressPlan.) e.g. {P_ADDRESS_PLAN_E164,

		P_ADDRESS_PLAN_IP})
P_UI_CALL_BASED	BOOLEAN_SET	Value = TRUE : User interaction can be performed on call level and a reference to a Call object can be used in the IpUIManager.createUICall() operation. Value = FALSE: No User interaction on call level is supported.
P_UI_AT_ALL_STAGES	BOOLEAN_SET	Value = TRUE: User Interaction can be performed at any stage during a call . Value = FALSE: User Interaction can be performed in case there is only one party in the call.
P_MEDIA_TYPE	INTEGER_SET	Specifies the media type used by the Service. Values are defined by data-type TpMediaType : P_AUDIO, P_VIDEO, P_DATA

The previous table lists properties related to capabilities of the SCS itself. The following table lists properties that are used in the context of the Service Level Agreement, e.g. to restrict the access of applications to the capabilities of the SCS.

Property	**Type**	**Description**
P_TRIGGERING_ADDRESSES	ADDRESS_RANGE_SET	Indicates for which numbers the notification may be set. For terminating notifications it applies to the terminating number, for originating notifications it applies only to the originating number.
P_NOTIFICATION_TYPES	INTEGER_SET	Indicates whether the application is allowed to set oritginating and/or terminating triggers in the ECN. Set is: P_ORIGINATING P_TERMINATING
P_MONITOR_MODE	INTEGER_SET	Indicates whether the application is allowed to monitor in interrupt and/or notify mode. Set is: P_INTERRUPT P_NOTIFY
P_NUMBERS_TO_BE_CHANGED	INTEGER_SET	Indicates which numbers the application is allowed to change or fill for legs in an incoming call. Allowed value set: {P_ORIGINAL_CALLED_PARTY_NUMBER, P_REDIRECTING_NUMBER, P_TARGET_NUMBER, P_CALLING_PARTY_NUMBER}.
P_CHARGEPLAN_ALLOWED	INTEGER_SET	Indicates which charging is allowed in the

		setCallChargePlan indicator. Allowed values: **{P_CHARGE_PER_TIME,** P_TRANSPARANT_CHARGING, P_CHARGE_PLAN}
P_CHARGEPLAN_MAPPING	INTEGER_INTEGER_MAP	Indicates the mapping of chargeplans (we assume they can be indicated with integers) to a logical network chargeplan indicator. When the chargeplan supports indicates P_CHARGE_PLAN then only chargeplans in this mapping are allowed.

6.6 Generic Call Control Data Definitions

This document provides the generic call control data definitions necessary to support the API specification.

This document is written using Hypertext link, to aid navigation through the data structures. Underlined text represents Hypertext links.

The general format of a data definition specification is described below.

- Data Type

This shows the name of the data type.

- Description

This describes the data type.

- Tabular Specification

This specifies the data types and values of the data type.

- Example

If relevant, an example is shown to illustrate the data type.

6.6.1 Generic Call Control Event Notification Data Definitions

TpCallEventName

Defines the names of event being notified. The following events are supported. The values may be combined by a logical 'OR' function when requesting the notifications. Additional events that can be requested / received during the call process are found in the TpCallReportType data-type.

Name	Value	Description
P_EVENT_NAME_UNDEFINED	0	Undefined
P_EVENT_GCCS_OFFHOOK_EVENT	1	GCCS – Offhook event This can be used for hot-line features. In case this event is set in the TpCallEventCriteria, only the originating address(es) may be specified in the criteria.
P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT	2	GCCS – Address information collected The network has collected the information from the A-party, but not yet analysed the information. The number can still be incomplete. Applications might set notifications for this event when part of the number analysis needs to be done in the application (see also the getMoreDialledDigits

		method on the call class).
P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT	4	GCCS – Address information is analysed The dialled number is a valid and complete number in the network.
P_EVENT_GCCS_CALLED_PARTY_BUSY	8	GCCS – Called party is busy
P_EVENT_GCCS_CALLED_PARTY_UNREACHABLE	16	GCCS – Called party is unreachable (e.g., the called party has a mobile telephone that is currently switched off).
P_EVENT_GCCS_NO_ANSWER_FROM_CALLED_PARTY	32	GCCS – No answer from called party
P_EVENT_GCCS_ROUTE_SELECT_FAILURE	64	GCCS – Failure in routing the call
P_EVENT_GCCS_ANSWER_FROM_CALL_PARTY	128	GCCS – Party answered call.

TpCallNotificationType

Defines the type of notification. Indicates whether it is related to the originating of the terminating user in the call.

Name	Value	Description
P_ORIGINATING	1	Indicates that the notification is related to the originating user in the call.
P_TERMINATING	2	Indicates that the notification is related to the terminating user in the call.

TpCallMonitorMode

Defines the mode that the call will monitor for events, or the mode that the call is in following a detected event.

Name	Value	Description
P_CALL_MONITOR_MODE_INTERRUPT	0	The call event is intercepted by the call control service and call processing is interrupted. The application is notified of the event and call processing resumes following an appropriate API call or network event (such as a call release)
P_CALL_MONITOR_MODE_NOTIFY	1	The call event is detected by the call control service but not intercepted. The application is notified of the event and call processing continues
P_CALL_MONITOR_MODE_DO_NOT_MONITOR	2	Do not monitor for the event

TpCallEventCriteria

Defines the Sequence of Data Elements that specify the criteria for a event notification.

Of the addresses only the Plan and the AddrString are used for the purpose of matching the notifications against the criteria.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddressRange	Defines the destination address or address range for which the notification is requested.
OriginatingAddress	TpAddressRange	Defines the origination address or a address range for which the notification is requested.
CallEventName	TpCallEventName	Name of the event(s)
CallNotificationType	TpCallNotificationType	Indicates whether it is related to the originating or the terminating user in the call.

MonitorMode	TpCallMonitorMode	Defines the mode that the call is in following the notification. Monitor mode P_CALL_MONITOR_MODE_DO_NOT_MONITOR is not a legal value here.

TpCallEventInfo

Defines the Sequence of Data Elements that specify the information returned to the application in a Call event notification.

Sequence Element Name	Sequence Element Type
DestinationAddress	TpAddress
OriginatingAddress	TpAddress
OriginalDestinationAddress	TpAddress
RedirectingAddress	TpAddress
CallAppInfo	TpCallAppInfoSet
CallEventName	TpCallEventName
CallNotificationType	TpCallNotificationType
MonitorMode	TpCallMonitorMode

6.6.2 Generic Call Control Data Definitions

IpCall

Defines the address of an IpCall Interface.

IpCallRef

Defines a Reference to type IpCall.

IpAppCall

Defines the address of an IpAppCall Interface.

IpAppCallRef

Defines a Reference to type IpAppCall

IpAppCallRefRef

Defines a Reference to type IpAppCallRef.

TpCallIdentifierRef

Defines a Reference to type TpCallIdentifier.

TpCallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Generic Call object

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallReference	IpCallRef	This element specifies the interface reference for the call object.
CallSessionID	TpSessionID	This element specifies the call session ID of the call.

IpAppCallControlManager

Defines the address of an IpAppCallControlManager Interface.

IpAppCallControlManagerRef

Defines a Reference to type IpAppCallControlManager.

IpCallControlManager

Defines the address of an IpCallControlManager Interface.

IpCallControlManagerRef

Defines a Reference to type IpCallControlManager.

TpCallAlertingMechanism

This data type is identical to a TpInt32, and defines the mechanism that will be used to alert a call party. The values of this data type are operator specific.

TpCallAppInfo

Defines the Tagged Choice of Data Elements that specify application-related call information.

	Tag Element Type	
	TpCallAppInfoType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_APP_ALERTING_MECHANISM	TPCallAlertingMechanism	CallAppAlertingMechanism
P_CALL_APP_NETWORK_ACCESS_TYPE	TpCallNetworkAccessType	CallAppNetworkAccessType
P_CALL_APP_TELE_SERVICE	TpCallTeleService	CallAppTeleService
P_CALL_APP_BEARER_SERVICE	TpCallBearerService	CallAppBearerService
P_CALL_APP_PARTY_CATEGORY	TpCallPartyCategory	CallAppPartyCategory
P_CALL_APP_PRESENTATION_ADDRESS	TpAddress	CallAppPresentationAddress
P_CALL_APP_GENERIC_INFO	TpString	CallAppGenericInfo
P_CALL_APP_ADDITIONAL_ADDRESS	TpAddress	CallAppAdditionalAddress

TpCallAppInfoType

Defines the type of call application-related specific information.

Name	Value	Description
P_CALL_APP_UNDEFINED	0	Undefined
P_CALL_APP_ALERTING_MECHANISM	1	The alerting mechanism or pattern to use
P_CALL_APP_NETWORK_ACCESS_TYPE	2	The network access type (e.g. ISDN)
P_CALL_APP_TELE_SERVICE	3	Indicates the tele-service (e.g. telephony)
P_CALL_APP_BEARER_SERVICE	4	Indicates the bearer service (e.g. 64kb/s unrestricted data).
P_CALL_APP_PARTY_CATEGORY	5	The category of the calling party
P_CALL_APP_PRESENTATION_ADDRESS	6	The address to be presented to other call parties
P_CALL_APP_GENERIC_INFO	7	Carries unspecified service-service information
P_CALL_APP_ADDITIONAL_ADDRESS	8	Indicates an additional address

TpCallAppInfoSet

Defines a Numbered Set of Data Elements of TpCallAppInfo.

TpCallBearerService

This data type defines the type of call application-related specific information (Q.931: Information Transfer Capability, and 3G TS 22.002)

Name	Value	Description
P_CALL_BEARER_SERVICE_UNKNOWN	0	Bearer capability information unknown at this time
P_CALL_BEARER_SERVICE_SPEECH	1	Speech
P_CALL_BEARER_SERVICE_DIGITALUNRESTRICTED	2	Unrestricted digital information
P_CALL_BEARER_SERVICE_DIGITALRESTRICTED	3	Restricted digital information
P_CALL_BEARER_SERVICE_AUDIO	4	3.1 kHz audio
P_CALL_BEARER_SERVICE_ DIGITALUNRESTRICTEDTONES	5	Unrestricted digital information with tomes/announcements
P_CALL_BEARER_SERVICE_VIDEO	6	Video

TpCallChargePlan

Defines the Sequence of Data Elements that specify the charge plan for the call.

Sequence Element Name	Sequence Element Type	Description
ChargeOrderType	TpCallChargeOrder	Charge order
Currency	TpString	Currency unit according to ISO-4217:1995
AdditionalInfo	TpString	Descriptive string which is sent to the billing system without prior evaluation. Could be included in the ticket.

Valid Currencies are:

ADP, AED, AFA, ALL, AMD, ANG, AON, AOR, ARS, ATS, AUD, AWG, AZM, BAM, BBD, BDT, BEF, BGL, BGN, BHD, BIF, BMD, BND, BOB, BOV, BRL, BSD, BTN, BWP, BYB, BZD, CAD, CDF, CHF, CLF, CLP, CNY, COP, CRC, CUP, CVE, CYP, CZK, DEM, DJF, DKK, DOP, DZD, ECS, ECV, EEK, EGP, ERN, ESP, ETB, EUR, FIM, FJD, FKP, FRF, GBP, GEL, GHC, GIP, GMD, GNF, GRD, GTQ, GWP, GYD, HKD, HNL, HRK, HTG, HUF, IDR, IEP, ILS, INR, IQD, IRR, ISK, ITL, JMD, JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KYD, KZT, LAK, LBP, LKR, LRD, LSL, LTL, LUF, LVL, LYD, MAD, MDL, MGF, MKD, MMK, MNT, MOP, MRO, MTL, MUR, MVR, MWK, MXN, MXV, MYR, MZM, NAD, NGN, NIO, NLG, NOK, NPR, NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PTE, PYG, QAR, ROL, RUB, RUR, RWF, SAR, SBD, SCR, SDD, SEK, SGD, SHP, SIT, SKK, SLL, SOS, SRG, STD, SVC, SYP, SZL, THB, TJR, TMM, TND, TOP, TPE, TRL, TTD, TWD, TZS, UAH, UGX, USD, USN, USS, UYU, UZS, VEB, VND, VUV, WST, XAF, XAG, XAU, XBA, XBB, XBC, XBD, XCD, XDR, XFO, XFU, XOF, XPD, XPF, XPT, XTS, XXX, YER, YUM, ZAL, ZAR, ZMK, ZRN, ZWD.

XXX is used for transactions where no currency is involved.

TpCallChargeOrder

Defines the Tagged Choice of Data Elements that specify the charge plan for the call.

	Tag Element Type	
	TpCallChargeOrderCategory	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_CHARGE_PER_TIME	TpChargePerTime	ChargePerTime
P_CALL_CHARGE_NETWORK	TpString	NetworkCharge

TpCallChargeOrderCategory

Defines the type of charging to be applied

Name	Value	Description
P_CALL_CHARGE_PER_TIME	0	Charge per time
P_CALL_CHARGE_NETWORK	1	Operator specific charge plan specification, e.g. charging table name / charging table entry

TpCallError

Defines the Sequence of Data Elements that specify the additional information relating to acall error.

Sequence Element Name	Sequence Element Type
ErrorTime	TpDateAndTime
ErrorType	TpCallErrorType
AdditionalErrorInfo	TpCallAdditionalErrorInfo

TpCallAdditionalErrorInfo

Defines the Tagged Choice of Data Elements that specify additional call error and call error specific information. This is also used to specify call leg errors and information errors.

	Tag Element Type	
	TpCallErrorType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_ERROR_UNDEFINED	NULL	Undefined
P_CALL_ERROR_INVALID_ADDRESS	TpAddressError	CallErrorInvalidAddress
P_CALL_ERROR_INVALID_STATE	NULL	Undefined

TpCallErrorType

Defines a specific call error.

Name	Value	Description
P_CALL_ERROR_UNDEFINED	0	Undefined; the method failed or was refused, but no specific reason can be given.
P_CALL_ERROR_INVALID_ADDRESS	1	The operation failed because an invalid address was given
P_CALL_ERROR_INVALID_STATE	2	The call was not in a valid state for the requested operation

TpCallFault

Defines the cause of the call fault detected.

Name	Value	Description
P_CALL_FAULT_UNDEFINED	0	Undefined
P_CALL_TIMEOUT_ON_RELEASE	1	This fault occurs when the final report has been sent to the application, but the application did not explicitly release or deassign the call object, within a specified time. The timer value is operator specific.
P_CALL_TIMEOUT_ON_INTERRUPT	2	This fault occurs when the application did not instruct the gateway how to handle the call within a specified time, after the gateway reported an event that was requested by the application in interrupt mode. The timer value is operator specific.

TpCallEndedReport

Defines the Sequence of Data Elements that specify the reason for the call ending.

Sequence Element Name	Sequence Element Type	
CallLegSessionID	TpSessionID	The leg that initiated the release of the call. If the call release was not initiated by the leg, then this value is set to –1.
Cause	TpCallReleaseCause	The cause of the call ending.

TpCallInfoReport

Defines the Sequence of Data Elements that specify the call information requested. Information that was not requested is invalid.

Sequence Element Name	Sequence Element Type	Description
CallInfoType	TpCallInfoType	The type of call report.
CallInitiationStartTime	TpDateAndTime	The time and date when the call, or follow-on call, was started.
CallConnectedToResourceTime	TpDateAndTime	The date and time when the call was connected to the resource. This data element is only valid when information on user interaction is reported.
CallConnectedToDestinationTime	TpDateAndTime	The date and time when the call was connected to the destination (i.e., when the destination answered the call). If the destination did not answer, the time is set to an empty string. This data element is invalid when information on user interaction is reported with

		an intermediate report.
CallEndTime	TpDateAndTime	The date and time when the call or follow-on call or user interaction was terminated.
Cause	TpCallReleaseCause	The cause of the termination.

A callInfoReport will be generated at the end of user interaction and at the end of the connection with the associated address. This means that either the destination related information is present or the resource related information, but not both.

TpCallInfoType

Defines the type of call information requested and reported. The values may be combined by a logical 'OR' function.

Name	Value	Description
P_CALL_INFO_UNDEFINED	00h	Undefined
P_CALL_INFO_TIMES	01h	Relevant call times
P_CALL_INFO_RELEASE_CAUSE	02h	Call release cause
P_CALL_INFO_INTERMEDIATE	04h	Send only intermediate reports. When this is not specified the information report will only be sent when the call has ended. When intermediate reports are requested a report will be generated between follow-on calls, i.e., when a party leaves the call.

TpCallNetworkAccessType

This data defines the bearer capabilities associated with the call. (3G TS 24.002) This information is network operator specific and may not always be available because there is no standard protocol to retrieve the information.

Name	Value	Description
P_CALL_NETWORK_ACCESS_TYPE_UNKNOWN	0	Network type information unknown at this time
P_CALL_NETWORK_ACCESS_TYPE_POT	1	POTS
P_CALL_NETWORK_ACCESS_TYPE_ISDN	2	ISDN
P_CALL_NETWORK_ACCESS_TYPE_DIALUPINTERNET	3	Dial-up Internet
P_CALL_NETWORK_ACCESS_TYPE_XDSL	4	xDLS
P_CALL_NETWORK_ACCESS_TYPE_WIRELESS	5	Wireless

TpCallPartyCategory

This data type defines the category of a calling party. (Q.763: Calling Party Category / Called Party Category)

Name	Value	Description
P_CALL_PARTY_CATEGORY_UNKNOWN	0	calling party's category unknown at this time
P_CALL_PARTY_CATEGORY_OPERATOR_F	1	operator, language French
P_CALL_PARTY_CATEGORY_OPERATOR_E	2	operator, language English

P_CALL_PARTY_CATEGORY_OPERATOR_G	3	operator, language German
P_CALL_PARTY_CATEGORY_OPERATOR_R	4	operator, language Russian
P_CALL_PARTY_CATEGORY_OPERATOR_S	5	operator, language Spanish
P_CALL_PARTY_CATEGORY_ORDINARY_SUB	6	ordinary calling subscriber
P_CALL_PARTY_CATEGORY_PRIORITY_SUB	7	calling subscriber with priority
P_CALL_PARTY_CATEGORY_DATA_CALL	8	data call (voice band data)
P_CALL_PARTY_CATEGORY_TEST_CALL	9	test call
P_CALL_PARTY_CATEGORY_PAYPHONE	10	payphone

TpCallReleaseCause

Defines the Sequence of Data Elements that specify the cause of the release of a call.

Sequence Element Name	Sequence Element Type
Value	TpInt32
Location	TpInt32

Note: the Value and Location are specified as in ITU-T recommendation Q.850.

TpCallServiceCode

Defines the Sequence of Data Elements that specify the service code and type of service code received during a call. The service code type defines how the value string should be interpreted.

Sequence Element Name	Sequence Element Type
CallServiceCodeType	TpCallServiceCodeType
ServiceCodeValue	TpString

TpCallServiceCodeType

Defines the different types of service codes that can be received during the call.

Name	Value	Description
P_CALL_SERVICE_CODE_UNDEFINED	0	The type of service code is unknown. The corresponding string is operator specific.
P_CALL_SERVICE_CODE_DIGITS	1	The user entered a digit sequence during the call. The corresponding string is an ascii representation of the received digits.
P_CALL_SERVICE_CODE_FACILITY	2	A facility information element is received. The corresponding string contains the facility information element as defined in ITU Q.932
P_CALL_SERVICE_CODE_U2U	3	A user-to-user message was received. The associated string contains the content of the user-to-user information element.
P_CALL_SERVICE_CODE_HOOKFLASH	4	The user performed a hookflash, optionally followed by some digits. The corresponding string is an ascii representation of the entered digits.
P_CALL_SERVICE_CODE_RECALL	5	The user pressed the register recall button, optionally followed by some digits. The corresponding string is an ascii representation of the entered digits.

TpCallTeleService

This data type defines the tele-service associated with the call. (Q.763: User Teleservice Information, Q.931: High Layer Compatitibility Information, and 3G TS 22.003)

Name	Value	Description
P_CALL_TELE_SERVICE_UNKNOWN	0	Teleservice information unknown at this time
P_CALL_TELE_SERVICE_TELEPHONY	1	Telephony
P_CALL_TELE_SERVICE_FAX_2_3	2	Facsimile Group 2/3
P_CALL_TELE_SERVICE_FAX_4_I	3	Facsimile Group 4, Class I
P_CALL_TELE_SERVICE_FAX_4_II_III	4	Facsimile Group 4, Classes II and III
P_CALL_TELE_SERVICE_VIDEOTEX_SYN	5	Syntax based Videotex
P_CALL_TELE_SERVICE_VIDEOTEX_INT	6	International Videotex interworking via gateways or interworking units
P_CALL_TELE_SERVICE_TELEX	7	Telex service
P_CALL_TELE_SERVICE_MHS	8	Message Handling Systems
P_CALL_TELE_SERVICE_OSI	9	OSI application
P_CALL_TELE_SERVICE_FTAM	10	FTAM application
P_CALL_TELE_SERVICE_VIDEO	11	Videotelephony
P_CALL_TELE_SERVICE_VIDEO_CONF	12	Videoconferencing
P_CALL_TELE_SERVICE_AUDIOGRAPH_CONF	13	Audiographic conferencing
P_CALL_TELE_SERVICE_MULTIMEDIA	14	Multimedia services
P_CALL_TELE_SERVICE_CS_INI_H221	15	Capability set of initial channel of H.221
P_CALL_TELE_SERVICE_CS_SUB_H221	16	Capability set of subsequent channel of H.221
P_CALL_TELE_SERVICE_CS_INI_CALL	17	Capability set of initial channel associated with an active 3.1 kHz audio or speech call.
P_CALL_TELE_SERVICE_DATATRAFFIC	18	Data traffic.
P_CALL_TELE_SERVICE_EMERGENCY_CALLS	1	Emergency Calls
P_CALL_TELE_SERVICE_SMS_MT_PP	2	Short message MT/PP
P_CALL_TELE_SERVICE_SMS_MO_PP	2	Short message MO/PP
P_CALL_TELE_SERVICE_CELL_BROADCAST	2	Cell Broadcast Service
P_CALL_TELE_SERVICE_ALT_SPEECH_FAX_3	2	Alternate speech and facsimile group 3
P_CALL_TELE_SERVICE_AUTOMATIC_FAX_3	2	Automatic Facsimile group 3
P_CALL_TELE_SERVICE_VOICE_GROUP_CALL	2	Voice Group Call Service
P_CALL_TELE_SERVICE_VOICE_BROADCAST	2	Voice Broadcast Service

TpCallSuperviseReport

Defines the responses from the call control service for calls that are supervised. The values may be combined by a logical 'OR' function.

Name	Value	Description

P_CALL_SUPERVISE_TIMEOUT	01h	The call supervision timer has expired
P_CALL_SUPERVISE_CALL_ENDED	02h	The call has ended, either due to timer expiry or call party release. In case the called party disconnects but a follow-on call can still be made also this indication is used.
P_CALL_SUPERVISE_TONE_APPLIED	04h	A warning tone has been applied. This is only sent in combination with P_CALL_SUPERVISE_TIMEOUT
P_CALL_SUPERVISE_UI_FINISHED	0	The user interaction has finished.

TpCallSuperviseTreatment

Defines the treatment of the call by the call control service when the call supervision timer expires. The values may be combined by a logical 'OR' function.

Name	Value	Description
P_CALL_SUPERVISE_RELEASE	01h	Release the call when the call supervision timer expires
P_CALL_SUPERVISE_RESPOND	02h	Notify the application when the call supervision timer expires
P_CALL_SUPERVISE_APPLY_TONE	04h	Send a warning tone to the originating party when the call supervision timer expires. If call release is requested, then the call will be released following the tone after an administered time period

TpCallReport

Defines the Sequence of Data Elements that specify the call report and call leg report specific information.

Sequence Element Name	Sequence Element Type
MonitorMode	TpCallMonitorMode
CallEventTime	TpDateAndTime
CallReportType	TpCallReportType
AdditionalReportInfo	TpCallAdditionalReportInfo

TpCallAdditionalReportInfo

Defines the Tagged Choice of Data Elements that specify additional call report information for certain types of reports..

	Tag Element Type	
	TpCallReportType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_REPORT_UNDEFINED	NULL	Undefined
P_CALL_REPORT_PROGRESS	NULL	Undefined
P_CALL_REPORT_ALERTING	NULL	Undefined
P_CALL_REPORT_ANSWER	NULL	Undefined
P_CALL_REPORT_BUSY	TpCallReleaseCause	Busy
P_CALL_REPORT_NO_ANSWER	NULL	Undefined
P_CALL_REPORT_DISCONNECT	TpCallReleaseCause	CallDisconnect
P_CALL_REPORT_REDIRECTED	TpAddress	ForwardAddress
P_CALL_REPORT_SERVICE_CODE	TpCallServiceCode	ServiceCode
P_CALL_REPORT_ROUTING_FAILURE	TpCallReleaseCause	RoutingFailure

TpCallReportRequest

Defines the Sequence of Data Elements that specify the criteria relating to call report requests.

Sequence Element Name	Sequence Element Type
MonitorMode	TpCallMonitorMode
CallReportType	TpCallReportType
AdditionalReportCriteria	TpCallAdditionalReportCriteria

TpCallAdditionalReportCriteria

Defines the Tagged Choice of Data Elements that specify specific criteria.

	Tag Element Type	
	TpCallReportType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_REPORT_UNDEFINED	NULL	Undefined
P_CALL_REPORT_PROGRESS	NULL	Undefined
P_CALL_REPORT_ALERTING	NULL	Undefined

P_CALL_REPORT_ANSWER	NULL	Undefined
P_CALL_REPORT_BUSY	NULL	Undefined
P_CALL_REPORT_NO_ANSWER	TpDuration	NoAnswerDuration
P_CALL_REPORT_DISCONNECT	NULL	Undefined
P_CALL_REPORT_REDIRECTED	NULL	Undefined
P_CALL_REPORT_SERVICE_CODE	TpCallServiceCode	ServiceCode
P_CALL_REPORT_ROUTING_FAILURE	NULL	Undefined

TpCallReportRequestSet

Defines a Numbered Set of Data Elements of TpCallReportRequest.

TpCallReportType

Defines a specific call event report type.

Name	Value	Description
P_CALL_REPORT_UNDEFINED	0	Undefined
P_CALL_REPORT_PROGRESS	1	Call routing progress event:an indication from the network that progress has been made in routing the call to the requested call party.
P_CALL_REPORT_ALERTING	2	Call is alerting at the call party
P_CALL_REPORT_ANSWER	3	Call answered at address
P_CALL_REPORT_BUSY	4	Called address refused call due to busy
P_CALL_REPORT_NO_ANSWER	5	No answer at called address
P_CALL_REPORT_DISCONNECT	6	The call party has disconnected.
P_CALL_REPORT_REDIRECTED	7	Call redirected to new address: an indication from the network that the call has been redirected to a new address.
P_CALL_REPORT_SERVICE_CODE	8	Mid-call service code received
P_CALL_REPORT_ROUTING_FAILURE	9	Call routing failed - re-routing is possible

TpCallLoadControlMechanism

Defines the Tagged Choice of Data Elements that specify the applied mechanism and associated parameters.

	Tag Element Type	
	TpCallLoadControlMechanismType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_LOAD_CONTROL_PER_INTERVAL	TpCallLoadControlIntervalRate	CallLoadControlPerInterval

TpCallLoadControlIntervalRate

Defines the call admission rate of the call load control mechanism used. This data type indicates the interval (in milliseconds) between calls that are admitted.

Name	Value	Description
P_CALL_LOAD_CONTROL_ADMIT_NO_CALLS	0	Infinite interval (do not admit any calls)
	1 - 60000	Duration in milliseconds

TpCallLoadControlMechanismType

Defines the type of call load control mechanism to use.

Name	Value	Description
P_CALL_LOAD_CONTROL_PER_INTERVAL	1	admit one call per interval

TpCallTreatment

Defines the Sequence of Data Elements that specify the the treatment for calls that will be handled only by the network (for example, call which are not admitted by the call load control mechanism).

Sequence Element Name	Sequence Element Type
ReleaseCause	TpCallReleaseCause
AdditionalTreatmentInfo	TpCallAdditionalTreatmentInfo

TpCallAdditionalTreatmentInfo

Defines the Tagged Choice of Data Elements that specify the information to be sent to a call party.

	Tag Element Type	
	TpCallTreatmentType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_TREATMENT_DEFAULT	NULL	Undefined
P_CALL_TREATMENT_RELEASE	NULL	Undefined
P_CALL_TREATMENT_SIAR	TpUICallInfoID	InformationToSend

TpCallTreatmentType

Defines the treatment for calls that will be handled only by the network.

Name	Value	Description
P_CALL_TREATMENT_DEFAULT	0	Default treatment
P_CALL_TREATMENT_RELEASE	1	Release the call
P_CALL_TREATMENT_SIAR	2	Send information to the user, and release the call (Send Info & Release)

TpCallEventCriteriaResultSetRef

Defines a refernce to TpCallEventCriteriaResultSet.

TpCallEventCriteriaResultSet

Defines a set of TpCallEventCriteriaResult.

TpCallEventCriteriaResult

Defines a sequence of data elements that specify a requested call event notification criteria with the associated assignmentID.

Sequence Element Name	Sequence Element Type	Sequence Element Description
EventCriteria	TpCallEventCriteria	The event criteria that were specified by the application.
AssignmentID	TpInt32	The associated assignmentID. This can be used to disable the notification.

7 MultiParty Call Control Service

7.1 Sequence Diagrams

7.1.1 Application initiated call setup

The following sequence diagram shows an application creating a call between party A and party B. Here, a call is created first. Then party A's call leg is created before triggers are set on it for answer and then routed to the call. On answer, an announcement is played indicating that the call is being set up to party B. While the announcement is being played, party B's call leg is created and then triggers are set on it for answer. On answer the announcement is cancelled and party B is routed to the call.

1: This message is used to create an object implementing the IpAppMultiPartyCall interface.

2: This message requests the object implementing the IpMultiPartyCallControlManager interface to create an object implementing the IpMultiPartyCall interface.

3: Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface (e.g. load control values not exceeded) is met it is created.

4: Once the object implementing the IpMultiPartyCall interface is created it is used to pass the reference of the object implementing the IpAppMultiPartyCall interface as the callback reference to the object implementing the IpMultiPartyCall interface. Note that the reference to the callback interface could already have been passed in the createCall.

5: This message instructs the object implementing the IpMultiPartyCall interface to create a call leg for customer A.

6: Assuming that the criteria for creating an object implementing the IpCallLeg interface is met, message 6 is used to create it.

7: This message requests the call leg for customer A to inform the application when the call leg answers the call.

8: The call is then routed to the originating call leg.

9: Assuming the call is answered, the object implementing party A's IpCallLeg interface passes the result of the call being answered back to its callback object. This message is then forwarded via another message (not shown) to the object implementing the IpAppLogic interface.

10: A UICall object is created and associated with the just created call leg.

11: This message is used to inform party A that the call is being routed to party B.

12: An indication that the dialogue with party A has commenced is returned via message 13 and eventually forwarded via another message (not shown) to the object implementing the IpAppLogic interface.

13: This message instructs the object implementing the IpMultiPartyCall interface to create a call leg for customer B.

14: Assuming that the criteria for creating a second object implementing the IpCallLeg interface is met, it is created.

15: This message requests the call leg for customer B to inform the application when the call leg answers the call.

16: The call is then routed to the call leg.

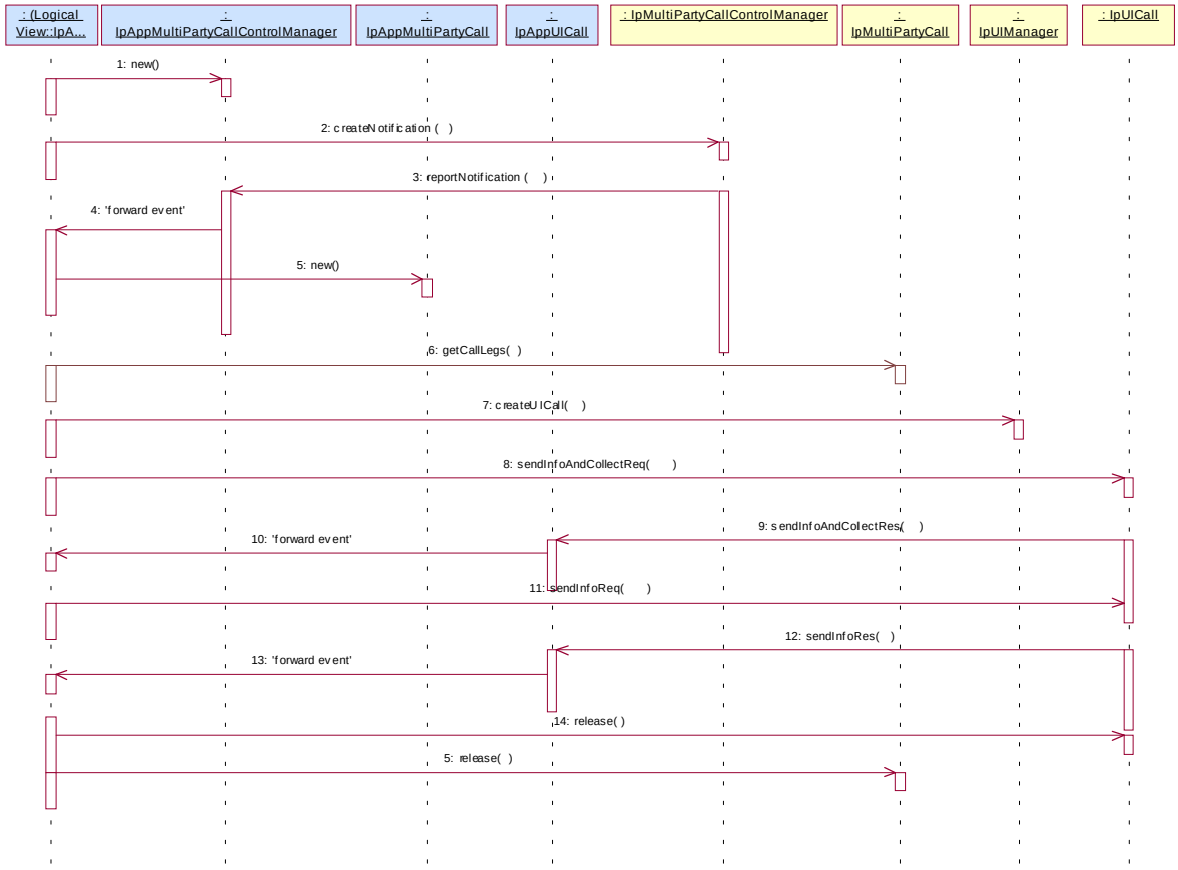
17: Assuming the call is answered, the object implementing party B's IpCallLeg interface passes the result of the call being answered back to its callback object. This message is then forwarded via another message (not shown) to the object implementing the IpAppLogic interface.

18: This message then instructs the object implementing the IpUICall interface to stop sending announcements to party A.

19: The application deassigns the call. This will also deassign the associated user interaction.

7.1.2 Call Barring 2

The following sequence diagram shows a call barring service, initiated as a result of a prearranged event being received by the framework. Before the call is routed to the destination number, the calling party is asked for a PIN code. The code is rejected and the call is cleared.



1: This message is used by the application to create an object implementing the IpAppMultiPartyCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface.

4: This message is used to forward message 3 to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppMultiPartyCall interface. The reference to this object is passed back to the object implementing the IpMultiPartyCallControlManager using the return parameter of the callEventNotify.

6: The application requests an list of all the legs currently in the call.

7: This message is used to create a UICall object that is associated with the incoming leg of the call.

8: The call barring service dialogue is invoked.

9: The result of the dialogue, which in this case is the PIN code, is returned to its callback object.

10: This message is used to forward the previous message to the IpAppLogic

11: Assuming an incorrect PIN is entered, the calling party is informed using additional dialogue of the reason why the call cannot be completed.

12: This message passes the indication that the additional dialogue has been sent.

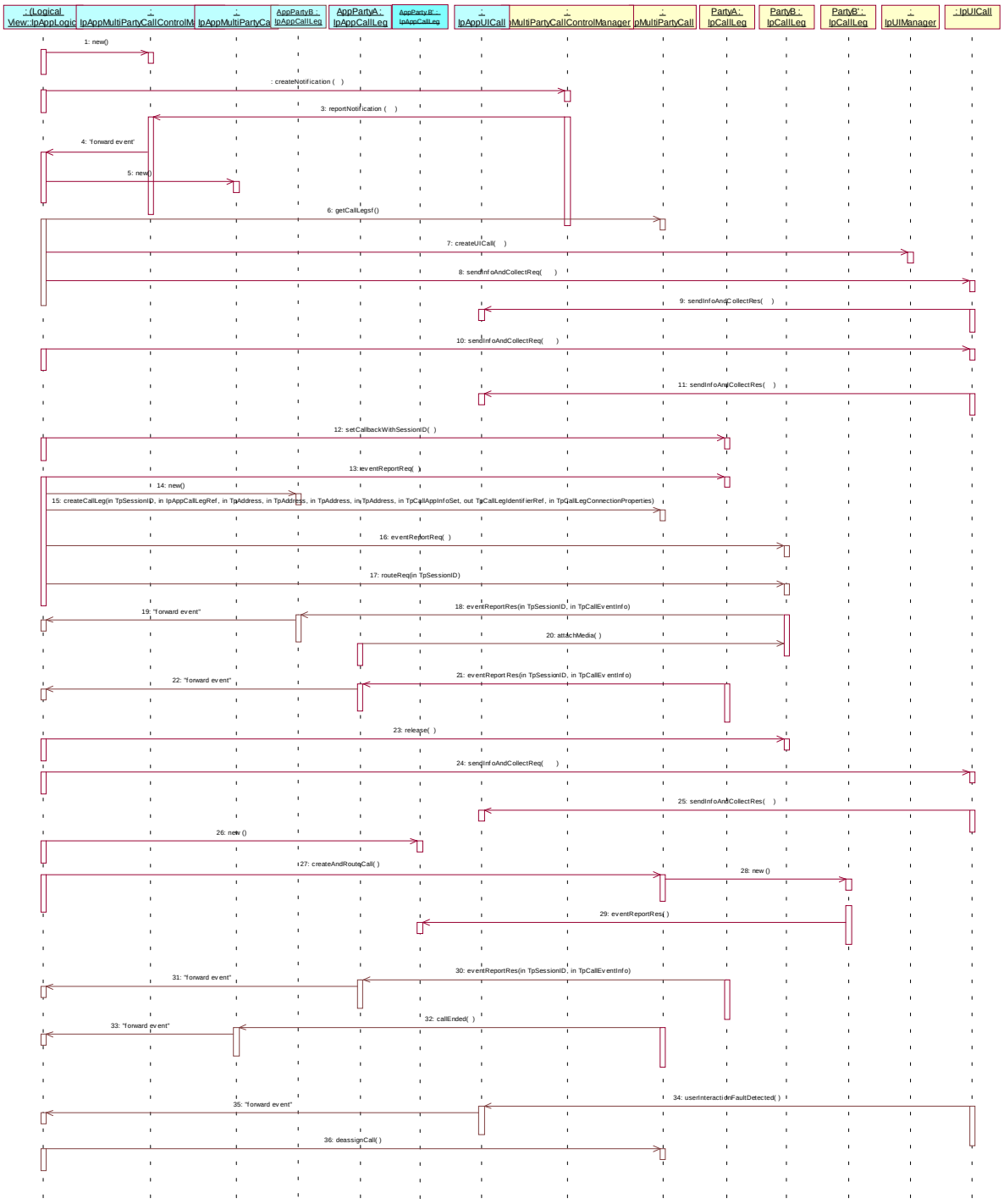
13: This message is used to forward the previous message to the IpAppLogic.

14: No more UI is required, so the UICall object is released.

15: This message is used by the application to clear the call.

7.1.3 Complex Card Service

The following sequence diagram shows an advanced card service, initiated as a result of a prearranged event being received by the framework. Before the call is made, the calling party is asked for an ID and PIN code. If the ID and PIN code are accepted, the calling party is prompted to enter the address of the destination party. A trigger of '#5' is then set on the controlling leg (the calling party's leg) such that if the calling party enters a '#5' an event will be sent to the application. The call is then routed to the destination party. Sometime during the call the calling party enters '#5' which causes the called leg to be released. The calling party is now prompted to enter the address of a new destination party, to which it is then routed.



1: This message is used by the application to create an object implementing the IpAppMultiPartyCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range result in the caller being prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall

interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface.

4: This message is used to forward message 3 to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppMultiPartyCall interface. The reference to this object is passed back to the object implementing the IpMultiPartyCallControlManager using the return parameter of message 3.

6: This message retuns the call legs currently in the call. In principle a reference to the call leg of the calling party is already obtained by the application when it was notified of the new call event.

7: This message is used to associate a user interaction object with the calling party.

8: The initial card service dialogue is invoked using this message.

9: The result of the dialogue, which in this case is the ID and PIN code, is returned to its callback object using this message and eventually forwarded via another message (not shown) to the IpAppLogic.

10: Assuming the correct ID and PIN are entered, the final dialogue is invoked.

11: The result of the dialogue, which in this case is the destination address, is returned and eventually forwarded via another message (not shown) to the IpAppLogic.

12: This message is used to forward the address of the callback object.

13: The trigger for follow-on calls is set (on service code).

14: A new AppCallLeg is created to receive callbacks for another leg. Alternatively, the already existing AppCallLeg object could be passed in the subsequent createCallLeg(). In that case the application has to use the sessionIDs of the legs to distinguish between callbacks destined for the A-leg and callbacks destined for the B-leg.

15: This message is used to create a new call leg object. The object is created in the idle state and not yet routed in the network.

16: The application requests to be notified when the leg is answered.

17: The application routes the leg. As a result the network will try to reach the associated party.

18: When the B-party answers the call, the application is notified.

19: The event is forwarded to the application logic.

20: Legs that are created and routed explicitly are by default in state detached. This means that the media is not connected to the other parties in the call. In order to allow inband communication between the new party and the other parties in the call the media have to be explicitly attached.

21: At some time during the call the calling party enters '#5'. This causes this message to be sent to the object implementing the IpAppCallLeg interface, which forwards this event as a message (not shown) to the IpAppLogic.

22: The event is forwarded to the application.

23: This message releases the called party.

24: Another user interaction dialogue is invoked.

25: The result of the dialogue, which in this case is the new destination address is returned and eventually forwarded via another message (not shown) to the IpAppLogic.

26: A new AppCallLeg is created to receive callbacks for another leg.

27: The call is then forward routed to the new destination party.

28: As a result a new Callleg object is created.

29: This message passes the result of the call being answered to its callback object and is eventually forwarded via another message (not shown) to the IpAppLogic.

30: When the A-party terminates the application is informed.

31: The event is forwarded to the application logic.

32: Since the release of the A-party will in this case terminate the entire call, the application is also notified with this message.

33: The event is forwarded to the application logic.

34: Since the user interaction object were not released at the moment that the call terminated, the application receives this message to indicate that the UI resources are released in the gateway and no further communication is possible.

35: The event is forwarded to the application logic.

36: The application deassigns the call object.

7.2 Class Diagrams

The multiparty call control service consists of two packages, one for the interfaces on the application side and one for interfaces on the service side.

The class diagrams in the following figures show the interfaces that make up the multi party call control application package and the multi party call control service package. This class diagram shows the interfaces of the multi-party call control application package and their relations to the interfaces of the multi-party call control service package.

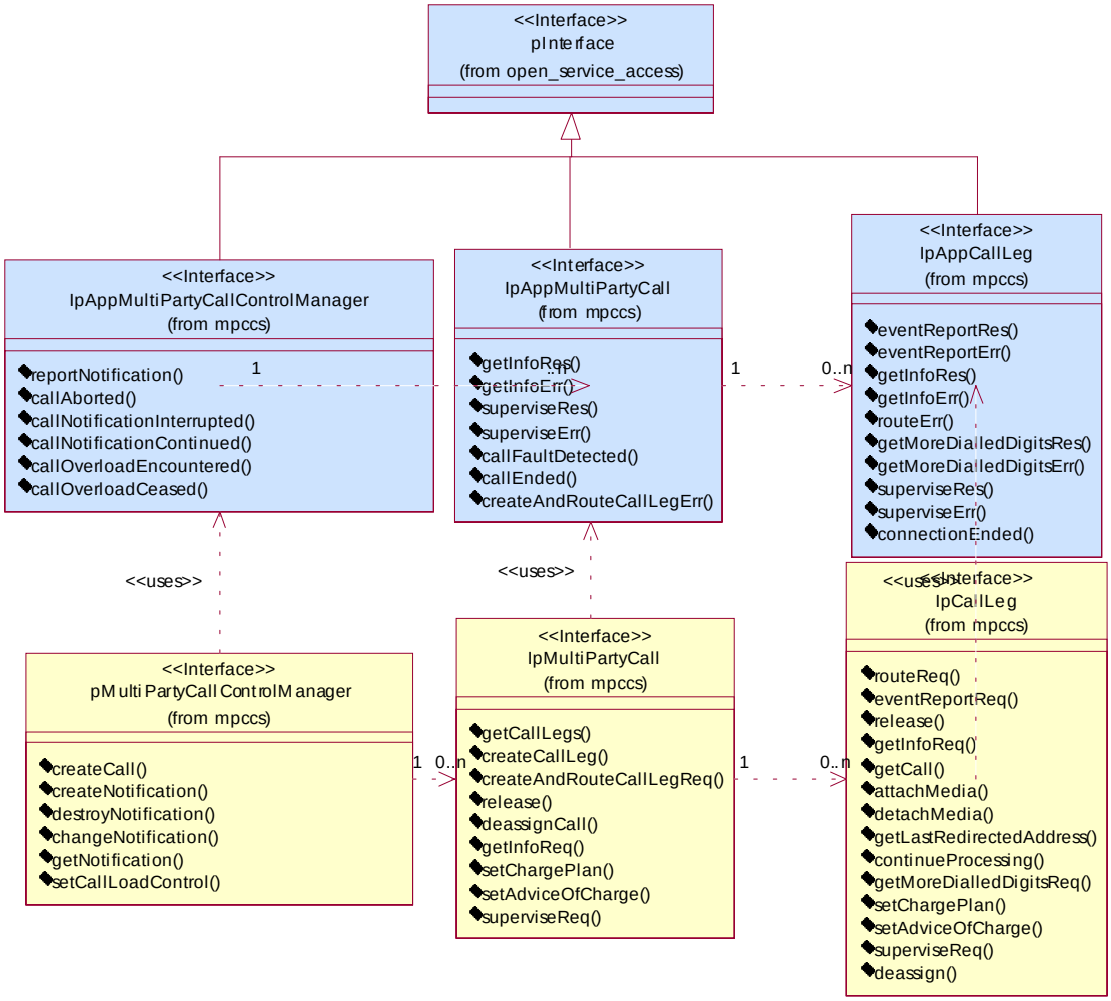


Figure: Application Interfaces

This class diagram shows the interfaces of the multi-party call control service package.

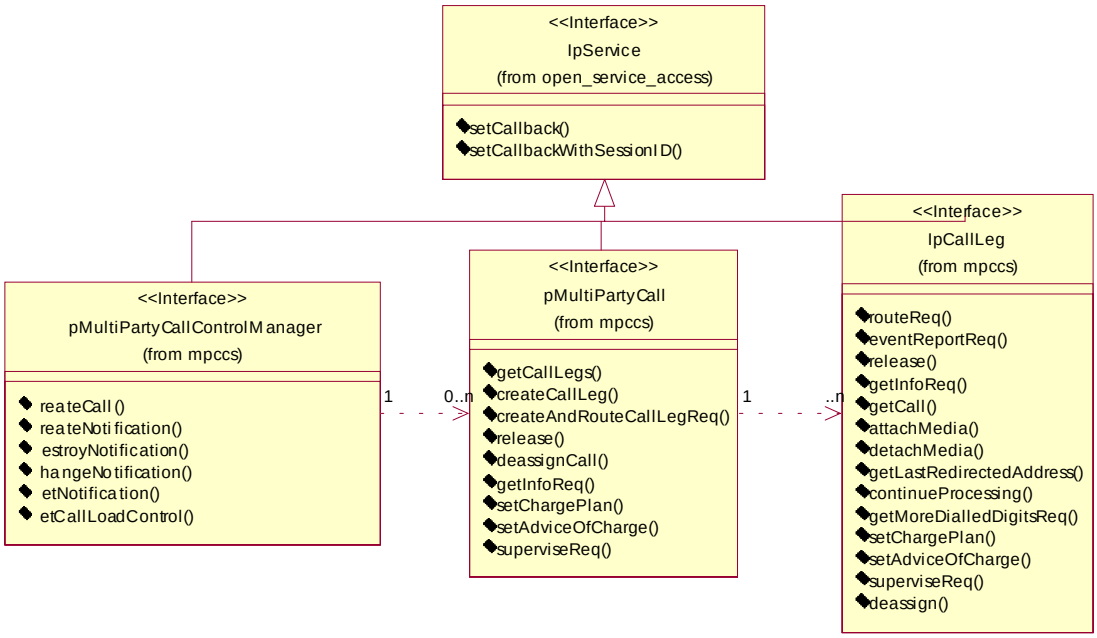


Figure: Service Interfaces

7.3 MultiParty Call Control Service Interface Classes

The Multi-party Call Control service enhances the functionality of the Generic Call Control Service with leg management. It also allows for multi-party calls to be established, i.e., up to a service specific number of legs can be connected simultaneously to the same call.

The Multi-party Call Control Service is represented by the IpMultiPartyCallControlManager, IpMultiPartyCall, IpCallLeg interfaces that interface to services provided by the network. Some methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppMultiPartyCallManager, IpAppMutliPartyCall and IpAppCallLeg to provide the callback mechanism.

7.3.1 Interface Class IpMultiPartyCallControlManager

Inherits from: IpService

This interface is the 'service manager' interface for the Multi-party Call Control Service. The multi-party call control manager interface provides the management functions to the multi-party call control service. The application programmer can use this interface to provide overload control functionality, create call objects and to enable or disable call-related event notifications.

<<Interface>> IpMultiPartyCallControlManager
createCall (appCall : in IpAppMultiPartyCallRef, callReference : out TpMultiPartyCallIdentifierRef) : TpResult createNotification (appCallControlManager : in IpAppMultiPartyCallControlManagerRef, notificationRequest

: in TpCallNotificationRequest, assignmentID : out TpAssignmentIDRef) : TpResult

destroyNotification (assignmentID : in TpAssignmentID) : TpResult

changeNotification (assignmentID : in TpAssignmentID, notificationRequest : in TpCallNotificationRequest) : TpResult

getNotification (notificationsRequested : out TpNotificationRequestedSetRef) : TpResult

setCallLoadControl (duration : in TpDuration, mechanism : in TpCallLoadControlMechanism, treatment : in TpCallTreatment, addressRange : in TpAddressRange, assignmentID : out TpAssignmentIDRef) : TpResult

Method

createCall()

This method is used to create a new call object.

Parameters

appCall : in IpAppMultiPartyCallRef

Specifies the application interface for callbacks from the call created.

callReference : out TpMultiPartyCallIdentifierRef

Specifies the interface reference and sessionID of the call created.

Raises

TpGCCSException,TpGeneralException

Method

createNotification()

This method is used to enable call notifications so that events can be sent to the application. If some application already requested notifications with criteria that overlap the specified criteria, the request is refused with P_GCCS_INVALID_CRITERIA.

The criteria are said to overlap if both originating and terminating ranges overlap and the same number plan is used and the same NotificationCallType is used.

If the same application requests two notifications with exactly the same criteria but different callback references, the second callback will be treated as an additional callback. This means that the callback will only be used in case when the first callback specified by the application is unable to handle the reportNotification (e.g., due to overload or failure).

Parameters

appCallControlManager : in IpAppMultiPartyCallControlManagerRef

If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the setCallback() method.

notificationRequest : in TpCallNotificationRequest

Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported. Examples of events are "incoming call attempt reported by network", "answer", "no answer", "busy". Individual addresses or address ranges may be specified for destination and/or origination.

assignmentID : out TpAssignmentIDRef

Specifies the ID assigned by the generic call control manager interface for this newly-enabled event notification.

Raises

TpGCCSException,TpGeneralException

Method

destroyNotification()

This method is used by the application to disable call notifications.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID given by the generic call control manager interface when the previous enableNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will return the error code P_INVALID_ASSIGNMENTID.

Raises

TpGCCSException,TpGeneralException

Method

changeNotification()

This method is used by the application to change the event criteria introduced with createNotification. Any stored criteria associated with the specified assignementID will be replaced with the specified criteria.

Parameters

assignmentID : in TpAssignmentID

Specifies the ID assigned by the generic call control manager interface for the event notification.

notificationRequest : in TpCallNotificationRequest

Specifies the new set of event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

Raises

TpGeneralException,TpGCCSException

Method

getNotification()

This method is used by the application to query the event criteria set with createNotification or changeNotification.

Parameters

notificationsRequested : out TpNotificationRequestedSetRef

Specifies the nofications that have been requested by the application.

Raises

TpGeneralException,TpGCCSException

Method

setCallLoadControl()

This method imposes or removes load control on calls made to a particular address range within the generic call control service. The address matching mechanism is similar as defined for TpCallEventCriteria.

Parameters

duration : in TpDuration

Specifies the duration for which the load control should be set.

A duration of 0 indicates that the load control should be removed.

A duration of -1 indicates an infinite duration (i.e., until disabled by the application)

A duration of -2 indicates the network default duration.

mechanism : in TpCallLoadControlMechanism

Specifies the load control mechanism to use (for example, admit one call per interval), and any necessary parameters, such as the call admission rate. The contents of this parameter are ignored if the load control duration is set to zero.

treatment : in TpCallTreatment

Specifies the treatment of calls that are not admitted. The contents of this parameter are ignored if the load control duration is set to zero.

addressRange : in TpAddressRange

Specifies the address or address range to which the overload control should be applied or removed.

assignmentID : out TpAssignmentIDRef

Specifies the assignmentID assigned by the gateway to this request. This assignementID can be used to correlate the callOverlloadEncountered and callOverloadCeased methods with the request.

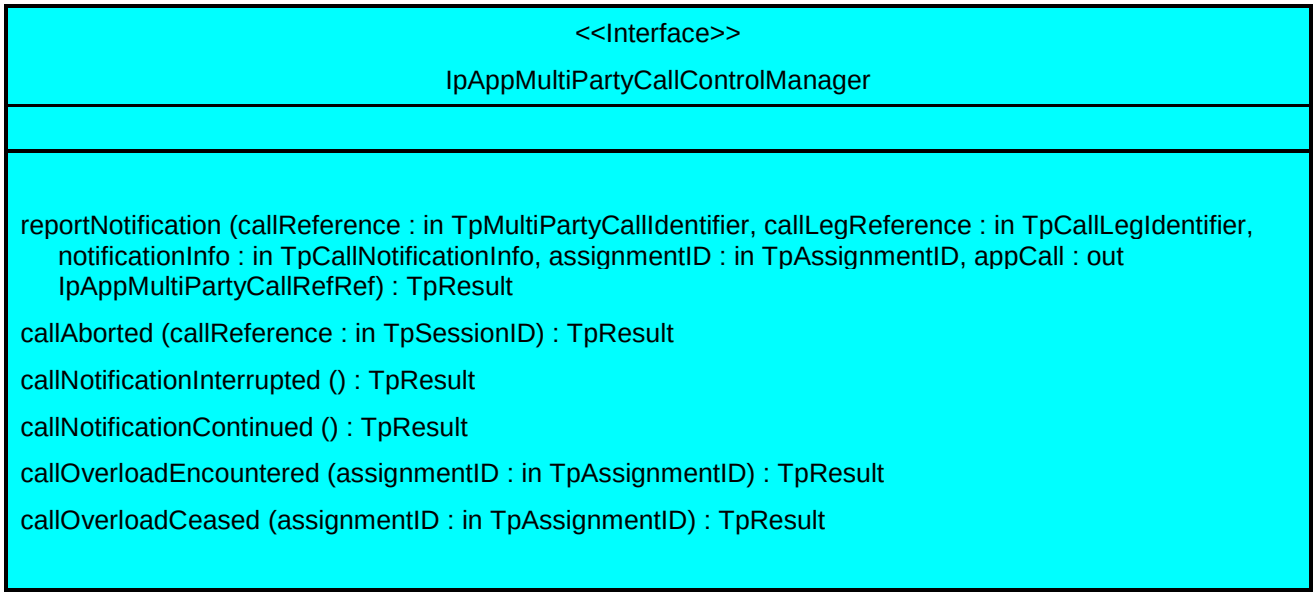
Raises

TpGeneralException,TpGCCSException

7.3.2 Interface Class IpAppMultiPartyCallControlManager

Inherits from: IpInterface

The Multi-Party call control manager application interface provides the application call control management functions to the Multi-Party call control service.



Method

`reportNotification()`

This method notifies the application of the arrival of a call-related event.

Parameters

`callReference : in TpMultiPartyCallIdentifier`

Specifies the reference to the call interface to which the notification relates.

`callLegReference : in TpCallLegIdentifier`

Specifies the reference to the callLeg interface to which the notification relates.

`notificationInfo : in TpCallNotificationInfo`

Specifies data associated with this event.

`assignmentID : in TpAssignmentID`

Specifies the assignment id which was returned by the createNotification() method. The application can use assignment id to associate events with event specific criteria and to act accordingly.

`appCall : out IpAppMultiPartyCallRefRef`

Specifies a reference to the application interface which implements the callback interface for the new call.

Raises

TpGCCSException,TpGeneralException

Method

callAborted()

This method indicates to the application that the call object (at the gateway) has aborted or terminated abnormally. No further communication will be possible between the call and application.

Parameters

callReference : in TpSessionID

Specifies the sessionID of call that has aborted or terminated abnormally.

Raises

TpGCCSException,TpGeneralException

Method

callNotificationInterrupted()

This method indicates to the application that all event notifications have been temporary interrupted (for example, due to faults detected).

Note that more permanent failures are reported via the Framework (integrity management).

Parameters

No Parameters were identified for this method

Raises

TpGCCSException,TpGeneralException

Method

callNotificationContinued()

This method indicates to the application that event notifications will again be possible.

Parameters

No Parameters were identified for this method

Method

callOverloadEncountered()

This method indicates that the network has detected overload and may have automatically imposed load control on calls requested to a particular address range or calls made to a particular destination within the call control service.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been encountered.

Raises

TpGeneralException,TpGCCSException

Method

callOverloadCeased()

This method indicates that the network has detected that the overload has ceased and has automatically removed any load controls on calls requested to a particular address range or calls made to a particular destination within the call control service.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been ceased

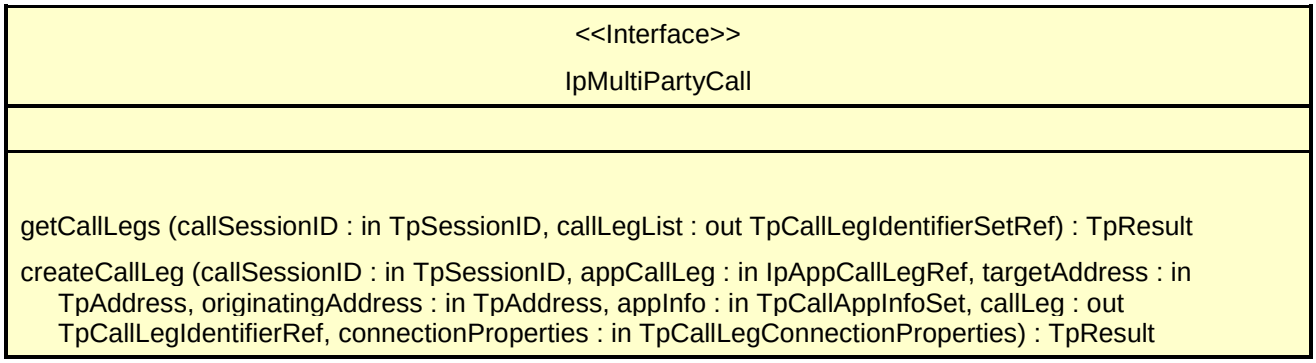
Raises

TpGeneralException,TpGCCSException

7.3.3 Interface Class IpMultiPartyCall

Inherits from: IpService

The Multi-Party Call provides the possibility to control the call routing, to request information from the call, control the charging of the call, to release the call and to supervise the call. It also gives the possibiltiy to manage call legs explicitly. Via the legs the application can also influence the media in multi-media calls. If an application uses the multi-party call control interface it may call the createAndRouteCallLeg() operation several times without disconnecting already connected destination. Therefore, an application may implicitly create more then one (destination) call leg. However, there can only be at most one call leg that owns the call ("call owner") at any time. In contrast to the conference service it is not possible to move legs to another call object.



createAndRouteCallLegReq (callSessionID : in TpSessionID, eventsRequested : in TpCallEventRequestSet, targetAddress : in TpAddress, originatingAddress : in TpAddress, appInfo : in TpCallAppInfoSet, appLegInterface : in IpAppCallLegRef, callLegReference : out TpCallLegIdentifierRef) : TpResult

release (callSessionID : in TpSessionID, cause : in TpCallReleaseCause) : TpResult

deassignCall (callSessionID : in TpSessionID) : TpResult

getInfoReq (callSessionID : in TpSessionID, callInfoRequested : in TpCallInfoType) : TpResult

setChargePlan (callSessionID : in TpSessionID, callChargePlan : in TpCallChargePlan) : TpResult

setAdviceOfCharge (callSessionID : in TpSessionID, aOCInfo : in TpAoCInfo, tariffSwitch : in TpDuration) : TpResult

superviseReq (callSessionID : in TpSessionID, time : in TpDuration, treatment : in TpCallSuperviseTreatment) : TpResult

Method

getCallLegs()

This method requests the identification of the call leg objects associated with the call object. Returns the legs in the order of creation.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callLegList : out TpCallLegIdentifierSetRef

Specifies the call legs associated with the call. The set contains both the sessionIDs and the interface references.

Raises

TpGCCSException, TpGeneralException

Method

createCallLeg()

This method requests the creation of a new call leg object.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

appCallLeg : in IpAppCallLegRef

Specifies the application interface for callbacks from the call leg created.

targetAddress : in TpAddress

Specifies the destination party to which the call should be routed.

originatingAddress : in TpAddress

Specifies the address of the originating (calling) party.

appInfo : in TpCallAppInfoSet

Specifies application-related information pertinent to the call leg (such as alerting method, tele-service type, service identities and interaction indicators).

callLeg : out TpCallLegIdentifierRef

Specifies the interface and sessionID of the call leg created.

connectionProperties : in TpCallLegConnectionProperties

Specifies the properties of the connection.

Raises

TpGeneralException,TpGCCSException

Method

createAndRouteCallLegReq()

This asynchronous operation requests creation and routing of a new callLeg. In case the connection to the destination party is established successfully the CallLeg is attached to the call, i.e. no explicit attachMedia() operation is needed. Requested events will be reported on the IpAppCallLeg interface. This interface the application must provide through the appLegInterface parameter.

The extra address information such as originatingAddress is optional. If not present (i.e., the plan is set to P_ADDRESS_PLAN_NOT_PRESENT), the information provided in corresponding addresses from the route is used, otherwise the network or gateway provided numbers will be used.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

eventsRequested : in TpCallEventRequestSet

Specifies the event specific criteria used by the application to define the events required. Only events that meet these criteria are reported. Examples of events are "adress analysed", "answer", "release".

targetAddress : in TpAddress

Specifies the destination party to which the call should be routed.

originatingAddress : in TpAddress

Specifies the address of the originating (calling) party.

appInfo : in TpCallAppInfoSet

Specifies application-related information pertinent to the call (such as alerting method, tele-service type, service identities and interaction indicators).

appLegInterface : in IpAppCallLegRef

Specifies a reference to the application interface that implements the callback interface for the new call leg. Requested events will be reported by the eventReportRes() operation on this interface.

callLegReference : out TpCallLegIdentifierRef

Specifies the reference to the CallLeg interface that was created.

Raises

TpGCCSException,TpGeneralException

Method

release()

This method requests the release of the call object and associated objects. The call will also be terminated in the network. If the application requested reports to be sent at the end of the call (e.g., by means of getInfoReq) these reports will still be sent to the application.

The application should always either release or deassign the call when it is finished with the call, unless a callFaultDetected is received by the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

cause : in TpCallReleaseCause

Specifies the cause of the release.

Raises

TpGCCSException,TpGeneralException

Method

deassignCall()

This method requests that the relationship between the application and the call and associated objects be de-assigned. It leaves the call in progress, however, it purges the specified call object so that the application has no further control of call processing. If a call is de-assigned that has call information reports, call leg event reports or call Leg information reports requested, then these reports will be disabled and any related information discarded.

The application should always either release or deassign the call when it is finished with the call, unless callFaultDetected is received by the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

Raises

TpGCCSException,TpGeneralException

Method

getInfoReq()

This asynchronous method requests information associated with the call to be provided at the appropriate time (for example, to calculate charging). This method must be invoked before the call is routed to a target address. Two types of reports can be requested; a final report or intermediate reports.

A final call report is sent when the call is ended. The call object will exist after the call is ended if information is required to be sent to the application at the end of the call. The call information will be sent after any call event reports.

Intermediate reports are received when the destination leg or party terminates or when the call ends. In case the originating party is still available the application can still initiate a follow-on call.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callInfoRequested : in TpCallInfoType

Specifies the call information that is requested.

Raises

TpGCCSException,TpGeneralException

Method

setChargePlan()

Set an operator specific charge plan for the call. The charge plan must be set before the call is routed to a target address. Depending on the operator the method can also be used to change the charge plan for ongoing calls.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callChargePlan : in TpCallChargePlan

Specifies the charge plan to use.

Raises

TpGCCSException,TpGeneralException

Method

setAdviceOfCharge()

This method allows for advice of charge (AOC) information to be sent to terminals that are capable of receiving this information.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

aOCInfo : in TpAoCInfo

Specifies two sets of Advice of Charge parameter.

tariffSwitch : in TpDuration

Specifies the tariff switch interval that signifies when the second set of AoC parameters becomes valid.

Raises

TpGeneralException,TpGCCSException

Method

superviseReq()

The application calls this method to supervise a call. The application can set a granted connection time for this call. If an application calls this operation before it routes a call or a user interaction operation the time measurement will start as soon as the call is answered by the B-party or the user interaction system.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

time : in TpDuration

Specifies the granted time in milliseconds for the connection.

treatment : in TpCallSuperviseTreatment

Specifies how the network should react after the granted connection time expired.

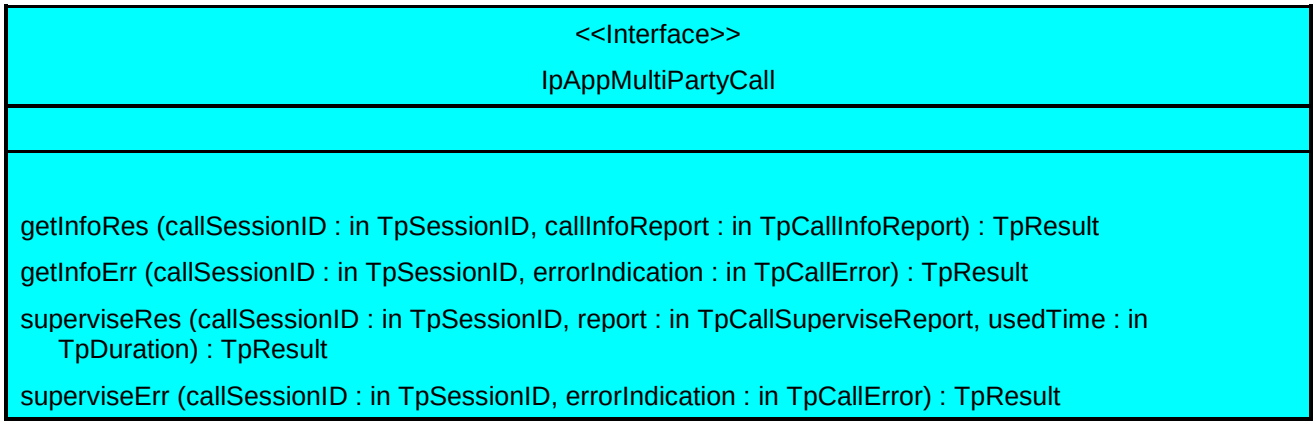
Raises

TpGCCSException,TpGeneralException

7.3.4 Interface Class IpAppMultiPartyCall

Inherits from: IpInterface

The Multi-Party call application interface is implemented by the client application developer and is used to handle call request responses and state reports.



callFaultDetected (callSessionID : in TpSessionID, fault : in TpCallFault) : TpResult

callEnded (callSessionID : in TpSessionID, report : in TpCallEndedReport) : TpResult

createAndRouteCallLegErr (callSessionID : in TpSessionID, callLegReference : in TpCallLegIdentifier, errorIndication : in TpCallError) : TpResult

Method

getInfoRes()

This asynchronous method reports time information of the finished call or call attempt as well as release cause depending on which information has been requested by getInfoReq. This information may be used e.g. for charging purposes. The call information will possibly be sent after reporting of all cases where the call or a leg of the call has been disconnected or a routing failure has been encountered.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callInfoReport : in TpCallInfoReport

Specifies the call information requested.

Raises

TpGCCSException,TpGeneralException

Method

getInfoErr()

This asynchronous method reports that the original request was erroneous, or resulted in an error condition.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGCCSException,TpGeneralException

Method

superviseRes()

This asynchronous method reports a call supervision event to the application when it has indicated it's interest in these kind of events.

It is also called when the connection is terminated before the supervision event occurs. Furthermore, this method is invoked as a response to the request also when a tariff switch happens in the network during an active call.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call

report : in TpCallSuperviseReport

Specifies the situation which triggered the sending of the call supervision response.

usedTime : in TpDuration

Specifies the used time for the call supervision (in milliseconds).

Raises

TpGCCSException,TpGeneralException

Method

superviseErr()

This asynchronous method reports a call supervision error to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGCCSException,TpGeneralException

Method

callFaultDetected()

This method indicates to the application that a fault in the network has been detected. The call may or may not have been terminated.

The system deletes the call object. Therefore, the application has no further control of call processing. No report will be forwarded to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call in which the fault has been detected.

fault : in TpCallFault

Specifies the fault that has been detected.

Raises

TpGCCSException,TpGeneralException

Method

callEnded()

This method indicates to the application that the call has terminated in the network. However, the application may still receive some results (e.g., getInfoRes) related to the call. The application is expected to deassign the call object after having received the callEnded.

Note that the event that caused the call to end might also be received separately if the application was monitoring for it.

Parameters

callSessionID : in TpSessionID

Specifies the call sessionID.

report : in TpCallEndedReport

Specifies the reason the call is terminated.

Raises

TpGeneralException,TpGCCSException

Method

createAndRouteCallLegErr()

This asynchronous method indicates that the request to route the call to the destination party was unsuccessful - the call could not be routed to the destination party (for example, the network was unable to route the call, the parameters were incorrect, the request was refused, etc.). Note that the event cases that can be monitored and correspond to an unsuccessful setup of a connection (e.g. busy, no_answer) will be reported by eventReportRes() and not by this operation.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callLegReference : in TpCallLegIdentifier

Specifies the reference to the CallLeg interface that was created.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGCCSException,TpGeneralException

7.3.5 Interface Class IpCallLeg

Inherits from: IpService

The call leg interface represents the logical call leg associating a call with an address. The call leg tracks its own states and allows charging summaries to be accessed. The leg represents the signalling relationship between the call and an address. An application that uses the IpCallLeg interface to set up connections has more control, e.g. by defining leg specific event request and can obtain call leg specific report and events.

<<Interface>> IpCallLeg
routeReq (callLegSessionID : in TpSessionID) : TpResult eventReportReq (callLegSessionID : in TpSessionID, eventsRequested : in TpCallEventRequestSet) : TpResult release (callLegSessionID : in TpSessionID, cause : in TpCallReleaseCause) : TpResult getInfoReq (callLegSessionID : in TpSessionID, callLegInfoRequested : in TpCallLegInfoType) : TpResult getCall (callLegSessionID : in TpSessionID, callReference : out TpMultiPartyCallIdentifierRef) : TpResult attachMedia (callLegSessionID : in TpSessionID) : TpResult detachMedia (callLegSessionID : in TpSessionID) : TpResult getLastRedirectedAddress (callLegSessionID : in TpSessionID, redirectedAddress : out TpAddressRef) : TpResult continueProcessing (callLegSessionID : in TpSessionID) : TpResult getMoreDialledDigitsReq (callLegSessionID : in TpSessionID, length : in TpInt32) : TpResult setChargePlan (callLegSessionID : in TpSessionID, callChargePlan : in TpCallChargePlan) : TpResult setAdviceOfCharge (callLegSessionID : in TpSessionID, aOCInfo : in TpAoCInfo, tarrifSwitch : in TpDuration) : TpResult superviseReq (callLegSessionID : in TpSessionID, time : in TpDuration, treatment : in TpCallSuperviseTreatment) : TpResult deassign (callLegSessionID : in TpSessionID) : TpResult

Method

routeReq()

This asynchronous method requests routing of the call leg to the remote party indicated by the routingAddress.

The extra address information (like originalDestinationAddress) is optional and may be set to unavailable (i.e., the plan is set to P_ADDRESS_PLAN_NOT_PRESENT). In this case information provided when routing to the origination will be used if applicable. Otherwise network or gateway provided addresses will be used.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

Raises

TpGeneralException,TpGCCSException

Method

eventReportReq()

This asynchronous method sets, clears or changes the criteria for the events that the call leg object will be set to observe.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

eventsRequested : in TpCallEventRequestSet

Specifies the event specific criteria used by the application to define the events required. Only events that meet these criteria are reported. Examples of events are "address analysed", "answer", "release".

Raises

TpGeneralException,TpGCCSException

Method

release()

This method requests the release of the call leg. If successful, the associated address (party) will be released from the call, and the call leg deleted. Note that in some cases releasing the party may lead to release of the complete call in the network. The application will be informed of this with callEnded().

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

cause : in TpCallReleaseCause

Specifies the cause of the release.

Raises

TpGeneralException,TpGCCSException

Method
getInfoReq()

This asynchronous method requests information associated with the call leg to be provided at the appropriate time (for example, to calculate charging). Note: in the call leg information must be accessible before the objects of concern are deleted.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

callLegInfoRequested : in TpCallLegInfoType

Specifies the call leg information that is requested.

Raises

TpGeneralException,TpGCCSException

Method
getCall()

This method requests the call associated with this call leg.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

callReference : out TpMultiPartyCallIdentifierRef

Specifies the interface and sessionID of the call associated with this call leg.

Raises

TpGeneralException,TpGCCSException

Method
attachMedia()

This method requests that the call leg be attached to its call object. This will allow transmission on all associated bearer connections or media channels to and from other parties in the call. The call leg must be in the connected state for this method to complete successfully.

Parameters

callLegSessionID : in TpSessionID

Specifies the sessionID of the call leg to attach to the call.

Raises

TpGeneralException,TpGCCSException

Method

detachMedia()

This method will detach the call leg from its call, i.e., this will prevent transmission on any associated bearer connections or media channels to and from other parties in the call. The call leg must be in the connected state for this method to complete successfully.

Parameters

callLegSessionID : in TpSessionID

Specifies the sessionID of the call leg to detach from the call.

Raises

TpGeneralException,TpGCCSException

Method

getLastRedirectedAddress()

Queries the last address the leg has been redirected to.

Parameters

callLegSessionID : in TpSessionID

Specifies the call session ID of the call leg.

redirectedAddress : out TpAddressRef

Specifies the last address where the call leg was redirected to.

Raises

TpGeneralException,TpGCCSException

Method

continueProcessing()

This operation continues processing of the call leg. Applications can invoke this operation after call leg processing was interrupted due to detection of a notification or event the application subscribed it's interest in.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

Raises

TpGeneralException,TpGCCSException

Method

getMoreDialledDigitsReq()

This asynchronous method requests to collect further digits and return them to the application. Depending on the administered data, the network may indicate a new call to the gateway if a caller goes off-hook or dialled only a few digits. The application then gets a new call event which contains no digits or only the few dialled digits in the event data. The application should then use this method if it requires more dialled digits, e.g. to perform screening.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call.

length : in TpInt32

Specifies the maximum number of digits to collect.

Raises

TpGeneralException, TpGCCSException

Method

setChargePlan()

Set an operator specific charge plan for the cal leg. The charge plan must be set before the call leg is routed to a target address. Depending on the operator the method can also be used to change the charge plan for ongoing calls.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call party.

callChargePlan : in TpCallChargePlan

Specifies the charge plan to use.

Raises

TpGeneralException,TpGCCSException

Method

setAdviceOfCharge()

This method allows for advice of charge (AOC) information to be sent to terminals that are capable of receiving this information.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call party.

aOCInfo : in TpAoCInfo

Specifies two sets of Advice of Charge parameter.

tarrifSwitch : in TpDuration

Specifies the tariff switch interval that signifies when the second set of AoC parameters becomes valid.

Raises

TpGeneralException,TpGCCSException

Method

superviseReq()

The application calls this method to supervise a call leg. The application can set a granted connection time for this call. If an application calls this function before it calls a routeReq() or a user interaction function the time measurement will start as soon as the call is answered by the B-party or the user interaction system.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call party.

time : in TpDuration

Specifies the granted time in milliseconds for the connection.

treatment : in TpCallSuperviseTreatment

Specifies how the network should react after the granted connection time expired.

Raises

TpGeneralException,TpGCCSException

Method

deassign()

This method requests that the relationship between the application and the call leg and associated objects be de-assigned. It leaves the call leg in progress, however, it purges the specified call leg object so that the application has no further control of call leg processing. If a call leg is de-assigned that has event reports or call leg information reports requested, then these reports will be disabled and any related information discarded.

The application should always either release or deassign the call leg when it is finished with the call, leg unless callFaultDetected is received by the application.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

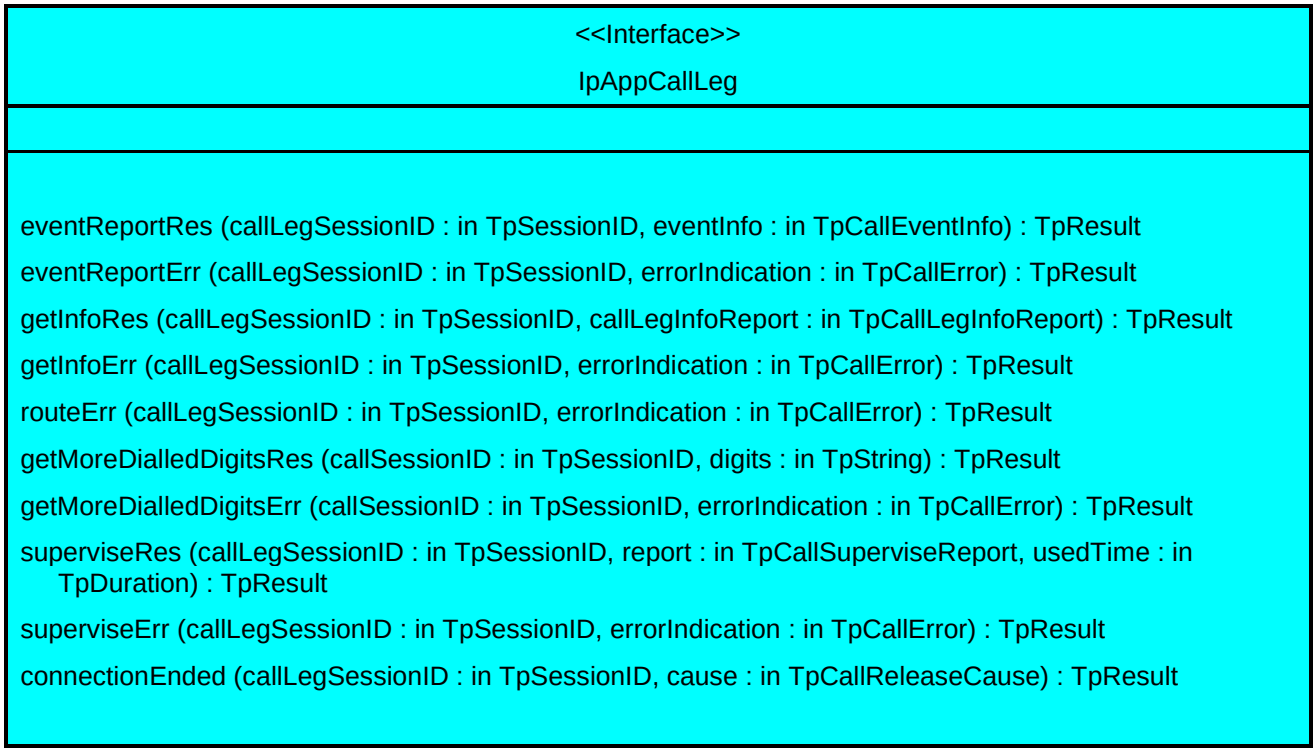
Raises

TpGeneralException,TpGCCSException

7.3.6 Interface Class IpAppCallLeg

Inherits from: IpInterface

The application call leg interface is implemented by the client application developer and is used to handle responses and errors associated with requests on the call leg in order to be able to receive leg specific information and events.



Method

eventReportRes()

This asynchronous method reports that an event has occurred that was requested to be reported (for example, a mid-call event, the party has requested to disconnect, etc.).

Depending on the type of event received, outstanding requests for events are discarded. The exact details of these so-called disarming rules are captured in the data definition of the event type.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg on which the event was detected.

eventInfo : in TpCallEventInfo

Specifies data associated with this event.

Raises

TpGeneralException,TpGCCSException

Method

eventReportErr()

This asynchronous method indicates that the request to manage call leg event reports was unsuccessful, and the reason (for example, the parameters were incorrect, the request was refused, etc.).

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGeneralException,TpGCCSException

Method

getInfoRes()

This asynchronous method reports all the necessary information requested by the application, for example to calculate charging.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg to which the information relates.

callLegInfoReport : in TpCallLegInfoReport

Specifies the call leg information requested.

Raises

TpGeneralException,TpGCCSException

Method

getInfoErr()

This asynchronous method reports that the original request was erroneous, or resulted in an error condition.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGeneralException,TpGCCSException

Method

routeErr()

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGCCSException,TpGeneralException

Method

getMoreDialledDigitsRes()

This asynchronous method returns the collected digits to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

digits : in TpString

Specifies the additional dialled digits if the string length is greater than zero.

Raises

TpGeneralException,TpGCCSException

Method

getMoreDialledDigitsErr()

This asynchronous method reports an error in collecting digits to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGeneralException,TpGCCSException

Method

superviseRes()

This asynchronous method reports a call leg supervision event to the application when it has indicated its interest in these kind of events.

It is also called when the connection to a party is terminated before the supervision event occurs. Furthermore, this method is invoked as a response to the request also when a tariff switch happens in the network during an active call.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg

report : in TpCallSuperviseReport

Specifies the situation which triggered the sending of the call leg supervision response.

usedTime : in TpDuration

Specifies the used time for the call leg supervision (in milliseconds).

Raises

TpGCCSException,TpGeneralException

Method

superviseErr()

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGCCSException,TpGeneralException

Method

connectionEnded()

This method indicates to the application that the connection has terminated in the network. However, the application may still receive some results (e.g., getInfoRes) related to the call leg. The application is expected to deassign the call leg object after having received the connectionEnded.

Note that the event that caused the connection to end might also be received separately if the application was monitoring for it.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

cause : in TpCallReleaseCause

Specifies the reason the connection is terminated.

Raises

TpGeneralException,TpGCCSException

7.4 MultiParty Call Control Service State Transition Diagrams

7.4.1 State Transition Diagrams for IpMultiPartyCallControlManager

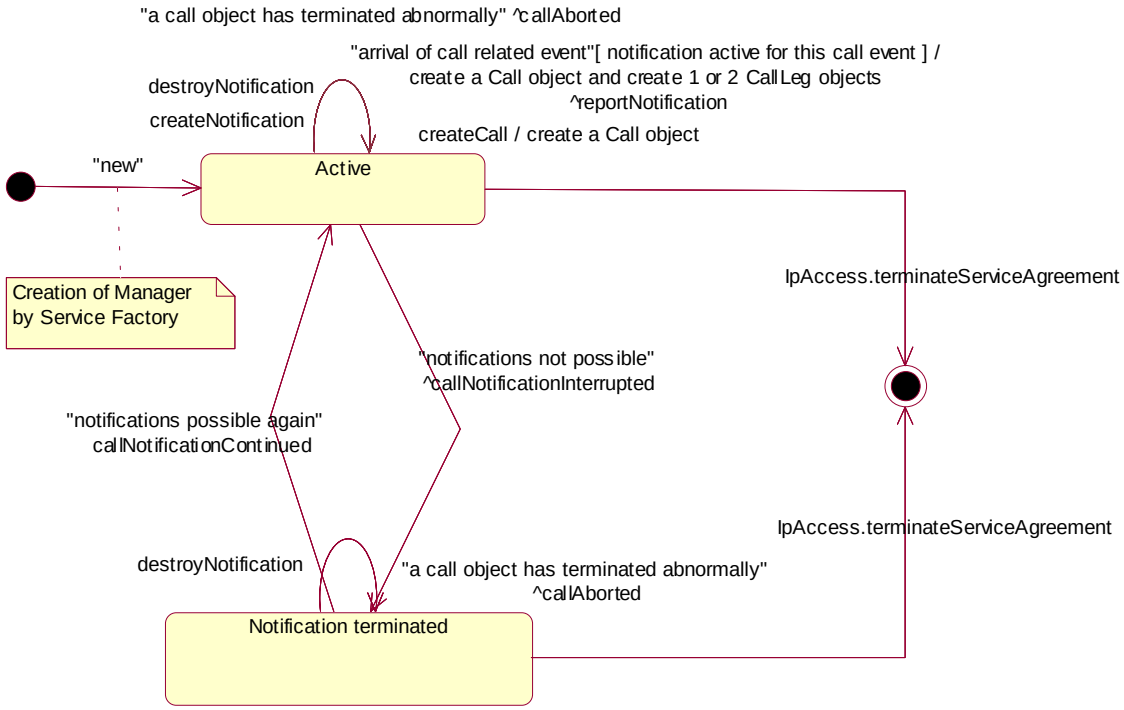


Figure : Application view and the Multi-Party Call Control Manager

7.4.1.1 Active State

In this state a relation between the Application and the Service has been established. The state allows the application to indicate that it is interested in call related events. In case such an event occurs, the Manager will create a Call object, depending on the specific event create 1 or 2 call Leg objects and inform the application.

The application can also indicate it is no longer interested in certain call related events by calling destroyNotification()..

7.4.1.2 Notification terminated State

When the Manager is in the Notification terminated state, events requested will not be forwarded to the application. There can be multiple reasons for this: for instance it might be that the application receives more notifications from the network than defined in the Service Level Agreement. Another example is that the Service has detected it receives no notifications from the network due to e.g. a link failure. In this state no requests for new notifications will be accepted.

7.4.2 State Transition Diagrams for IpMultiPartyCall

The state transition diagram shows the application view on the MultiParty Call object. The diagram is an extension to the state diagram of the Call object in the sense that more than 2 parties are allowed to participate in a call.

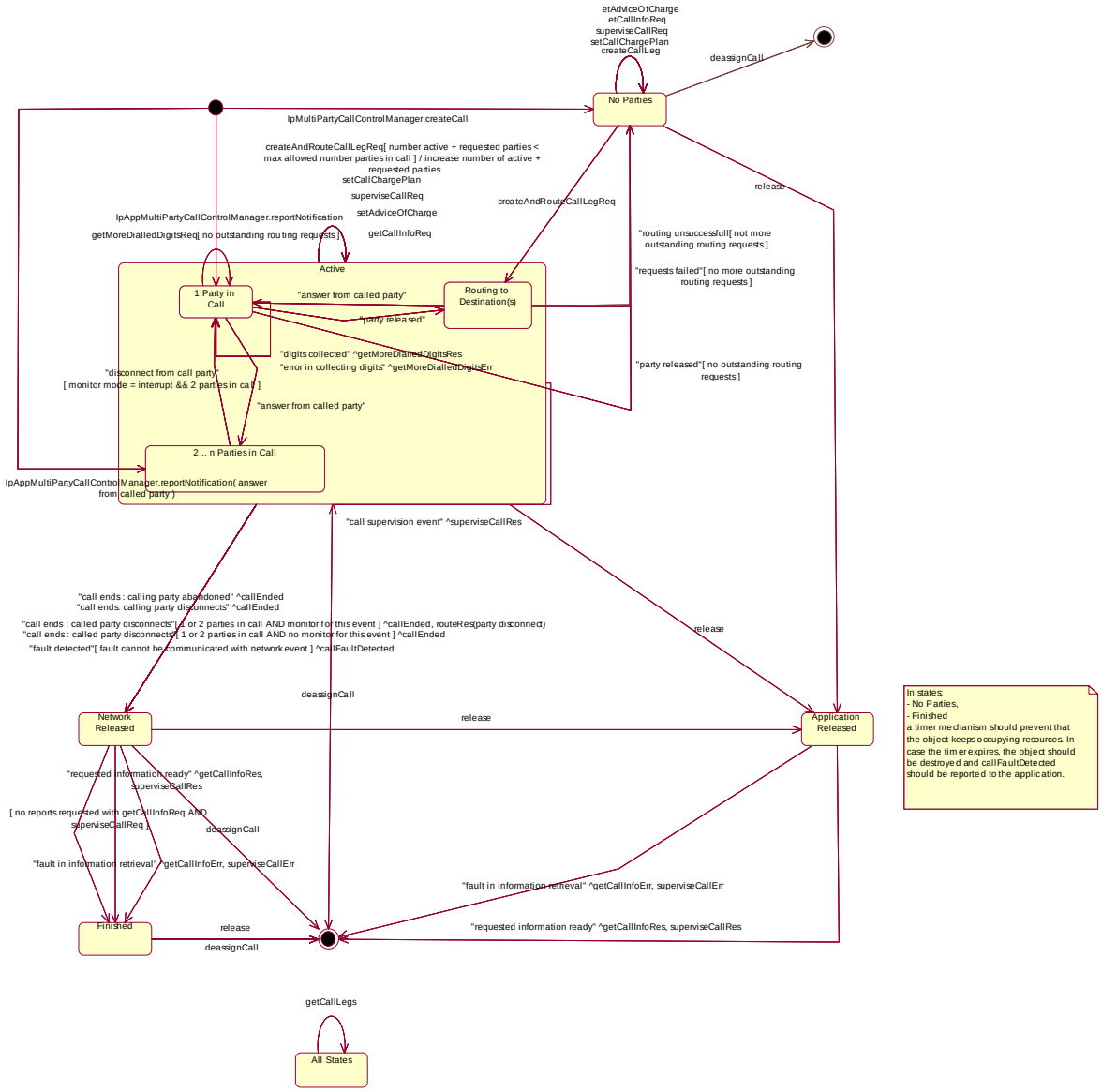


Figure : Application view on the MultiParty Call object

7.4.2.1 Active State

In this state a call between two parties is being setup or present. Refer to the substates for more details.

The application can request the gateway for a certain type of charging of the call by calling setCallChargePlan(). The application can request for charging related information by calling getCallInfoReq(). Furthermore the application can request supervision of the call by calling superviseCallReq(). It is also allowed to send Advice of Charge information by calling setAdviceOfCharge().

7.4.2.2 Network Released State

In this state the call has ended and the Gateway collects the possible call information requested with getCallInfoReq() and / or superviseCallReq(). In case the application has not requested additional call related information a transition to the Idle state is made immediately.

7.4.2.3 No Parties State

In this state the Call object has been created. The application can request the gateway for a certain type of charging of the call by calling setCallChargePlan(). The application can request for charging related information by calling getCallInfoReq(). Furthermore the application can request supervision of the call by calling superviseCallReq().

7.4.2.4 Application Released State

In this state the application has requested to release the Call object and the Gateway collects the possilbe call information requested with getCallInfoReq() and / or superviseCallReq(). In case the application has not requested additional call related information the Call object is destroyed immediately.

7.4.2.5 Finished State

In this state the call has ended and no call related information is to be send to the application. The application can only release the call object. Calling the deassignCall() operation has the same effect. Note that the application has to release the object itself as good OO practice requires that when an object was created on behalf of a certain entity, this entity is also responsible for destroying it when the object is no longer needed.

7.4.2.6 2 .. n Parties in Call State

In this state a successful connection between at least two parties is established.

In this state user interaction is possible, depending on the underlying network.

7.4.2.7 1 Party in Call State

In this state there is one party in the call.

In case the call originated from the network the application can now request for more digits in case the address is not yet complete or the application can request for a connection to a called party be established by calling the operation createAndRouteCallLegReq().

In case the called party was reached by issueing a routing request, the application can request a connection to an additional party by calling the operation createAndRouteCallLegReq() again.

Otherwise, it depends on the actual number of invoked (and still outstanding or successful) routing requests whether the application can still setup a connection to another called party. Also in this case the called party can disconnect before another party is reached. In this case depending on the actual configuration, either the call is ended or a transition is made back to the Routing to Destinations substate or the No Parties state, depending on whether there are outstanding routing requests.

In this state user interaction is possible.

7.4.2.8 Routing to Destination(s) State

In this state there is at least one outstanding routing request.

7.4.3 State Transition Diagrams for IpCallLeg

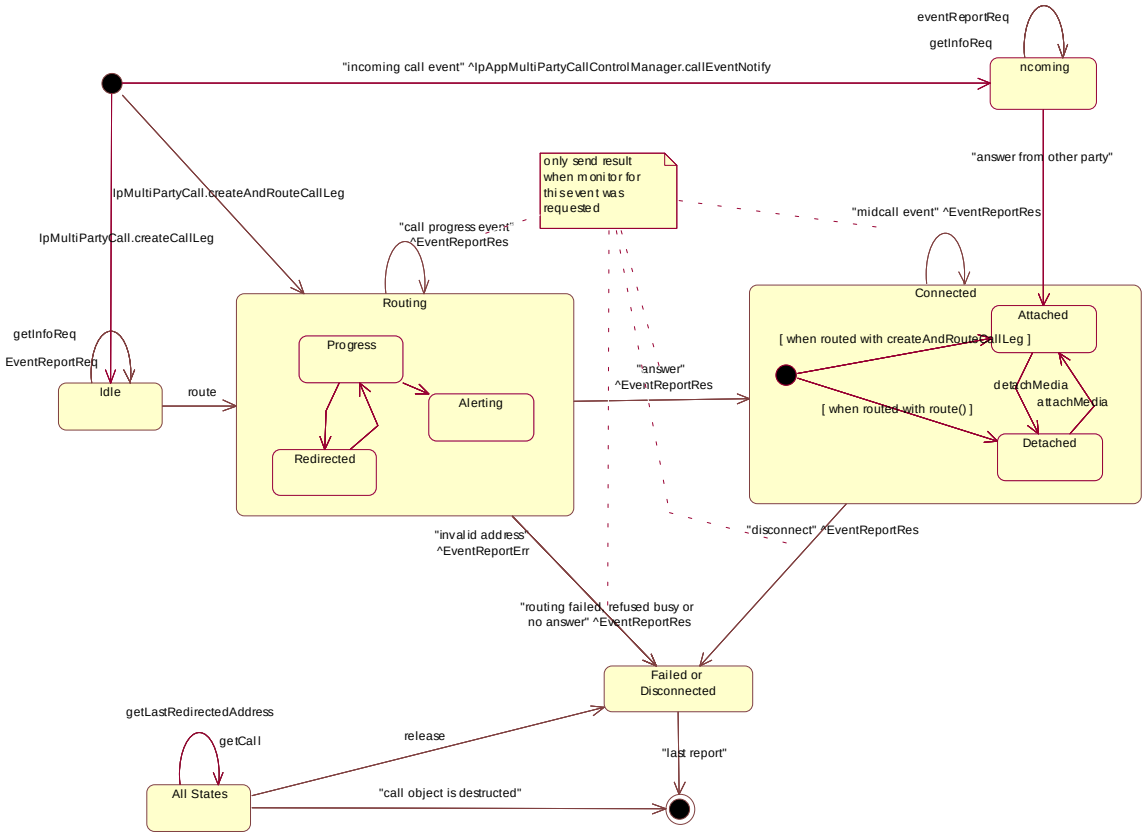


Figure : Application view on the CallLeg object

7.4.3.1 Idle State

In this state a new CallLeg object has been created and the application has not yet issued a routing request.

7.4.3.2 Routing State

In this state a connection to the call party is being established.

7.4.3.3 Connected State

In this state a connection to the call party is established.

In case the request for the connection was made by createAndRouteCallLeg on the Call object, the call party is also attached to the Call.

In case the request was made by route() the call party still needs to be attached to the Call.

7.4.3.4 Failed or Disconnected State

In this state no connection to the call party could be established or the call party has disconnected.

The reason that no connection could be established can be that an invalid address was specified, the network aborted routing or the call party was busy.

7.4.3.5 Incoming State

This state is only valid for an incoming Call Leg in case and there is no call established to another party.

7.4.3.6 Progress State

In this sub-state the network has indicated there is progress in routing the CallLeg.

7.4.3.7 Alerting State

In this sub-state the network has indicated there the terminal of the party is alerting.

7.4.3.8 Redirected State

In this sub-state the network has indicated the call party has redirected calls to another address.

7.4.3.9 Attached State

In this sub-state the media of the Call Leg object is attached to a Call object.

7.4.3.10 Detached State

In this sub-state the media of the Call Leg object is not attached to a Call object.

7.5 Multi-Party Call Control Service Properties

The following table lists properties relevant for the Multi-Party Call Control API. These properties are additional to the properties of the generic Call Control, from which the Multi-Party Call Control is an extension.

Property	Type	Description
P_MAX_CALLLEGS_PER_CALL	INTEGER_SET	Indicates how many parties can be in one call.
P_UI_CALLLEG_BASED	BOOLEAN_SET	Value = TRUE : User interaction can be performed on leg level and a reference to a CallLeg object can be used in the IpUIManager.createUICall() operation. Value = FALSE : No user interaction on leg level is supported.
P_ROUTING_WITH_CALLLEG_OPERATIONS	BOOLEAN_SET	Value = TRUE : the atomic operations for routing a CallLeg are supported {IpMultiPartyCall.createCallLeg(), IpCallLeg.eventReportReq(), IpCallLeg.route(), IpCallLeg.attachMedia()} Value = FALSE : the convenience function has to be used for routing a CallLeg.
P_MEDIA_ATTACH_EXPLICIT	BOOLEAN_SET	Value = TRUE : the CallLeg must be explicitly attached to a Call. Value = FALSE : the CallLeg is automatically attached to a Call, no IpCallLeg.attachMedia() is needed when a party answers.

7.6 Multi-Party Call Control Data Definitions

This document provides the generic call control data definitions necessary to support the API specification.

The general format of a data definition specification is described below.

- Data Type

This shows the name of the data type.

- Description

This describes the data type.

- Tabular Specification

This specifies the data types and values of the data type.

- Example

If relevant, an example is shown to illustrate the data type.

7.6.1 Event Notification Data Definitions

No specific event notification data defined.

7.6.2 Multi-Party Call Control Data Definitions

IpCallLeg

Defines the address of an `IpCallLeg` Interface.

IpCallLegRef

Defines a `Reference` to type IpCallLeg.

IpCallLegRefRef

Defines a `Reference` to type IpCallLegRef.

IpAppCallLeg

Defines the address of an `IpAppCallLeg` Interface.

IpAppCallLegRef

Defines a `Reference` to type IpAppCallLeg.

IpMultiPartyCall

Defines the address of an `IpMultiPartyCall` Interface.

IpMultiPartyCallRef

Defines a `Reference` to type IpMultiPartyCall.

IpAppMultiPartyCall

Defines the address of an `IpMultiPartyCall` Interface.

IpAppMultiPartyCallRef

Defines a Reference to type IpMultiPartyCall.

IpMultiPartyCallControlManager

Defines the address of an IpMultiPartyCall Interface.

IpMultiPartyCallControlManagerRef

Defines a Reference to type IpMultiPartyCall.

IpAppMultiPartyCallControlManager

Defines the address of an IpMultiPartyCall Interface.

IpAppMultiPartyCallControlManagerRef

Defines a Reference to type IpMultiPartyCall.

TpMultiPartyCallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Call Leg object

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallReference	IpMultiPartyCallRef	This element specifies the interface reference for the Multi-party call object.
CallSessionID	TpSessionID	This element specifies the call session ID.

TpMultiPartyCallIdentifierRef

Defines a Reference to type TpCallLegIdentifier.

TpMultiPartyCallIdentifierSet

Defines a Numbered Set of Data Elements of TpMultiPartyCallIdentifier.

TpMultiPartyCallIdentifierSetRef

Defines a Reference to type TpMultiPartyCallIdentifierSet.

TpCallAppInfo

Defines the Tagged Choice of Data Elements that specify application-related call information.

	Tag Element Type	
	TpCallAppInfoType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_APP_ALERTING_MECHANISM	TPCallAlertingMechanism	CallAppAlertingMechanism

P_CALL_APP_NETWORK_ACCESS_TYPE	TpCallNetworkAccessType	CallAppNetworkAccessType
P_CALL_APP_TELE_SERVICE	TpCallTeleService	CallAppTeleService
P_CALL_APP_BEARER_SERVICE	TpCallBearerService	CallAppBearerService
P_CALL_APP_PARTY_CATEGORY	TpCallPartyCategory	CallAppPartyCategory
P_CALL_APP_PRESENTATION_ADDRESS	TpAddress	CallAppPresentationAddress
P_CALL_APP_GENERIC_INFO	TpString	CallAppGenericInfo
P_CALL_APP_ADDITIONAL_ADDRESS	TpAddress	CallAppAdditionalAddress
P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS	TpAddress	CallAppOriginalDestinationAddress
P_CALL_APP_REDIRECTING_ADDRESS	TpAddress	CallAppRedirectingAddress

TpCallAppInfoType

Defines the type of call application-related specific information.

Name	Value	Description
P_CALL_APP_UNDEFINED	0	Undefined
P_CALL_APP_ALERTING_MECHANISM	1	The alerting mechanism or pattern to use
P_CALL_APP_NETWORK_ACCESS_TYPE	2	The network access type (e.g. ISDN)
P_CALL_APP_TELE_SERVICE	3	Indicates the tele-service (e.g. telephony)
P_CALL_APP_BEARER_SERVICE	4	Indicates the bearer service (e.g. 64kb/s unrestricted data).
P_CALL_APP_PARTY_CATEGORY	5	The category of the calling party
P_CALL_APP_PRESENTATION_ADDRESS	6	The address to be presented to other call parties
P_CALL_APP_GENERIC_INFO	7	Carries unspecified service-service information
P_CALL_APP_ADDITIONAL_ADDRESS	8	Indicates an additional address
P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS	9	Contains the original address specified by the originating user when launching the call.
P_CALL_APP_REDIRECTING_ADDRESS	10	Contains the address of the user from which the call is diverting.

TpCallEventRequest

Defines the Sequence of Data Elements that specify the criteria relating to call report requests.

Sequence Element Name	Sequence Element Type
CallEventType	TpCallEventType
AdditionalCallEventCriteria	TpAdditionalCallEventCriteria
CallMonitorMode	TpCallMonitorMode

TpCallEventRequestSet

Defines a Numbered Set of Data Elements of TpCallEventRequest.

TpCallEventType

Defines a specific call event report type.

Name	Value	Description
P_CALL_EVENT_UNDEFINED	0	Undefined
P_CALL_EVENT_CALL_ATTEMPT	1	A Call attempt takes place (e.g. Offhook event)
P_CALL_EVENT_ADDRESS_COLLECTED	2	The destination address has been collected
P_CALL_EVENT_ADDRESS_ANALYSED	3	The destination address has been analysed
P_CALL_EVENT_PROGRESS	4	Call routing progress event:an indication from the network that progress has been made in routing the call to the

		requested call party.
P_CALL_EVENT_ALERTING	5	Call is alerting at the call party
P_CALL_EVENT_ANSWER	6	Call answered at address
P_CALL_EVENT_RELEASE	7	A Call has been released or the call could not be routed
P_CALL_EVENT_REDIRECTED	8	Call redirected to new address: an indication from the network that the call has been redirected to a new address.
P_CALL_EVENT_SERVICE_CODE	9	Mid-call service code received

The table below defines the disarming rules for dynamic events. In case such an event occurs the table shows which events are disarmed (are not monitored anymore) and should be re-armed by eventReportReq() in case the application is still interested in these events.

Event Occured	Events Disarmed
P_CALL_EVENT_UNDEFINED	Not Applicable
P_CALL_EVENT_CALL_ATTEMPT	Not applicable, can only be armed as trigger
P_CALL_EVENT_ADDRESS_COLLECTED	P_CALL_EVENT_ADDRESS_COLLECTED
P_CALL_EVENT_ADDRESS_ANALYSED	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED
P_CALL_EVENT_PROGRESS	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED P_CALL_EVENT_PROGRESS
P_CALL_EVENT_ALERTING	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED P_CALL_EVENT_PROGRESS P_CALL_EVENT_ALERTING P_CALL_EVENT_RELEASE with criteria: • P_USER_NOT_AVAILABLE • P_BUSY • P_NOT_REACHABLE • P_ROUTING_FAILURE • P_CALL_RESTRICTED • P_UNAVAILABLE_RESOURCES
P_CALL_EVENT_ANSWER	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED P_CALL_EVENT_PROGRESS P_CALL_EVENT_ALERTING P_CALL_EVENT_RELEASE with criteria: • P_USER_NOT_AVAILABLE • P_BUSY • P_NOT_REACHABLE • P_ROUTING_FAILURE • P_CALL_RESTRICTED • P_UNAVAILABLE_RESOURCES • P_NO_ANSWER • P_PREMATURE_DISCONNECT P_CALL_EVENT_ANSWER
P_CALL_EVENT_RELEASE	All pending events are disarmed
P_CALL_EVENT_REDIRECTED	P_CALL_EVENT_REDIRECTED
P_CALL_EVENT_SERVICE_CODE	P_CALL_EVENT_SERVICE_CODE

TpAdditionalCallEventCriteria

Defines the Tagged Choice of Data Elements that specify specific criteria.

	Tag Element Type	
	TpCallEventType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_EVENT_UNDEFINED	NULL	Undefined
P_CALL_EVENT_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_ADDRESS_COLLECTED	TpInt32	MinAddressLength
P_CALL_EVENT_ADDRESS_ANALYSED	NULL	Undefined
P_CALL_EVENT_PROGRESS	NULL	Undefined
P_CALL_EVENT_ALERTING	NULL	Undefined
P_CALL_EVENT_ANSWER	NULL	Undefined
P_CALL_EVENT_RELEASE	TpCallReleaseCauseSet	ReleaseCauseSet
P_CALL_EVENT_REDIRECTED	NULL	Undefined
P_CALL_EVENT_SERVICE_CODE	TpCallServiceCode	ServiceCode

TpCallReleaseCauseSet

Defines a Numbered Set of Data Elements of TpCallReleaseCause.

TpCallEventInfo

Defines the Sequence of Data Elements that specify the event report specific information.

Sequence Element Name	Sequence Element Type
CallEventType	TpCallEventType
AdditionalCallEventInfo	TpAdditionalCallEventInfo
CallMonitorMode	TpCallMonitorMode
CallEventTime	TpDateAndTime

TpCallAdditionalEventInfo

Defines the Tagged Choice of Data Elements that specify additional call event information for certain types of events.

	Tag Element Type	
	TpCallEventType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_EVENT_UNDEFINED	NULL	Undefined
P_CALL_EVENT_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_ADDRESS_COLLECTED	TpAddress	CollectedAddress
P_CALL_EVENT_ADDRESS_ANALYSED	TpAddress	CalledAddress
P_CALL_EVENT_PROGRESS	NULL	Undefined
P_CALL_EVENT_ALERTING	NULL	Undefined
P_CALL_EVENT_ANSWER	NULL	Undefined
P_CALL_EVENT_RELEASE	TpCallReleaseCause	ReleaseCause
P_CALL_EVENT_REDIRECTED	TpAddress	ForwardAddress
P_CALL_EVENT_SERVICE_CODE	TpCallServiceCode	ServiceCode

TpCallNotificationRequest

Defines the Sequence of Data Elements that specify the criteria for an event notification

Sequence Element Name	Sequence Element Type	Description
CallNotificationScope	TpCallNoficationScope	Defines the scope of the nofication request.
CallEventsRequested	TpCallEventRequestSet	Defines the events which are requested

TpCallNotificationScope

Defines a the sequence of Data elements that specify the scope of a notification request.

Of the addresses only the Plan and the AddrString are used for the purpose of matching the notifications against the criteria.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddressRange	Defines the destination address or address range for which the notification is requested.

OriginatingAddress	TpAddressRange	Defines the origination address or address range for which the notification is requested.
NotificationCallType	TpNotificationCallType	Defines wheter the notification is requested for a originating or terminating call.

TpNotificationCallType

Defines the type of call for which the notification is requested or reported.

Name	Value	Description
P_ORIGINATING	1	Indicates that the notification is related to the originating user in the call.
P_TERMINATING	2	Indicates that the notification is related to the terminating user in the call.

TpCallNotificationInfo

Defines the Sequence of Data Elements that specify the information returned to the application in a Call notification report.

Sequence Element Name	Sequence Element Type	Description
CallNotificationReportScope	TpCallNotificationReportScope	Defines the scope of the notification report.
CallAppInfo	TpCallAppInfoSet	Contains additonal call info.
CallEventInfo	TpCallEventInfo	Contains the event which is reported.

TpCallNotificationReportScope

Defines the Sequence of Data Elements that specify the scope for which a notification report was sent.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddress	Contains the destination address of the call.
OriginatingAddress	TpAddress	Contains the origination address of the call
NotificationCallType	TpNotificationCallType	Indicates if the notification was reported for an originating or terminating call.

TpNotificationRequested

Defines the Sequence of Data Elements that specify the criteria relating to event requests.

Sequence Element Name	Sequence Element Type
AppCallNotificationRequest	TpCallNotificationRequest
AssignmentID	TpInt32

TpNotificationsRequestedSet

Defines a numbered Set of Data Elements of TpNotificationRequested

TpNotificationsRequestedSetRef

Defines a reference to the type TpNotificationsRequestSet

TpCallReleaseCause

Defines the reason for which a call is released

Name	Value	Description
P_UNDEFINED	0	The reason of release isn't known, because no info was received from the network.
P_USER_NOT_AVAILBLE	1	The user isn't available in the network. This means that the number isn't allocated or that the user isn't registered.
P_BUSY	2	The user is busy.
P_NO_ANSWER	3	No answer was received
P_NOT_REACHABLE	4	The user terminal isn't reachable
P_ROUTING_FAILURE	5	A routing failure occurred. For example an invalid address was received
P_PREMATURE_DISCONNECT	6	The user disconnected the call during setup phase.
P_DISCONNECTED	7	Call disconnect by the end user.
P_CALL_RESTRICTED	8	The call was subject of restrictions
P_UNAVAILABLE_RESOURCE	9	No resources where available to establisch the call.
P_GENERAL_FAILURE	10	A general network failure occurred.

TpCallLegIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Call Leg object

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallLegReference	IpCallLegRef	This element specifies the interface reference for the callLeg object.
CallLegSessionID	TpSessionID	This element specifies the callLeg session ID.

TpCallLegIdentifierRef

Defines a Reference to type TpCallLegIdentifier.

TpCallLegIdentifierSet

Defines a Numbered Set of Data Elements of TpCallLegIdentifier.

TpCallLegIdentifierSetRef

Defines a Reference to type TpCallLegIdentifierSet.

TpCallLegAttachMechanism

Defines how a CallLeg should be attached to the call.

Name	Value	Description
P_CALLLEG_ATTACH_IMPLICITLY	0	CallLeg should be attached implicitly to the call.
P_CALLLEG_ATTACH_EXPLICITLY	1	CallLeg should be attached explicitly to the call by using the attachMedia() operation. This allows e.g. the application to do first user interaction to the party before he / she is placed in the call.

TpCallLegConnectionProperties

Defines the Sequence of Data Elements that specify the connection properties of the Call Leg object

Sequence Element Name	Sequence Element Type	Sequence Element Description
AttachMechanism	TpCallLegAttachMechanism	Defines how a CallLeg should be attached to the call.

TpCallLegInfoReport

Defines the Sequence of Data Elements that specify the call leg information requested.

Sequence Element Name	Sequence Element Type	description
CallLegInfoType	TpCallLegInfoType	The type of the call leg.
CallLegStartTime	TpDateAndTime	The time and date when the call leg was started (i.e., the leg was routed).
CallLegConnectedToResourceTime	TpDateAndTime	The date and time when the call leg was connected to the resource. If no resource was connected the time is set to an empty string. Either this element is valid or the CallConnectedToAddressTime is valid, depending on whether the report is sent as a result of user interaction.
CallLegConnectedToAddressTime	TpDateAndTime	The date and time when the call leg was connected to the destination (i.e., when the destination answered the call). If the destination did not answer, the time is set to an empty string. Either this element is valid or the CallConnectedToResourceTime is valid, depending on whether the report is sent as a result of user interaction.
CallLegEndTime	TpDateAndTime	The date and time when the call leg was released.

ConnectedAddress	TpAddress	The address of the party associated with the leg. If during the call the connected address was received from the party then this is returned, otherwise the destination address (for legs connected to a destination) or the originating address (for legs connected to the origination) is returned.
CallLegReleaseCause	TpCallReleaseCause	The cause of the termination. May be present with P_CALL_LEG_INFO_RELEASE_CAUSE was specified.
CallAppInfo	TpCallAppInfoSet	Additional information for the leg. May be present with P_CALL_LEG_INFO_APPINFO was specified.

TpCallLegInfoType

Defines the type of call leg information requested and reported. The values may be combined by a logical 'OR' function.

Name	Value	Description
P_CALL_LEG_INFO_UNDEFINED	00h	Undefined
P_CALL_LEG_INFO_TIMES	01h	Relevant call times
P_CALL_LEG_INFO_RELEASE_CAUSE	02h	Call leg release cause
P_CALL_LEG_INFO_ADDRESS	04h	Call leg connected address
P_CALL_LEG_INFO_APPINFO	08h	Call leg application related information

Annex A (normative): OMG IDL Description of Call Control SCF

The OMG IDL representation of this interface specification is contained in a text file (contained in archive 2919804IDL.ZIP) which accompanies the present document.

Annex B (informative): Differences between this draft and 3GPP 29.198 R99

The following is a list of the differences between this draft and 3GPP 29.198 R99, for those interfaces which are common to both documents. Any new interfaces with respect to Release 99 are not listed.

B.1 Interface IpCallControlManager

enableCallNotification (appCallControlManager~~Interface~~ : in IpAppCallControlManagerRef, eventCriteria : in TpCallEventCriteria, assignmentID : out TpAssignmentIDRef) : TpResult

createCall (appCall : in IpAppCallRef, callReference : out TpCallIdentifierRef) : TpResult

setCallLoadControl (duration : in TpDuration, mechanism : in TpCallLoadControlMechanism, treatment : in TpCallTreatment, addressRange : in TpAddressRange, assignmentID : out TpAssignmentIDRef) : TpResult

B.2 Interface IpAppCallControlManager

callEventNotify (callReference : in TpCallIdentifier, eventInfo : in TpCallEventInfo, assignmentID : in TpAssignmentID, appCall~~Interface~~ : out IpAppCallRefRef) : TpResult

callOverloadEncountered (assignmentID : in TpAssignmentID) : TpResult

callOverloadCeased (assignmentID : in TpAssignmentID) : TpResult

B.3 Interface IpCall

getMoreDialledDigitsReq (callSessionID : in TpSessionID, length : in TpInt32) : TpResult

B.4 Interface IpAppCall

getMoreDialledDigitsRes (callSessionID : in TpSessionID, digits : in TpString) : TpResult

getMoreDialledDigitsErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for Information

3GPP TS 29.198-5 V1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface;
Part 5: Generic User Interaction
(Release 4)

Keywords
API, OSA, IDL, UI, User Interaction

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

This document is part of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA). The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA API's. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

This document specifies the User Interaction Service Capability Feature (SCF) aspects of the interface. All aspects of the User Interaction SCF are defined here, these being:

- Sequence Diagrams
- Class Diagrams
- Interface specification plus detailed method descriptions
- State Transition diagrams
- Data definitions
- IDL Description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modelling Language (UML).

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

[1] 3GPP TS 29.198-1 "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in TS 29.198-1 [1] apply.

3.2 Symbols

For the purposes of the present document, the symbols in TS 29.198-1 [1] apply.

3.3 Abbreviations

For the purposes of the present document, the abbreviations in TS 29.198-1 [1] apply.

4 Generic and Call User Interaction SCF

The Generic User Interaction service capability feature is used by applications to interact with end users. It consists of two interfaces:

1) User Interaction Manager, containing management functions for User Interaction related issues;

2) Generic User Interaction, containing methods to interact with an end-user.

The Generic User Interaction service capability feature is described in terms of the methods in the Generic User Interaction interfaces.

The following table gives an overview of the Generic User Interaction methods and to which interfaces these methods belong.

Table 1: Overview of Generic User Interaction interfaces and their methods

User Interaction Manager	Generic User Interaction
createUI	sendInfoReq
createUICall	sendInfoRes
createNotification	sendInfoErr
destroyUINotification	sendInfoAndCollectReq
reportNotification	sendInfoAndCollectRes
userInteractionAborted	sendInfoAndCollectErr
userInteractionNotificationInterrupted	release
userInteractionNotificationContinued	UserInteractionFaultDetected
changeNotification	
getNotification	

The following table gives an overview of the Call User Interaction methods and to which interfaces these methods belong.

Table 2: Overview of Call User Interaction interfaces and their methods

User Interaction Manager	Call User Interaction
As defined for the Generic User Interaction SCF	Inherits from Generic User Interaction and adds:
	recordMessageReq
	recordMessageRes
	recordMessageErr
	deleteMessageReq
	deleteMessageRes
	deleteMessageErr
	abortActionReq
	abortActionRes
	abortActionErr

The IpUI Interface provides functions to send information to, or gather information from the user, i.e. this interface allows applications to send SMS and USSD messages. An application can use this interface independently of other SCFs. The IpUICall Interface provides functions to send information to, or gather information from the user (or call party) attached to a call.

The following sections describe each aspect of the Generic User Interaction Service Capability Feature (SCF).

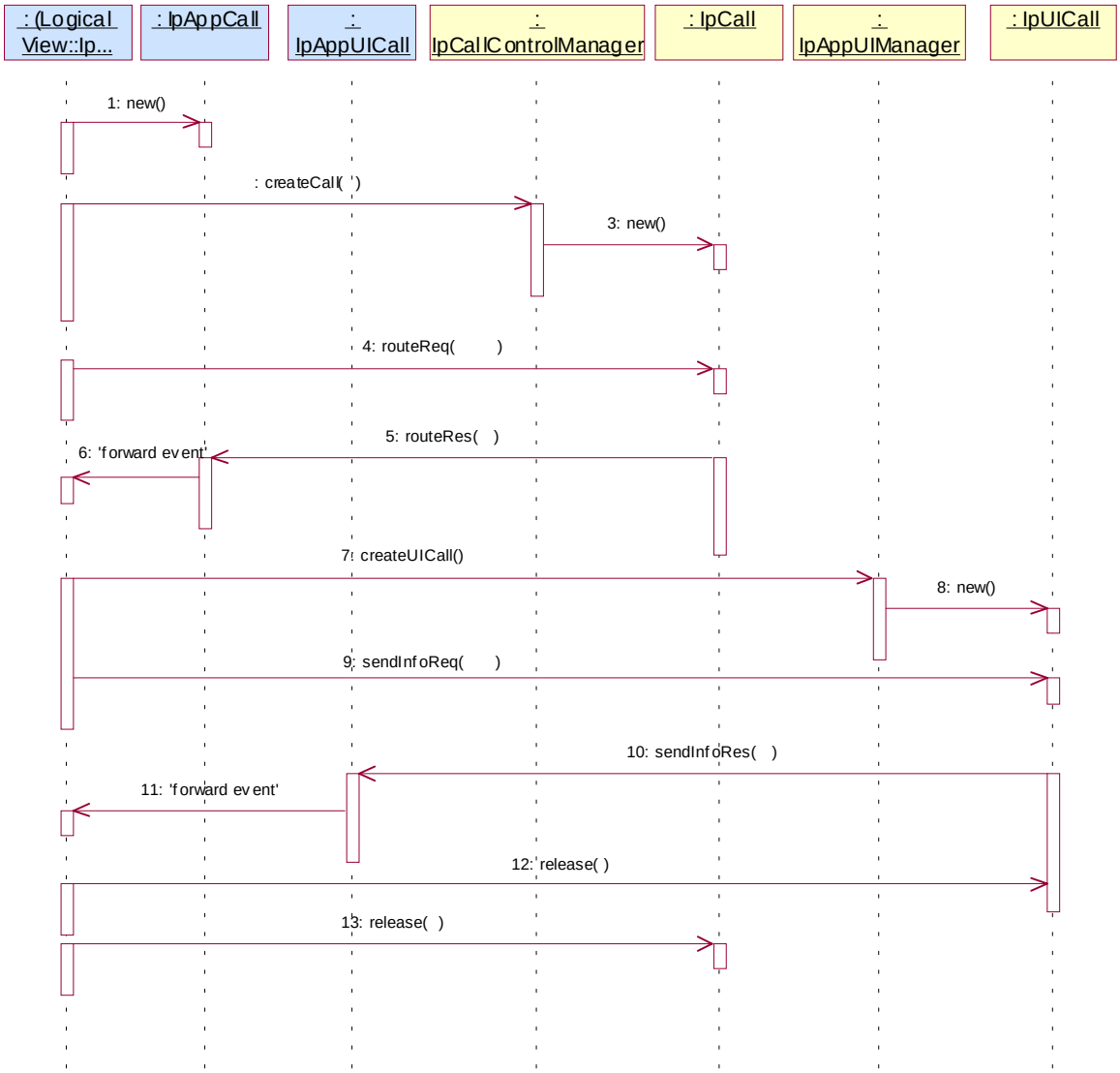
The order is as follows:

- the Sequence diagrams give the reader a practical idea of how each of the service capability feature is implemented;
- the Class relationships section show how each of the interfaces applicable to the SCF, relate to one another;
- the Interface specification section describes in detail each of the interfaces shown within the Class diagram part. This section also includes Call User interation;
- the State Transition Diagrams (STD) show the progression of internal processes either in the application, or Gateway;
- the Data definitions section show a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.

5 Sequence Diagrams

5.1 Alarm Call

The following sequence diagram shows a 'reminder message', in the form of an alarm, being delivered to a customer as a result of a trigger from an application. Typically, the application would be set to trigger at a certain time, however, the application could also trigger on events.



1: This message is used to create an object implementing the IpAppCall interface.

2: This message requests the object implementing the IpCallControlManager interface to create an object implementing the IpCall interface.

3: Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met it is created.

4: This message instructs the object implementing the IpCall interface to route the call to the customer destined to receive the 'reminder message'

5: This message passes the result of the call being answered to its callback object.

6: This message is used to forward the previous message to the IpAppLogic.

7: The application requests a new UICall object that is associated with the call object.

8: Assuming all criteria are met, a new UICall object is created by the service.

9: This message instructs the object implementing the IpUICall interface to send the alarm to the customer's call.

10: When the announcement ends this is reported to the call back interface.

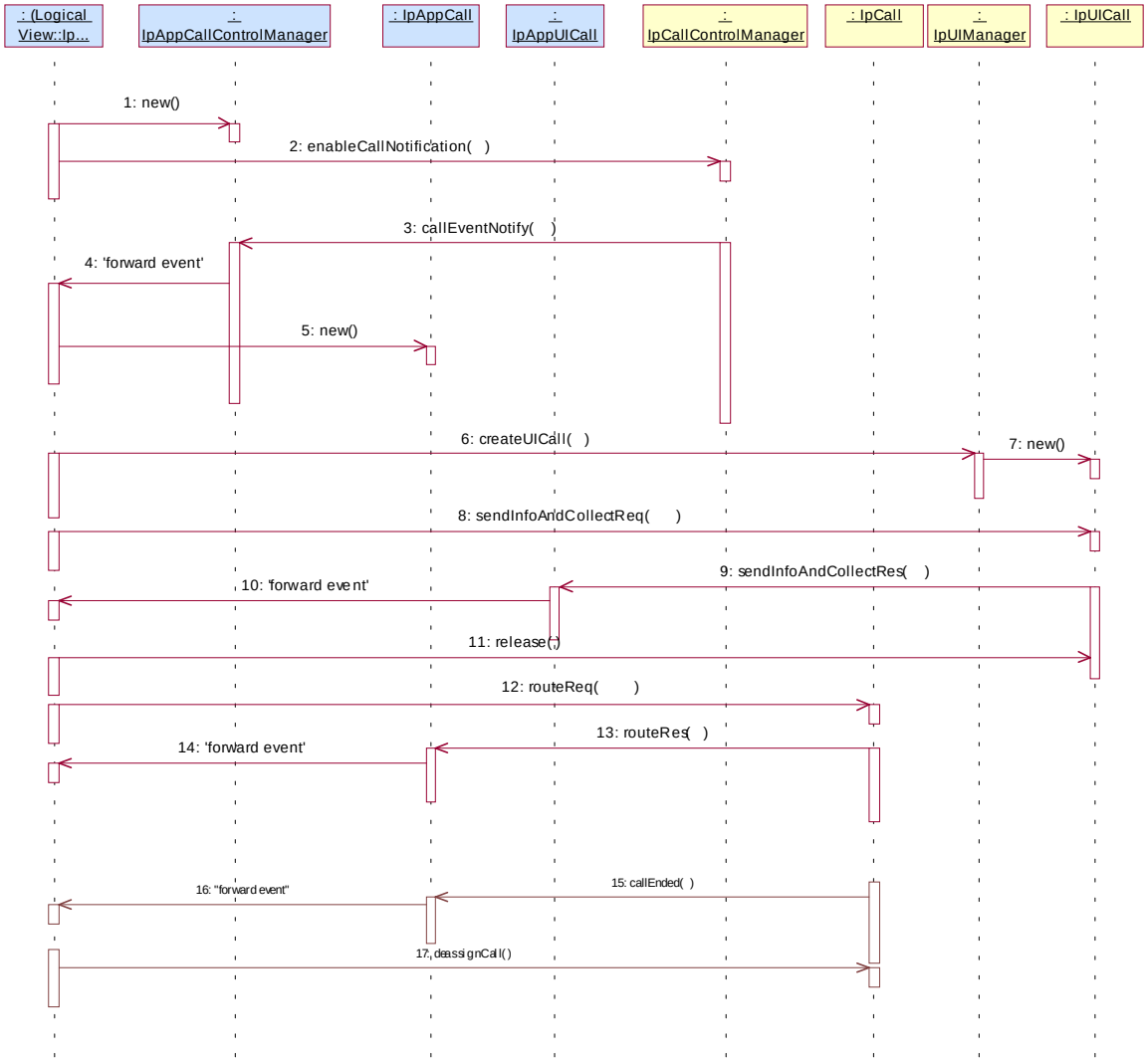
11: The event is forwarded to the application logic.

12: The application releases the UICall object, since no further announcements are required. Alternatively, the application could have indicated P_FINAL_REQUEST in the sendInfoReq in which case the UICall object would have been implicitly released after the announcement was played.

13: The application releases the call and all associated parties.

5.2 Call Barring 1

The following sequence diagram shows a call barring service, initiated as a result of a prearranged event being received by the framework. Before the call is routed to the destination number, the calling party is asked for a PIN code. The code is accepted and the call is routed to the original called party.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria set, arrives, a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward the previous message to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.

6: This message is used to create a new UICall object. The reference to the call object is given when creating the UICall.

7: Provided all the criteria are fulfilled, a new UICall object is created.

8: The call barring service dialogue is invoked.

9: The result of the dialogue, which in this case is the PIN code, is returned to its callback object.

10: This message is used to forward the previous message to the IpAppLogic.

11: This message releases the UICall object.

12: Assuming the correct PIN is entered, the call is forward routed to the destination party.

13: This message passes the result of the call being answered to its callback object.

14: This message is used to forward the previous message to the IpAppLogic

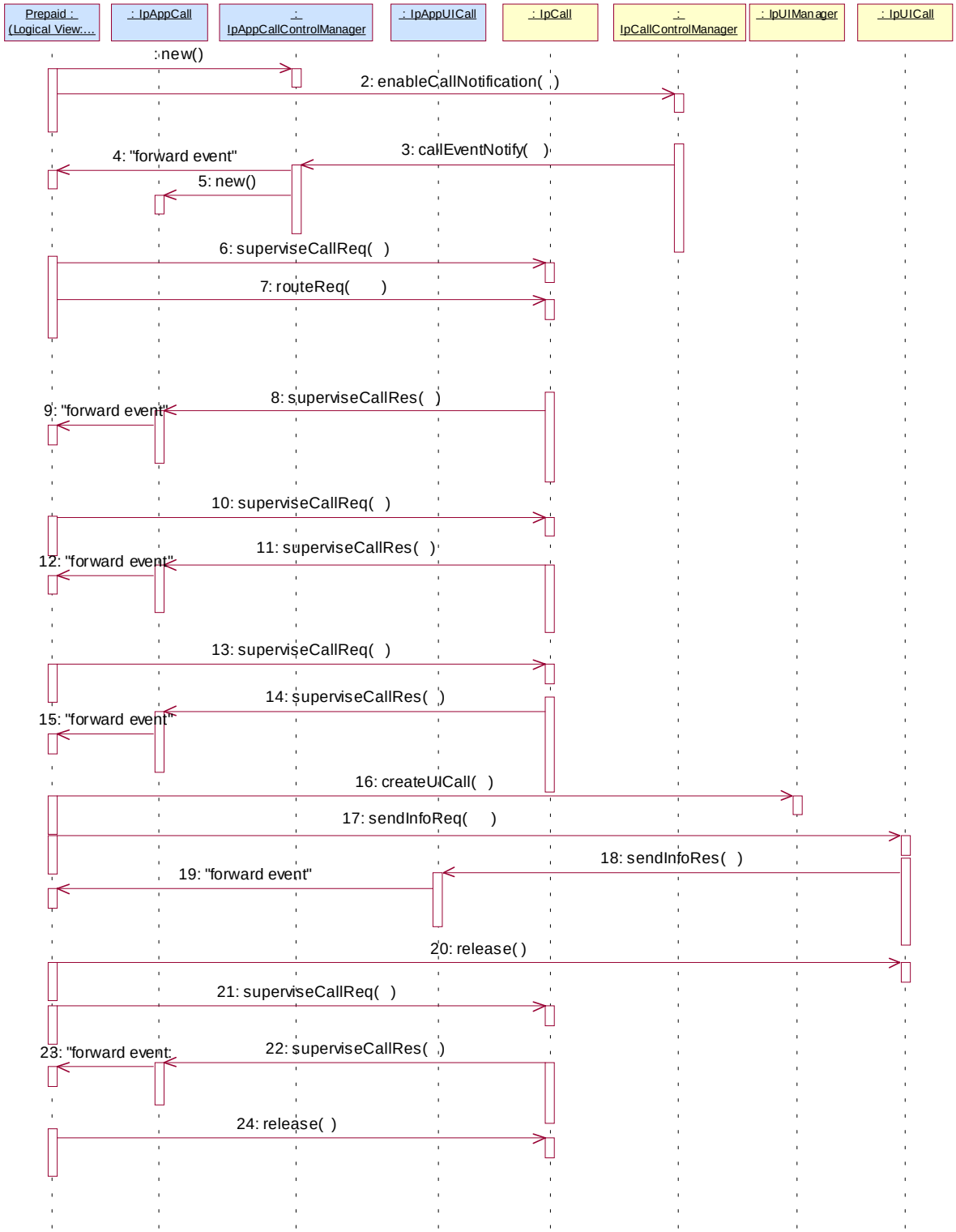
15: When the call is terminated in the network, the application will receive a notification. This notification will always be received when the call is terminated by the network in a normal way, the application does not have to request this event explicitly.

16: The event is forwarded to the application.

17: The application must free the call related resources in the gateway by calling deassignCall.

5.3 Prepaid

This sequence shows a Pre-paid application. The subscriber is using a pre-paid card or credit card to pay for the call. The application each time allows a certain timeslice for the call. After the timeslice, a new timeslice can be started or the application can terminate the call. In the following sequence the end-user will received an announcement before his final timeslice.



1: This message is used by the application to create an object implementing the IpAppGenericCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a pre-paid service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: The incoming call triggers the Pre-Paid Application (PPA).

4: The message is forwarded to the application.

5: A new object on the application side for the Generic Call object is created

6: The Pre-Paid Application (PPA) requests to supervise the call. The application will be informed after the period indicated in the message. This period is related to the credits left on the account of the pre-paid subscriber.

7: Before continuation of the call, PPA sends all charging information, a possible tariff switch time and the call duration supervision period, towards the GW which forwards it to the network.

8: At the end of each supervision period the application is informed and a new period is started.

9: The message is forwarded to the application.

10: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

11: At the end of each supervision period the application is informed and a new period is started.

12: The message is forwarded to the application.

13: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

14: When the user is almost out of credit an announcement is played to inform about this. The announcement is played only to the leg of the A-party, the B-party will not hear the announcement.

15: The message is forwarded to the application.

16: A new UICall object is created and associated with the controlling leg.

17: An announcement is played to the controlling leg informing the user about the near-expiration of his credit limit. The B-subscriber will not hear the announcement.

18: When the announcement is completed the applicaiton is informed.

19: The message is forwarded to the application.

20: The application releases the UICall object.

21: The user does not terminate so the application terminates the call after the next supervision period.

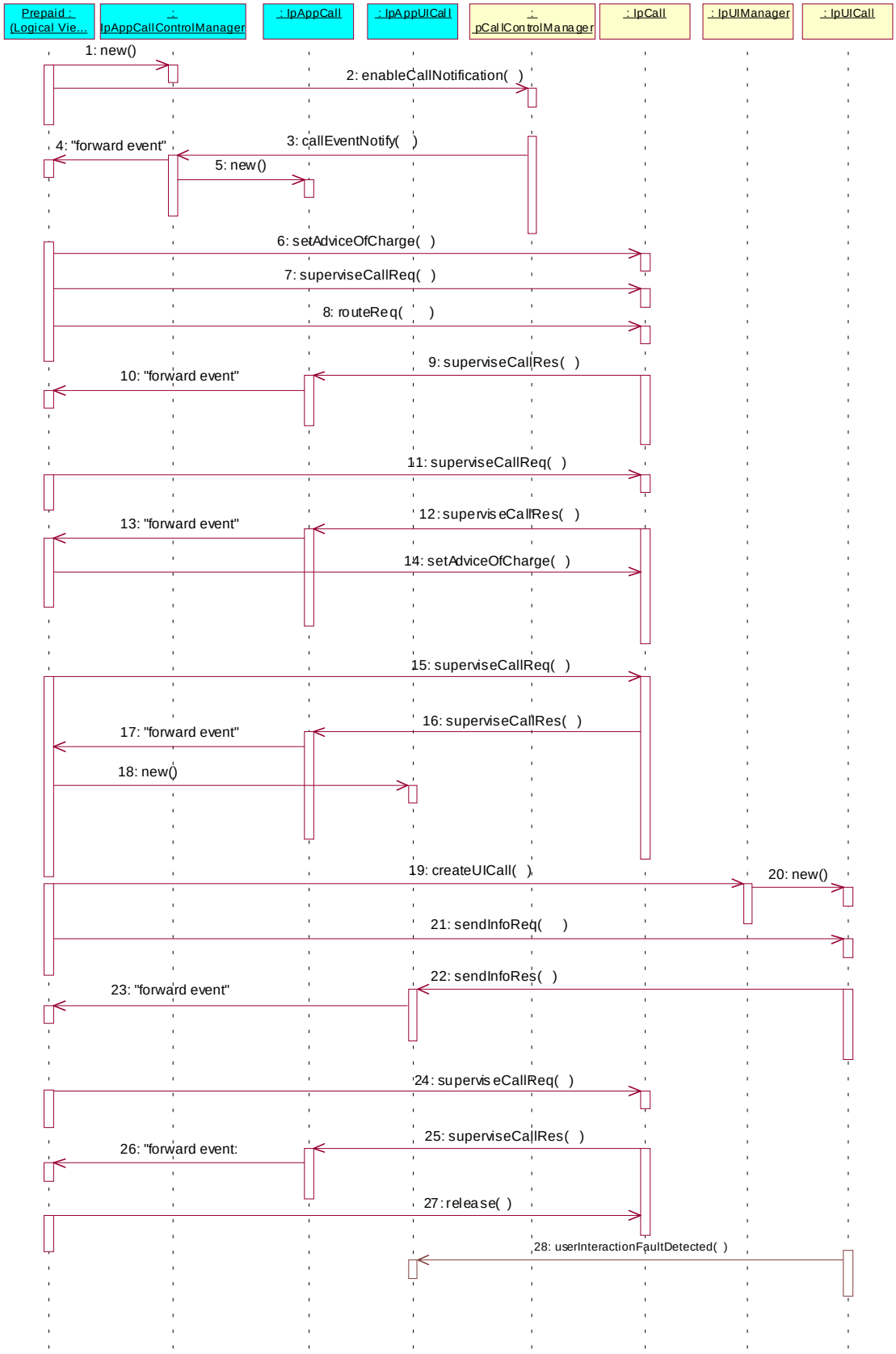
22: The supervision period ends

23: The event is forwarded to the logic.

24: The application terminates the call. Since the user interaction is already explicitly terminated no userInteractionFaultDetected is sent to the application.

5.4 Pre-Paid with Advice of Charge (AoC)

This sequence shows a Pre-paid application that uses the Advice of Charge feature. The application will send the charging information before the actual call setup and when during the call the charging changes new information is sent in order to update the end-user. Note: the Advice of Charge feature requires an application in the end-user terminal to display the charges for the call, depending on the information received from the application.


Prepaid : (Logical Vie...
: IpAppCallControlManager
: IpAppCall
: IpAppUICall
: pCallControlManager
: IpCall
: IpUIManager
: IpUICall
1: new()
2: enableCallNotification( )
3: callEventNotify( )
4: "forward event"
5: new()
6: setAdviceOfCharge( )
7: superviseCallReq( )
8: routeReq( )
9: superviseCallRes( )
10: "forward event"
11: superviseCallReq( )
12: superviseCallRes( )
13: "forward event"
14: setAdviceOfCharge( )
15: superviseCallReq( )
16: superviseCallRes( )
17: "forward event"
18: new()
19: createUICall( )
20: new()
21: sendInfoReq( )
22: sendInfoRes( )
23: "forward event"
24: superviseCallReq( )
25: superviseCallRes( )
26: "forward event:
27: release( )
28: userInteractionFaultDetected( )

1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a pre-paid service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: The incoming call triggers the Pre-Paid Application (PPA).

4: The message is forwarded to the application.

5: A new object on the application side for the Call object is created

6: The Pre-Paid Application (PPA) sends the AoC information (e.g the tariff switch time). (it shall be noted the PPA contains ALL the tariff information and knows how to charge the user).

During this call sequence 2 tariff changes take place. The call starts with tariff 1, and at the tariff switch time (e.g., 18:00 hours) switches to tariff 2. The application is not informed about this (but the end-user is!)

7: The Pre-Paid Application (PPA) requests to supervise the call. The application will be informed after the period indicated in the message. This period is related to the credits left on the account of the pre-paid subscriber.

8: The application requests to route the call to the destination address.

9: At the end of each supervision period the application is informed and a new period is started.

10: The message is forwarded to the application.

11: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

12: At the end of each supervision period the application is informed and a new period is started.

13: The message is forwarded to the application.

14: Before the next tariff switch (e.g., 19:00 hours) the application sends a new AOC with the tarif switch time. Again, at the tariff switch time,the network will send AoC information to the end-user.

15: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

16: When the user is almost out of credit an announcement is played to inform about this (19-21). The announcement is played only to the leg of the A-party, the B-party will not hear the announcement.

17: The message is forwarded to the application.

18: The application creates a new call back interface for the User interaction messages.

19: A new UI Call object that will handle playing of the announcement needs to be created

20: The Gateway creates a new UI call object that will handle playing of the announcement.

21: With this message the announcement is played to the calling party.

22: The user indicates that the call should continue.

23: The message is forwarded to the application.

24: The user does not terminate so the application terminates the call after the next supervision period.

25: The user is out of credit and the application is informed.

26: The message is forwarded to the application.

27: With this message the application requests to release the call.

28: Terminating the call which has still a UICall object associated will result in a userInteractionFaultDetected. The UICall object is terminated in the gateway and no further communication is possible between the UICall and the application.

6 Class Diagrams

The application generic user interaction service package consists of one IpAppUIManager interface, zero or more IpAppUI interfaces and zero or more IpAppUICall interfaces.

The generic user interaction service package consists of one IpUIManager interface, zero or more IpUI interfaces and zero or more IpUICall interfaces.

The class diagram in the following figure shows the interfaces that make up the application generic user interaction service package and the generic user interaction service package. Communication between these packages is done via the <<uses>> relationships.

The IpUICall implements call related user interaction and it inherits from the non call related IpUI interface. The same holds for the corresponding application interfaces.

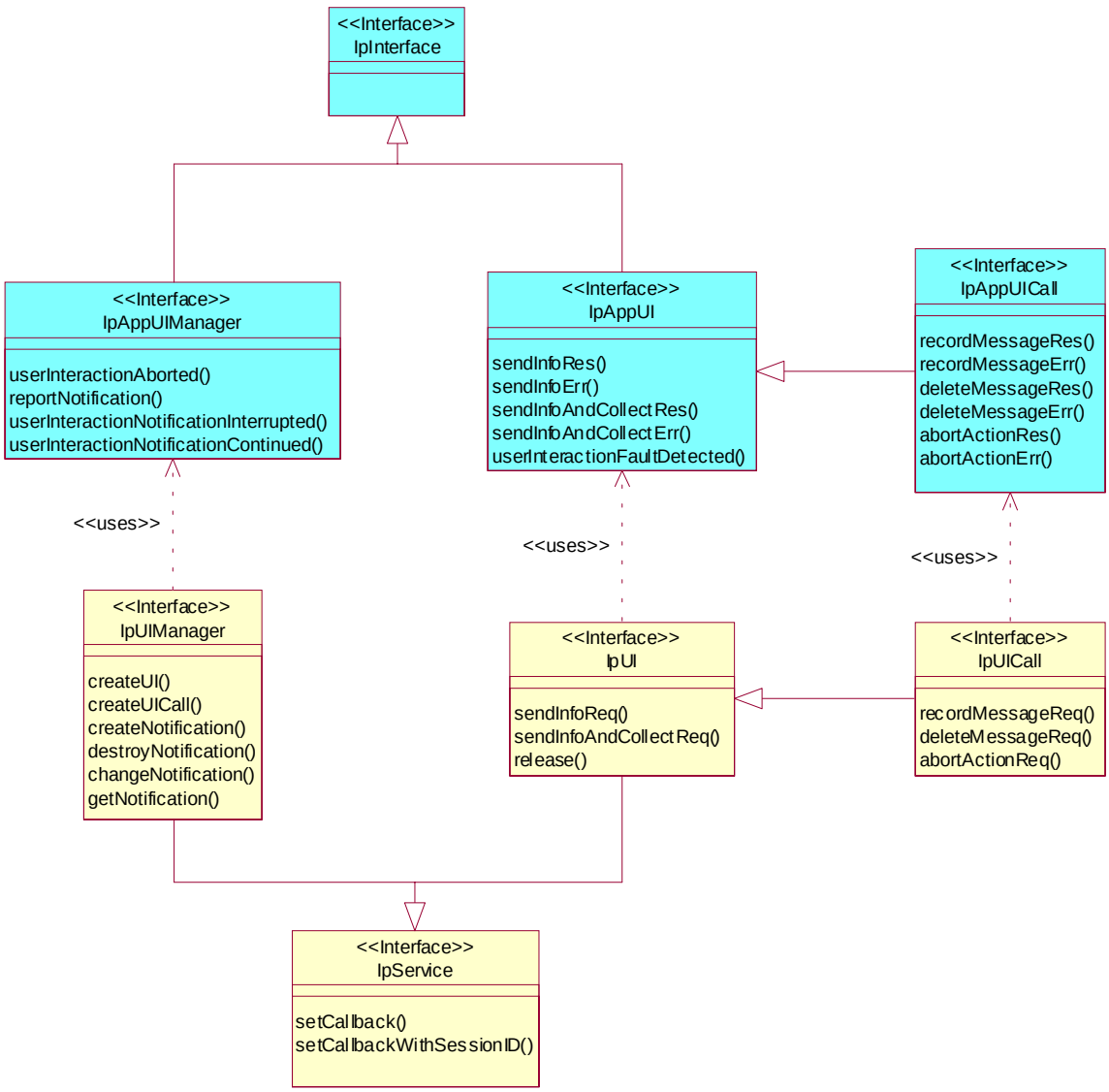


Figure: Generic User Interaction Package Overview

7 The Service Interface Specifications

7.1 Interface Specification Format

This section defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

7.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for enterprise-based client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>

7.1.2 Method descriptions

Each method (API method “call”) is described. All methods in the API return a value of type `TpResult`, indicating, amongst other things, if the method invocation was sucessfully executed or not.

Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a '`Req`' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a '`Res`' or '`Err`' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name> or IpSvc<name>` interfaces to provide the callback mechanism.

7.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.

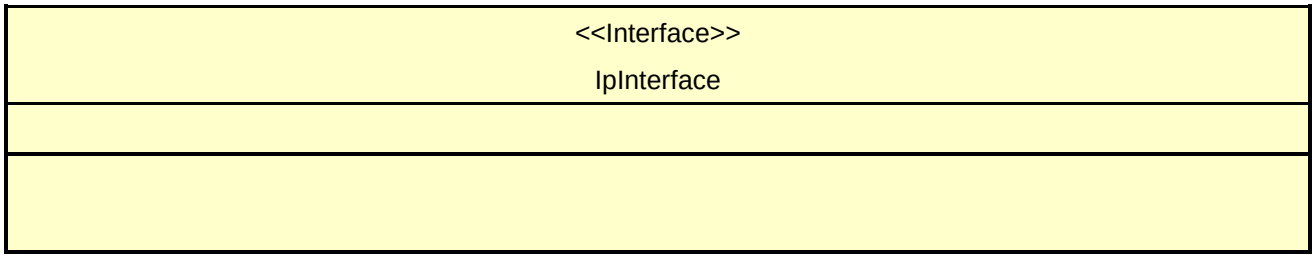
7.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

7.2 Base Interface

7.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.



7.3 Service Interfaces

7.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

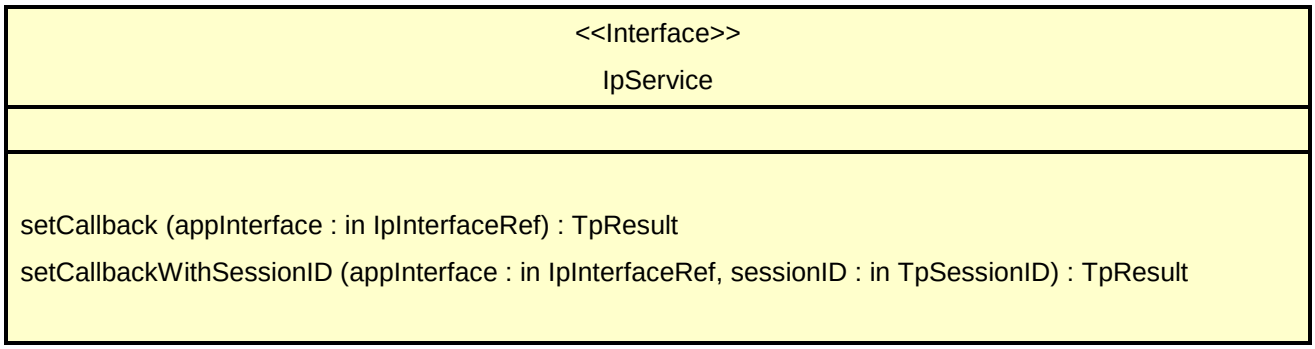
The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

7.4 Generic Service Interface

7.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.



Method
`setCallback()`

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application.

Parameters

`appInterface : in IpInterfaceRef`

Specifies a reference to the application interface, which is used for callbacks

Raises

`TpGeneralException`

Method
`setCallbackWithSessionID()`

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

sessionID : in TpSessionID

Specifies the session for which the service can invoke the application's callback interface.

Raises

TpGeneralException

8 Generic User Interaction Interface Classes

The Generic User Interaction Service interface (GUIS) is used by applications to interact with end users. The GUIS is represented by the IpUIManager, IpUI and IpUICall interfaces that interface to services provided by the network. To handle responses and reports, the developer must implement IpAppUIManager and IpAppUI interfaces to provide the callback mechanism.

8.1 Interface Class IpUIManager

Inherits from: IpService.

This interface is the 'service manager' interface for the Generic User Interaction Service and provides the management functions to the Generic User Interaction Service.

<<Interface>>

IpUIManager

createUI (appUI : in IpAppUIRef, userAddress : in TpAddress, userInteraction : out TpUIIdentifierRef) : TpResult

createUICall (appUI : in IpAppUICallRef, uiTargetObject : in TpUITargetObject, userInteraction : out TpUICallIdentifierRef) : TpResult

createNotification (appUIManager : in IpAppUIManagerRef, eventCriteria : in TpUIEventCriteria, assignmentID : out TpAssignmentIDRef) : TpResult

destroyNotification (assignmentID : in TpAssignmentID) : TpResult

changeNotification (assignmentID : in TpAssignmentID, evenCriteria : in TpUIEventCriteria) : TpResult

getNotification (eventCriteria : out TpUIEventCriteriaResultSetRef) : TpResult

Method

createUI()

This method is used to create a new user interaction object for non-call related purposes

Parameters

appUI : in IpAppUIRef

Specifies the application interface for callbacks from the user interaction created.

userAddress : in TpAddress

Indicates the end-user with whom to interact.

userInteraction : out TpUIIdentifierRef

Specifies the interface and sessionID of the user interaction created.

Raises

TpGUISException,TpGeneralException

Method

createUICall()

This method is used to create a new user interaction object for call related purposes.

The user interaction can take place to the specified party or to all parties in a call. Note that for certain implementation user interaction can only be performed towards the controlling call party, which shall be the only party in the call.

Parameters

appUI : in IpAppUICallRef

Specifies the application interface for callbacks from the user interaction created.

uiTargetObject : in TpUITargetObject

Specifies the object on which to perform the user interaction. This can either be a Call, Multi-party Call or call leg object.

userInteraction : out TpUICallIdentifierRef

Specifies the interface and sessionID of the user interaction created.

Raises

TpGUISException,TpGeneralException

Method

createNotification()

This method is used by the application to install specified notification criteria, for which the reporting is implicitly activated. If some application already requested notifications with criteria that overlap the specified criteria, the request is refused with P_GUIS_INVALID_CRITERIA.

The criteria are said to overlap if both originating and terminating ranges overlap and the same number plan is used and the same servicecode is used.

If the same application requests two notifications with exactly the same criteria but different callback references, the second callback will be treated as an additional callback. This means that the callback will only be used in case when the first callback specified by the application is unable to handle the reportNotification (e.g., due to overload or failure).

Parameters

appUIManager : in IpAppUIManagerRef

If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the setCallback() method.

eventCriteria : in TpUIEventCriteria

Specifies the event specific criteria used by the application to define the event required, like user address and service code.

assignmentID : out TpAssignmentIDRef

Specifies the ID assigned by the generic user interaction manager interface for this newly installed notification criteria.

Raises

TpGUISException,TpGeneralException

Method

destroyNotification()

This method is used by the application to destroy previously installed notification criteria via the createNotification method.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID given by the generic user interaction manager interface when the previous createNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will return the error code P_INVALID_ASSIGNMENTID.

Raises

TpGUISException,TpGeneralException

Method

changeNotification()

This method is used by the application to change the event criteria introduced with createNotification method. Any stored notification request associated with the specified assignementID will be replaced with the specified events requested.

Parameters

assignmentID : in TpAssignmentID

Specifies the ID assigned by the manager interface for the event notification.

evenCriteria : in TpUIEventCriteria

Specifies the new set of event criteria used by the application to define the event required. Only events that meet these criteria are reported.

Raises

TpGUISException,TpGeneralException

Method

getNotification()

This method is used by the application to query the event criteria set with createNotification or changeNotification.

Parameters

eventCriteria : out TpUIEventCriteriaResultSetRef

Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

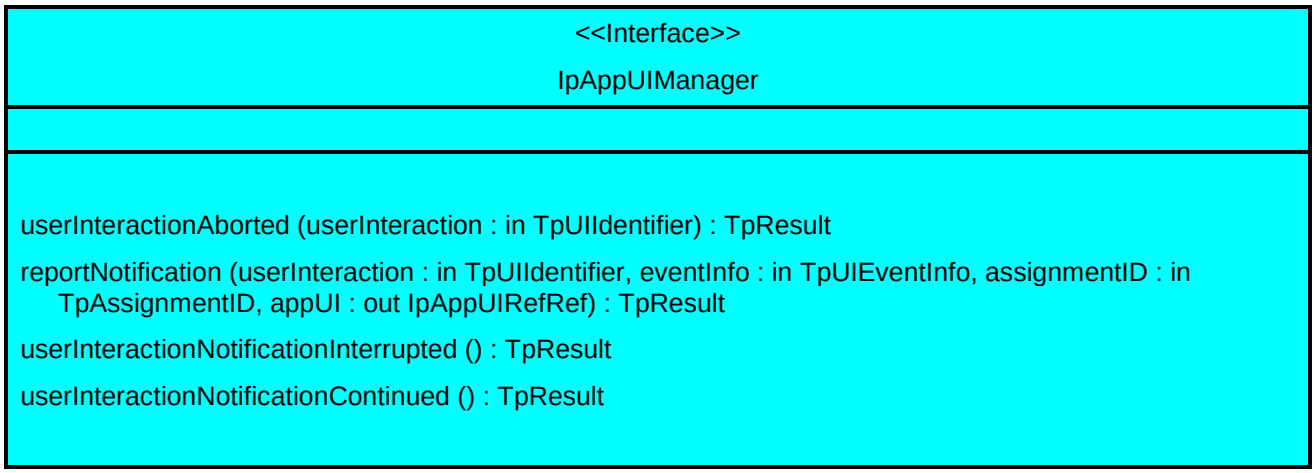
Raises

TpGUISException,TpGeneralException

8.2 Interface Class IpAppUIManager

Inherits from: IpInterface.

The Generic User Interaction Service manager application interface provides the application callback functions to the Generic User Interaction Service.



Method

userInteractionAborted()

This method indicates to the application that the User Interaction service instance has terminated or closed abnormally. No further communication will be possible between the User Interaction service instance and application.

Parameters

userInteraction : in TpUIIdentifier

Specifies the interface and sessionID of the user interaction service that has terminated.

Raises

TpGUISException,TpGeneralException

Method

reportNotification()

This method notifies the application of an occured network event which matches the criteria installed by the createNotification method.

Parameters

userInteraction : in TpUIIdentifier

Specifies the reference to the interface and the sessionID to which the notification relates.

eventInfo : in TpUIEventInfo

Specifies data associated with this event.

assignmentID : in TpAssignmentID

Specifies the assignment id which was returned by the createNotification() method. The application can use assignment id to associate events with event specific criteria and to act accordingly.

appUI : out IpAppUIRefRef

Specifies a reference to the application interface, which implements the callback interface for the new user interaction.

Raises

TpGUISException,TpGeneralException

Method

userInteractionNotificationInterrupted()

This method indicates to the application that all event notifications have been temporary interrupted (for example, due to faults detected). Note that more permanent failures are reported via the Framework (integrity management).

Parameters

No Parameters were identified for this method

Raises

TpGUISException,TpGeneralException

Method
userInteractionNotificationContinued()

This method indicates to the application that event notifications will again be possible.

Parameters
No Parameters were identified for this method

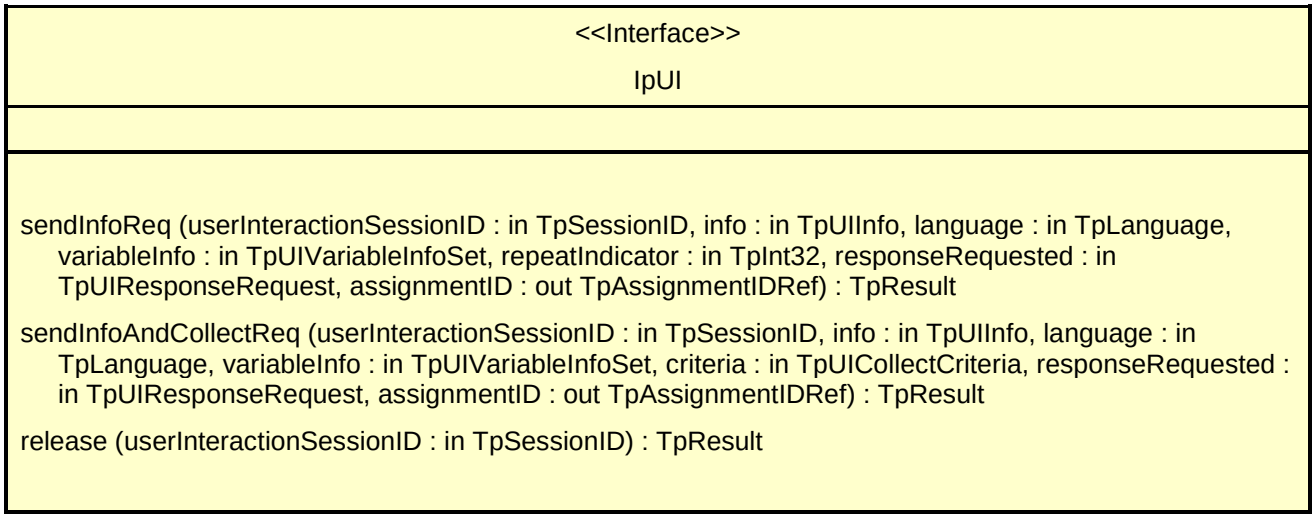
Raises

TpGUISException,TpGeneralException

8.3 Interface Class IpUI

Inherits from: IpService.

The User Interaction Service Interface provides functions to send information to, or gather information from the user. An application can use the User Interaction Service Interface independently of other services.



Method
sendInfoReq()

This asynchronous method plays an announcement or sends other information to the user.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

info : in TpUIInfo

Specifies the information to send to the user. This information can be:

- an infoID, identifying pre-defined information to be send (announcement and/or text);

- a string, defining the text to be sent;

- a URL , identifying pre-defined information or data to be sent to or downloaded into the terminal.

language : in TpLanguage

Specifies the Language of the information to be send to the user.

variableInfo : in TpUIVariableInfoSet

Defines the variable part of the information to send to the user.

repeatIndicator : in TpInt32

Defines how many times the information shall be sent to the end-user. A value of zero (0) indicates that the announcement shall be repeated until the call or call leg is released or an abortActionReq() is sent.

responseRequested : in TpUIResponseRequest

Specifies if a response is required from the call user interaction service, and any action the service should take.

assignmentID : out TpAssignmentIDRef

Specifies the ID assigned by the generic user interaction interface for a user interaction request.

Raises

TpGUISException,TpGeneralException

Method

sendInfoAndCollectReq()

This asynchronous method plays an announcement or sends other information to the user and collects some information from the user. The announcement usually prompts for a number of characters (for example, these are digits or text strings such as "YES" if the user's terminal device is a phone).

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

info : in TpUIInfo

Specifies the ID of the information to send to the user. This information can be:

- an infoID, identifying pre-defined information to be send (announcement and/or text);

- a string, defining the text to be sent;

- a URL , identifying pre-defined information or data to be sent to or downloaded into the terminal

language : in TpLanguage

Specifies the Language of the information to be send to the user.

variableInfo : in TpUIVariableInfoSet

Defines the variable part of the information to send to the user.

criteria : in TpUICollectCriteria

Specifies additional properties for the collection of information, such as the maximum and minimum number of characters, end character, first character timeout and inter-character timeout.

responseRequested : in TpUIResponseRequest

Specifies if a response is required from the call user interaction service, and any action the service should take. For this case it can especially be used to indicate e.g. the final request.

assignmentID : out TpAssignmentIDRef

Specifies the ID assigned by the generic user interaction interface for a user interaction request.

Raises

TpGUISException,TpGeneralException

Method

release()

This method requests that the relationship between the application and the user interaction object be released. It causes the release of the used user interaction resources and interrupts any ongoing user interaction.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction created.

Raises

TpGUISException,TpGeneralException

8.4 Interface Class IpAppUI

Inherits from: IpInterface.

The User Interaction Application Interface is implemented by the client application developer and is used to handle generic user interaction request responses and reports.

<<Interface>> IpAppUI
sendInfoRes (userInteractionSessionID : in TpSessionID, assignmentID : in TpAssignmentID, response : in TpUIReport) : TpResult sendInfoErr (userInteractionSessionID : in TpSessionID, assignmentID : in TpAssignmentID, error : in TpUIError) : TpResult sendInfoAndCollectRes (userInteractionSessionID : in TpSessionID, assignmentID : in TpAssignmentID, response : in TpUIReport, collectedInfo : in TpString) : TpResult sendInfoAndCollectErr (userInteractionSessionID : in TpSessionID, assignmentID : in TpAssignmentID, error : in TpUIError) : TpResult userInteractionFaultDetected (userInteractionSessionID : in TpSessionID, fault : in TpUIFault) : TpResult

Method

sendInfoRes()

This asynchronous method informs the application about the start or the completion of a sendInfoCallReq(). This response is called only if the responseRequested parameter of the sendInfoCallReq() method was set to P_UICALL_RESPONSE_REQUIRED.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

assignmentID : in TpAssignmentID

Specifies the ID assigned by the generic user interaction interface for a user interaction request.

response : in TpUIReport

Specifies the type of response received from the user.

Raises

TpGUISException,TpGeneralException

Method

sendInfoErr()

This asynchronous method indicates that the request to send information was unsuccessful.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

assignmentID : in TpAssignmentID

Specifies the ID assigned by the generic user interaction interface for a user interaction request.

error : in TpUIError

Specifies the error which led to the original request failing.

Raises

TpGUISException,TpGeneralException

Method

sendInfoAndCollectRes()

This asynchronous method returns the information collected to the application.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

assignmentID : in TpAssignmentID

Specifies the ID assigned by the generic user interaction interface for a user interaction request.

response : in TpUIReport

Specifies the type of response received from the user.

collectedInfo : in TpString

Specifies the information collected from the user.

Raises

TpGUISException,TpGeneralException

Method

sendInfoAndCollectErr()

This asynchronous method indicates that the request to send information and collect a response was unsuccessful.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

assignmentID : in TpAssignmentID

Specifies the ID assigned by the generic user interaction interface for a user interaction request.

error : in TpUIError

Specifies the error which led to the original request failing.

Raises

TpGUISException,TpGeneralException

Method

userInteractionFaultDetected()

This method indicates to the application that a fault has been detected in the user interaction.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the interface and sessionID of the user interaction service in which the fault has been detected.

fault : in TpUIFault

Specifies the fault that has been detected.

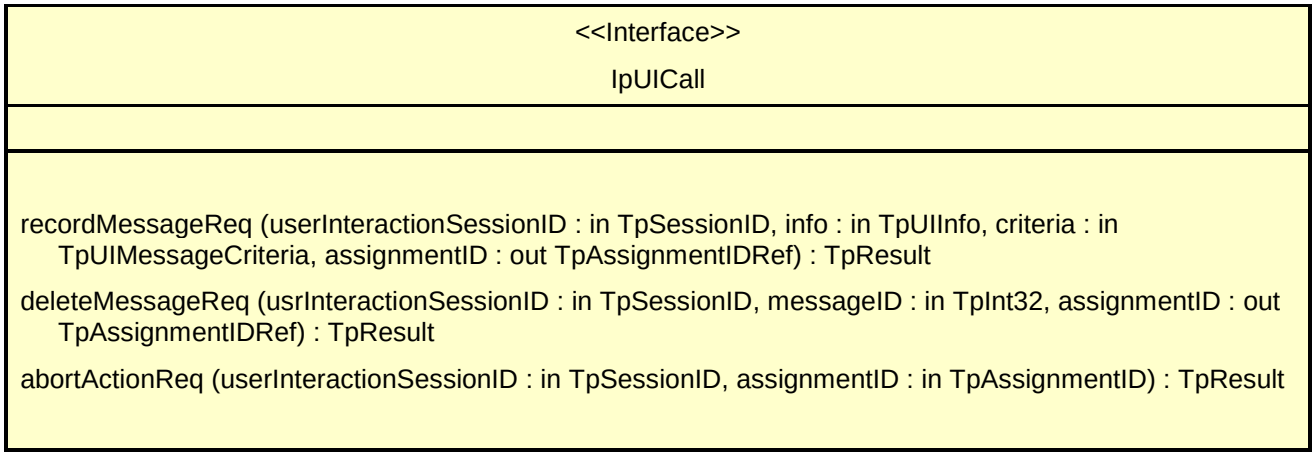
Raises

TpGUISException,TpGeneralException

8.5 Interface Class IpUICall

Inherits from: IpUI.

The Call User Interaction Service Interface provides functions to send information to, or gather information from the user (or call party) to which a call leg is connected. An application can use the Call User Interaction Service Interface only in conjunction with another service interface, which provides mechanisms to connect a call leg to a user. At present, only the Call Control service supports this capability.



Method

recordMessageReq()

This asynchronous method allows the recording of a message. The recorded message can be played back at a later time with the sendInfoReq() method.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

info : in TpUIInfo

Specifies the information to send to the user. This information can be either an ID (for pre-defined announcement or text), a text string, or an URL (indicating the information to be sent, e.g. an audio stream).

criteria : in TpUIMessageCriteria

Defines the criteria for recording of messages

assignmentID : out TpAssignmentIDRef

Specifies the ID assigned by the generic user interaction interface for a user interaction request.

Raises

TpGUISException,TpGeneralException

Method

deleteMessageReq()

This asynchronous method allows to delete a recorded message.

Parameters

usrInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

messageID : in TpInt32

Specifies the message ID.

assignmentID : out TpAssignmentIDRef

Specifies the ID assigned by the generic user interaction interface for a user interaction request.

Raises

TpGUISException,TpGeneralException

Method

abortActionReq()

This asynchronous method aborts a user interaction operation, e.g. a sendInfoReq(), from the specified call leg. The call and call leg are otherwise unaffected. The user interaction call service interrupts the current action on the specified leg.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

assignmentID : in TpAssignmentID

Specifies the user interaction request to be cancelled.

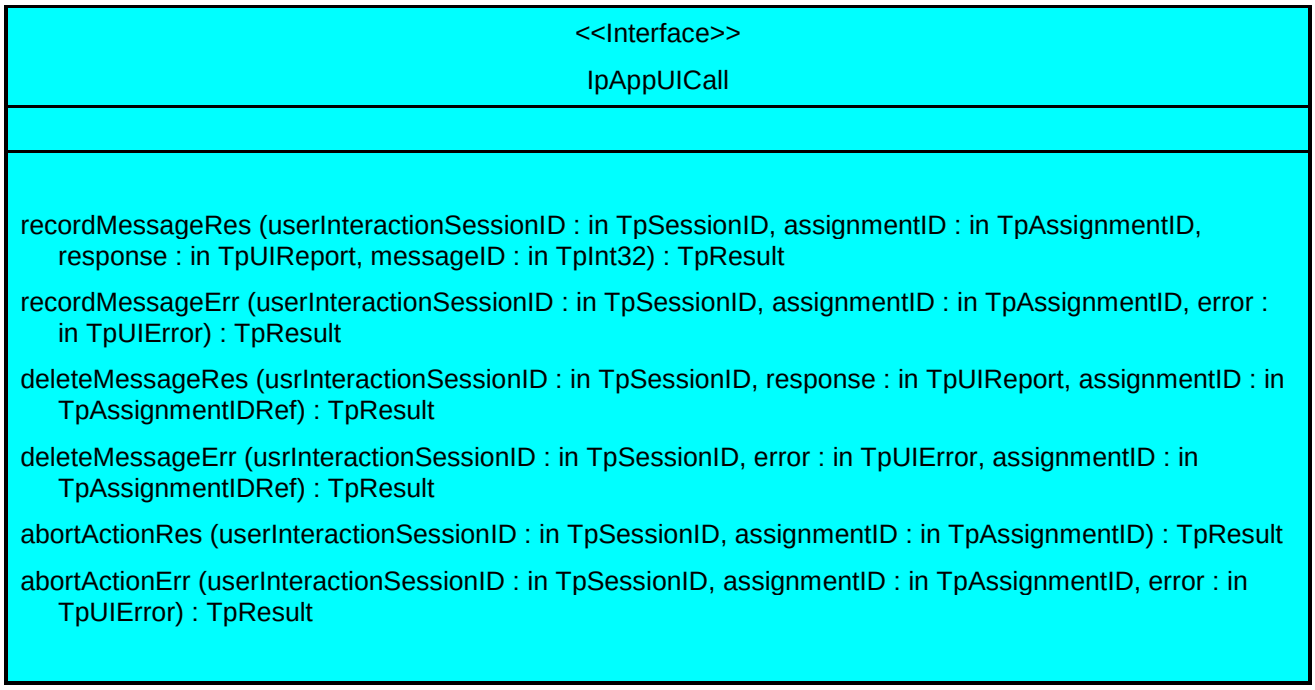
Raises

TpGUISException,TpGeneralException

8.6 Interface Class IpAppUICall

Inherits from: IpAppUI.

The Call User Interaction Application Interface is implemented by the client application developer and is used to handle call user interaction request responses and reports.



Method

recordMessageRes()

This method returns whether the message is successfully recorded or not. In case the message is recorded, the ID of the message is returned.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

assignmentID : in TpAssignmentID

Specifies the ID assigned by the call user interaction interface for a user interaction request.

response : in TpUIReport

Specifies the type of response received from the device where the message is stored.

messageID : in TpInt32

Specifies the ID that was assigned to the message by the device where the message is stored.

Method

recordMessageErr()

This method indicates that the request for recording of a message was not successful.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

assignmentID : in TpAssignmentID

Specifies the ID assigned by the call user interaction interface for a user interaction request.

error : in TpUIError

Specifies the error which led to the original request failing.

Method

deleteMessageRes()

This method returns whether the message is successfully deleted or not.

Parameters

usrInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

response : in TpUIReport

Specifies the type of response received from the device where the message was stored.

assignmentID : in TpAssignmentIDRef

Specifies the ID assigned by the call user interaction interface for a user interaction request.

Raises

TpGUISException,TpGeneralException

Method

deleteMessageErr()

This method indicates that the request for deleting a message was not successful.

Parameters

usrInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

error : in TpUIError

Specifies the error which led to the original request failing.

assignmentID : in TpAssignmentIDRef

Specifies the ID assigned by the call user interaction interface for a user interaction request.

Raises

TpGUISException,TpGeneralException

Method
abortActionRes()

This asynchronous method confirms that the request to abort a user interaction operation on a call leg was successful.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

assignmentID : in TpAssignmentID

Specifies the ID assigned by the call user interaction interface for a user interaction request.

Raises

TpGUISException,TpGeneralException

Method
abortActionErr()

This asynchronous method indicates that the request to abort a user interaction operation on a call leg resulted in an error.

Parameters

userInteractionSessionID : in TpSessionID

Specifies the user interaction session ID of the user interaction.

assignmentID : in TpAssignmentID

Specifies the ID assigned by the call user interaction interface for a user interaction request.

error : in TpUIError

Specifies the error which led to the original request failing.

Raises

TpGUISException,TpGeneralException

9 State Transition Diagrams

9.1 State Transition Diagrams for IpUIManager

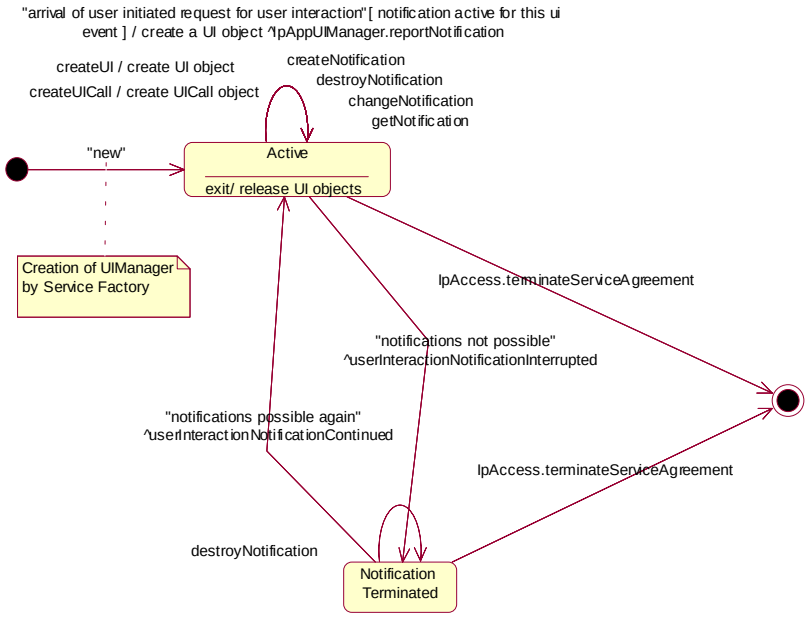


Figure : Application view on the UI Manager

9.1.1 Active State

In this state a relation between the Application and a User Interaction Service Capability Feature (Generic User Interaction or Call User Interaction) has been established. The application is now able to request creation of UI and/orUICall objects.

9.1.2 Notification Terminated State

When the UI manager is in the Notification terminated state, events requested with createNotification() will not be forwarded to the application. There can be multiple reasons for this: for instance it might be that the application receives more notifications than defined in the Service Level Agreement. Another example is that the SCS has detected it receives no notifications from the network due to e.g. a link failure. In this state no requests for new notifications will be accepted.

9.2 State Transition Diagrams for IpUI

The state transition diagram shows the application view on the User Interaction object.

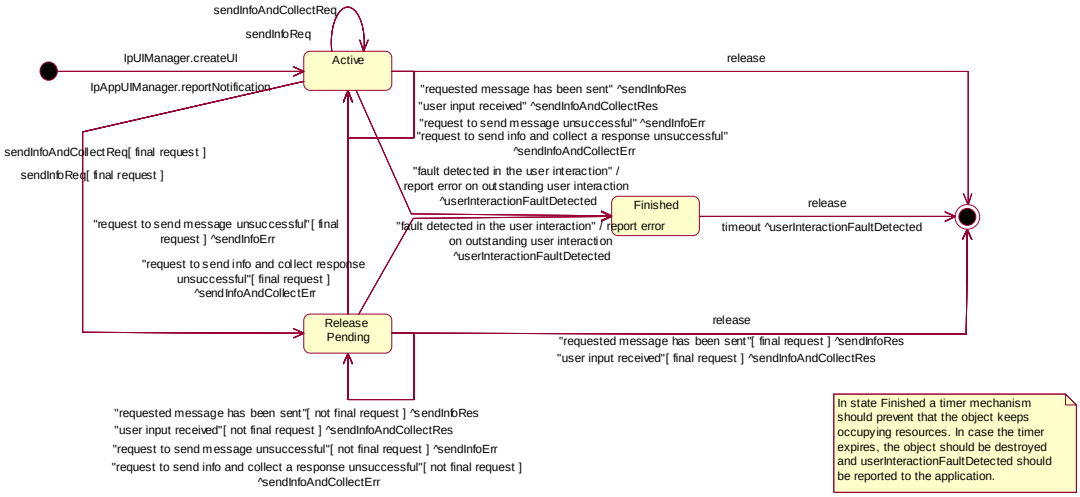


Figure : Application view on the UI object

9.2.1 Active State

In this state the UI object is available for requesting messages to be send to the network.

In case a fault is detected on the user interaction (e.g. a link failure to the IVR system), userInteractionFaultDetected() will be invoked on the application and an error will be reported on all outstanding requests.

9.2.2 Release Pending State

A transition to this state is made when the Application has indicated that after a certain message no further messages need to be sent to the end-user. There are, however, still a number of messages that are not yet completed. When the last message is sent or when the last user interaction has been obtained, the UI object is destroyed.

In case the final request failed or the application requested to abort the final request, a transition is made back to the Active state.

In case a fault is detected on the user interaction (e.g. a link failure to the IVR system), userInteractionFaultDetected() will be invoked on the application and an error will be reported on all outstanding requests.

9.2.3 Finished State

In this state the user interaction has ended. The application can only release the UI object. Note that the application has to release the object itself as good OO practice requires that when an object is created on behalf of a certain entity, this entity is also responsible for destroying it when the object is no longer needed.

9.3 State Transition Diagrams for IpUICall

The state transition diagram shows the application view on the Call User Interaction object.

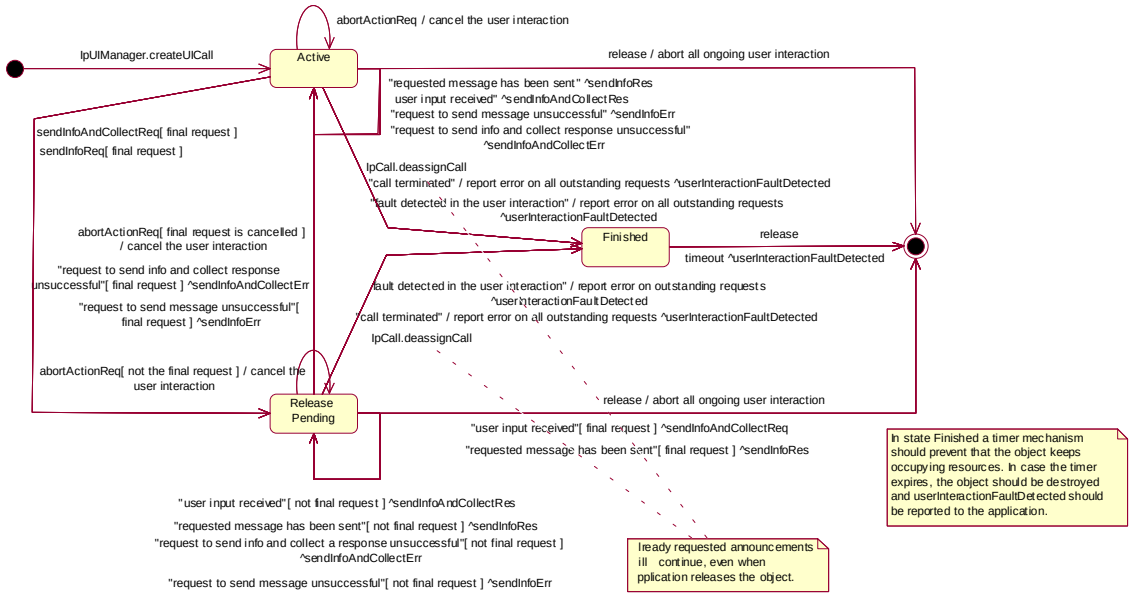


Figure : Application view on the UICall object

9.3.1 Active State

In this state a UICall object is available for announcements to be played to an end-user or obtaining information from the end-user.

When the application de-assigns the related Call object, a transition is made to the Finished state. However, all requested announcements will continue, even when the application releases the UICall object.

When the related call is due to some reason terminated, a transition is made to the Finished state, the operation userInteractionFaultDetected() will be invoked on the application and an error will be reported on all outstanding requests.

In case a fault is detected on the user interaction (e.g. a link failure to the IVR system), userInteractionFaultDetected() will be invoked on the application and an error will be reported on all outstanding requests.

9.3.2 Release Pending State

A transition to this state is made when the Application has indicated that after a certain announcement no further announcements need to be played to the end-user. There are, however, still a number of announcements that are not yet completed. When the last announcement is played or when the last user interaction has been obtained, the UICall object is destroyed. In case the final request failed or the application requested to abort the final request, a transition is made back to the Active state.

When the application de-assigns the related Call object, a transition is made to the Finished state. However, all requested announcements will continue, even when the application releases the UICall object.

When the related call is due to some reason terminated, a transition is made to the Finished state, the operation userInteractionFaultDetected() will be invoked on the application and an error will be reported on all outstanding requests.

In case a fault is detected on the user interaction (e.g. a link failure to the IVR system), userInteractionFaultDetected() will be invoked on the application and an error will be reported on all outstanding requests.

9.3.3 Finished State

In this state the user interaction has ended. The application can only release the UICall object. Note that the application has to release the object itself as good OO practice requires that when an object is created on behalf of a certain entity, this entity is also responsible for destroying it when the object is no longer needed.

10 Service Properties

10.1 User Interaction Service Properties

The following table lists properties relevant for the User Interaction API.

Property	Type	Description
P_INFO_TYPE	INTEGER_SET	Specifies whether the UI SCS supports text or URLs etc. Allowed value set: {P_INFO_ID, P_URL, P_TEXT}

The previous table lists properties related to capabilities of the SCS itself. The following table lists properties that are used in the context of the Service Level Agreement, e.g. to restrict the access of applications to the capabilities of the SCS.

Property	Type	Description
P_TRIGGERING_ADDRESSES	ADDRESS_RANGE_SET	Specifies which numbers the notification may be set
P_SERVICE_CODE	INTEGER_SET	Specifies the service codes that may be used for notification requests.

11 Data Definitions

11.1 TpUIFault

Defines the cause of the UI fault detected.

Name	Value	Description
P_UI_FAULT_UNDEFINED	0	Undefined
P_UI_CALL_ENDED	1	The related Call object has been terminated. Therefore, the UICall object is also terminated. No further interaction is possible with this object.

11.2 IpUI

Defines the address of an `IpUI` Interface.

11.3 IpUIRef

Defines a `Reference` to type `IpUI`.

11.4 IpUIRefRef

Defines a `Reference` to type `IpUIRef`.

11.5 IpAppUI

Defines the address of an `IpAppUI` Interface.

11.6 IpAppUIRef

Defines a `Reference` to type `IpAppUI`.

11.7 IpAppUIRefRef

Defines a `Reference` to type `IpAppUIRef.`

11.8 IpAppUIManager

Defines the address of an `IpAppUIManager` Interface.

11.9 IpAppUIManagerRef

Defines a `Reference` to type `IpAppUIManager.`

11.10 TpUICallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the UICall object

Structure Element Name	Structure Element Type	Structure Element Description
`UICallRef`	`IpUICallRef`	`This element specifies the interface reference for the UICall object.`
`UserInteractionSessionID`	`TpSessionID`	`This element specifies the user interaction session ID.`

11.11 TpUICallIdentifierRef

Defines a reference to type TpUICallIdentifier.

11.12 TpUICollectCriteria

Defines the `Sequence of Data Elements` that specify the additional properties for the collection of information, such as the end character, first character timeout, inter-character timeout, and maximum interaction time.

Structure Element Name	Structure Element Type
MinLength	TpInt32
MaxLength	TpInt32
EndSequence	TpString
StartTimeout	TpDuration
InterCharTimeout	TpDuration

The structure elements specify the following criteria:

MinLength: Defines the minimum number of characters (e.g. digits) to collect.

MaxLength: Defines the maxmum number of characters (e.g. digits) to collect.

EndSequence: Defines the character or characters which terminate an input of variable length, e.g. phonenumbers.

StartTimeout: specifies the value for the first character time-out timer. The timer is started when the announcement has been completed or has been interrupted. The user should enter the start of the response (e.g. first digit) before the timer expires. If the start of the response is not entered before the timer expires, the input is regarded to be erroneous. After receipt of the start of the response, which may be valid or invalid, the timer is stopped.

InterCharTimeOut: specifies the value for the inter-character time-out timer.The timer is started when a response (e.g. digit) is received, and is reset and restarted when a subsequent response is received. The responses may be valid or invalid. the announcement has been completed or has been interrupted.

Input is considered successful if the following applies:

If the EndSequence is not present (i.e. NULL):

- when the InterCharTimeOut timer expires; or
- when the number of valid digits received equals the MaxLength.

If the EndSequence is present:

- when the InterCharTimeOut timer expires; or
- when the EndSequence is received; or
- when the number of valid digits received equals the MaxLength.

In the case the number of valid characters received is less than the MinLength when the InterCharTimeOut timer expires or when the EndSequence is received, the input is considered erroneous.

The collected characters (including the EndSequence) are sent to the client application when input has been successful.

11.13 TpUIError

Defines the UI error codes.

Name	Value	Description
P_UI_ERROR_UNDEFINED	0	Undefined error
P_UI_ERROR_ILLEGAL_INFO	1	The specified information (InfoId, InfoData, or InfoAddress) is invalid
P_UI_ERROR_ID_NOT_FOUND	2	A legal InfoId is not known to the the User Interaction service
P_UI_ERROR_RESOURCE_UNAVAILABLE	3	The information resources used by the User Interaction

		service are unavailable, e.g. due to an overload situation.
P_UI_ERROR_ILLEGAL_RANGE	4	The values for minimum and maximum collection length are out of range
P_UI_ERROR_IMPROPER_USER_RESPONSE	5	Improper user response
P_UI_ERROR_ABANDON	6	The specified leg is disconnected before the send information completed
P_UI_ERROR_NO_OPERATION_ACTIVE	7	There is no active user interaction for the specified leg. Either the application did not start any user interaction or the user interaction was already finished when the `abortAction_Req()` was called.
P_UI_ERROR_NO_SPACE_AVAILABLE	8	There is no more storage capacity to record the message when the `recordMessage()` operation was called
P_UI_ERROR RESOURCE TIMEOUT	9	The request has been accepted by the resource but it did not report a result.

The call user interaction object will be automatically de-assigned if the error P_UI_ERROR_ABANDON is reported, as a corresponding call or call leg object no longer exists.

11.14 TpUIEventCriteria

Defines the `Sequence of Data Elements` that specify the additional criteria for receiving a UI notification

Structure Element Name	Structure Element Type	Description
OriginatingAddress	TpAddressRange	Defines the originating address for which the notification is requested.
DestinationAddress	TpAddressRange	Defines the destination address or address range for which the notification is requested.
ServiceCode	TpString	Defines a 2 digit code indicating the UI to be triggered. The value is operator specific.

11.15 TpUIEventCriteriaResultSetRef

Defines a reference to TpUIEventCriteriaResultSet

11.16 TPUIEventCriteriaResultSet

Defines a set of TpUIEventCriteriaResult

11.17 TPUIEventCriteriaResult

Defines a sequence of data elements that specify a requested event notification criteria with the associated assignmentID.

Structure Element Name	Structure Element Type	Structure Element Description
EventCriteria	TpUIEventCriteria	The event criteria that were specified by the application.
AssignmentID	TpInt32	The associated assignmentID. This can be used to disable the notification.

11.18 TpUIEventInfo

Defines the Sequence of Data Elements that specify a UI notification

Structure Element Name	Structure Element Type	
OriginatingAddress	TpAddress	Defines the originating address.
DestinationAddress	TpAddress	Defines the destination address.
ServiceCode	TpString	Defines a 2 digit code indicating the UI to be triggered. The value is operator specific.
DataTypeIndication	TpUIEventInfoDataType	Identifies the type of contents in the dataString.
DataString	TpString	Freely defined data string with a limited length e.g. 160 bytes according to the network policy.

11.19 TpUIEventInfoDataType

Defines the type of the dataString parameter in the method userInteractionEventNotify.

Name	Value	Description
P_UI_EVENT_DATA_TYPE_UNDEFINED	0	Undefined (e.g. binary data)
P_UI_EVENT_DATA_TYPE_UNSPECIFIED	1	Unspecified data
P_UI_EVENT_DATA_TYPE_TEXT	2	Text
P_UI_EVENT_DATA_TYPE_USSD_DATA	3	USSD data starting with coding scheme

11.20 TpUIIdentifier

Defines the Sequence of Data Elements that unambiguously specify the UI object

Structure Element Name	Structure Element Type	Structure Element Description
UIRef	IpUIRef	This element specifies the interface reference for the UI object.
UserInteractionSessionID	TpSessionID	This element specifies the user interaction session ID.

11.21 TpUIIdentifierRef

Defines a reference to type TpUIIdentifier.

11.22 TpUIInfo

Defines the Tagged Choice of Data Elements that specify the information to send to the user.

	Tag Element Type	
	TpUIInfoType	

Tag Element Value	Choice Element Type	Choice Element Name

P_UI_INFO_ID	TpInt32	InfoId
P_UI_INFO_DATA	TpString	InfoData
P_UI_INFO_ADDRESS	TpURL	InfoAddress

The choice elements represents the following:

`InfoID:` defines the ID of the user information script or stream to send to an end-user. The values of this data type are operator specific.

`InfoData:` defines the data to be sent to an end-user's terminal. The data is free-format and the encoding is depending on the resources being used..

`InfoAddress:` defines the URL of the text or stream to be sent to an end-user's terminal.

11.23 TpUIInfoType

Defines the type of the information to be send to the user.

Name	Value	Description
P_UI_INFO_ID	1	The information to be send to an end-user consists of an ID
P_UI_INFO_DATA	2	The information to be send to an end-user consists of a data string
P_UI_INFO_ADDRESS	3	The information to be send to an end-user consists of a URL.

11.24 TpUIMessageCriteria

Defines the `Sequence of Data Elements` that specify the additional properties for the recording of a message

Structure Element Name	Structure Element Type
EndSequence	TpString
MaxMessageTime	TpDuration
MaxMessageSize	TpInt32

The structure elements specify the following criteria:

`EndSequence:` Defines the character or characters which terminate an input of variable length, e.g. phonenumbers.

`MaxMessageTime:` specifies the maximum duration in seconds of the message that is to be recorded.

`MaxMessageSize:` If this parameter is non-zero, it specifies the maximum size in bytes of the message that is to be recorded.

11.25 TpUIReport

Defines the UI reports if a response was requested.

Name	Value	Description
P_UI_REPORT_UNDEFINED	0	Undefined report
P_UI_REPORT_INFO_SENT	1	Confirmation that the information has been sent
P_UI_REPORT_INFO_COLLECTED	2	Information collected., meeting the specified criteria.
P_UI_REPORT_NO_INPUT	3	No information collected. The user immediately entered the delimiter character. No valid information has been returned
P_UI_REPORT_TIMEOUT	4	No information collected. The user did not input any response before the input timeout expired
P_UI_REPORT_MESSAGE_STORED	5	A message has been stored successfully
P_UI_REPORT_MESSAGE_NOT_STORED	6	The message has not been stored successfully
P_UI_REPORT_MESSAGE_DELETED	7	A message has been deleted successfully
P_UI_REPORT_MESSAGE_NOT_DELETED	8	A message has not been deleted successfully

11.26 TpUIResponseRequest

Defines the situations for which a response is expected following the user interaction.

Name	Value	Description
P_UI_RESPONSE_REQUIRED	1	The User Interaction Call must send a response when the request has completed.
P_UI_LAST_ANNOUNCEMENT_IN_A_ROW	2	This is the final announcement within a sequence. It might, however, be that additional announcements will be requested at a later moment. The User Interaction Call service may release any used resources in the network. The UI object will not be released.
P_UI_FINAL_REQUEST	4	This is the final request. The UI object will be released after the information has been presented to the user.

This parameter represent a so-called bitmask, i.e. the values can be added to derived the final meaning.

11.27 TpUITargetObjectType

Defines the type of object where user interaction should be performed upon.

Name	Value	Description
P_UI_TARGET_OBJECT_CALL	0	User-interaction will be performed on a complete Call.
P_UI_TARGET_OBJECT_MULTI_PARTY_CALL	1	User-interaction will be performed on a complete Multi-party Call.
P_UI_TARGET_OBJECT_CALL_LEG	2	User-interaction will be performed on a single Call Leg.

11.28 TpUITargetObject

Defines the `Tagged Choice of Data Elements` that specify the object to perform user interaction on.

	Tag Element Type	
	TpUITargetObjectType	

Tag Element Value	Choice Element Type	Choice Element Name
P_UI_TARGET_OBJECT_CALL	TpCallIdentifier	Call
P_UI_TARGET_OBJECT_MULTI_PARTY_CALL	TpMultiPartyCallIdentifier	MultiPartyCall
P_UI_TARGET_OBJECT_CALL_LEG	TpCallLegIdentifier	CallLeg

11.29 TpUIVariableInfo

Defines the Tagged Choice of Data Elements that specify the variable parts in the information to send to the user.

	Tag Element Type	
	TpUIVariablePartType	

Tag Element Value	Choice Element Type	Choice Element Name
P_UI_VARIABLE_PART_INT	TpInt32	VariablePartInteger
P_UI_VARIABLE_PART_ADDRESS	TpString	VariablePartAddress
P_UI_VARIABLE_PART_TIME	TpTime	VariablePartTime
P_UI_VARIABLE_PART_DATE	TpDate	VariablePartDate
P_UI_VARIABLE_PART_PRICE	TpPrice	VariablePartPrice

11.30 TpUIVariableInfoSet

Defines a Numbered Set of Data Elements of TpUIVariableInfo.

11.31 TpUIVariablePartType

Defines the type of the variable parts in the information to send to the user.

Name	Value	Description
P_UI_VARIABLE_PART_INT	0	Variable part is of type integer
P_UI_VARIABLE_PART_ADDRESS	1	Variable part is of type address
P_UI_VARIABLE_PART_TIME	2	Variable part is of type time
P_UI_VARIABLE_PART_DATE	3	Variable part is of type date
P_UI_VARIABLE_PART_PRICE	4	Variable part is of type price

Annex A (normative): OMG IDL Description of User Interaction SCF

The OMG IDL representation of this interface specification is contained in a text file (ui.idl contained in archive 2919805IDL.ZIP) which accompanies the present document.

Annex B (informative): Differences between this draft and 3GPP 29.198 R99

B.1 Interface IpUIManager

createenableUINotification (appInterface appUIManager : in IpAppUIManagerRef, eventCriteria : in TpUIEventCriteria, assignmentID : out TpAssignmentIDRef) : TpResult

createUICall (appUI : in IpAppUICallRef, uiTargetObject : in TpUITargetObject, callIdentifier : in cc::TpCallIdentifier, callLegIdentifier : in cc::TpCallLegIdentifier, userInteraction : out TpUICallIdentifierRef) : TpResult

destroydisableUINotification (assignmentID : in TpAssignmentID) : TpResult

changeNotification (assignmentID : in TpAssignmentID, eventCriteria : in TpUIEventCriteria) : TpResult

getNotification (eventCriteria : out TpCallEventCriteriaResultSetRef) : TpResult

B.2 Interface IpAppUIManager

UserInteractionEventNotifyreportNotification (uiuserInteraction : in TpUIIdentifier , eventInfo : in TpUIEventInfo , assignmentID : in TpAssignmentID , appInterface appUI : out IpAppUIRefRef) : TpResult

B.3 Interface IpUI

sendInfoReq (userInteractionSessionID : in TpSessionID, info : in TpUIInfo, language : in TpLanguage, variableInfo : in TpUIVariableInfoSet, repeatIndicator : in TpInt32, responseRequested : in TpUIResponseRequest, assignmentID : out TpAssignmentIDRef) : TpResult

sendInfoReq (userInteractionSessionID : in TpSessionID, info : in TpUIInfo, language : in TpLanguage, variableInfo : in TpUIVariableInfoSet, repeatIndicator : in TpInt32, responseRequested : in TpUIResponseRequest, assignmentID : out TpAssignmentIDRef) : TpResult

B.4 Interface IpAppUI

sendInfoAndCollectRes(userInteractionSessionID : in TpSessionID , assignmentID : in TpAssignmentID, response : in TpUIReport , infocollectedInfo : in TpString) : TpResult

B.5 Interface IpUICall

The following method was added:

deleteMessageReq(userInteractionSessionID : in TpSessionID , messageID : in TpInt32 , assignmentID : out TpAssignmentIDRef) : TpResult

B.6 Interface IpAppUICall

The following methods were added:

deleteMessageRes(userInteractionSessionID : in TpSessionID , response : in TpUIReport , assignmentID : in TpAssignmentID) : TpResult

deleteMessageErr(userInteractionSessionID : in TpSessionID , error : in TpUIError , assignmentID : in TpAssignmentID) : TpResult

B.7 Type TpUIReport

TpUIReport

Defines the UI ~~call~~ reports if a response was requested.

Name	Value	Description
P_UI_REPORT_UNDEFINED	0	Undefined report
~~P_UI_REPORT_ANNOUNCEMENT_ENDED~~ P_UI_REPORT_INFO_SENT	1	Confirmation that the ~~announcement~~ information has ~~ended~~been sent
~~P_UI_REPORT_LEGAL_INPUT~~ P_UI_REPORT_INFO_COLLECTED	2	Information collected., meeting the specified criteria.
P_UI_REPORT_NO_INPUT	3	No information collected. The user immediately entered the delimiter character. No valid information has been returned
P_UI_REPORT_TIMEOUT	4	No information collected. The user did not input any response before the input timeout expired
P_UI_REPORT_MESSAGE_STORED	5	A message has been stored successfully
P_UI_REPORT_MESSAGE_NOT_STORED	6	The message has not been stored successfully
P_UI_REPORT_MESSAGE_DELETED	7	A message has been deleted successfully
P_UI_REPORT_MESSAGE_NOT_DELETED	8	A message has not been deleted successfully

B.8 Type TpUIError

TpUIError

Defines the UI ~~call~~ error codes.

Name	Value	Description
P_UI_ERROR_UNDEFINED	0	Undefined error
P_UI_ERROR_ILLEGAL_~~ID~~INFO	1	The specified information ~~id~~(InfoId, InfoData, or InfoAddress) ~~specified~~ is invalid
P_UI_ERROR_ID_NOT_FOUND	2	A legal ~~information id~~InfoId is not known to the User Interaction service
P_UI_ERROR_RESOURCE_UNAVAILABLE	3	The information resources used by the User Interaction service are unavailable, e.g. due to an overload situation.
P_UI_ERROR_ILLEGAL_RANGE	4	The values for minimum and maximum collection length are out of range
P_UI_ERROR_IMPROPER_~~CALLER~~_USER_RESPONSE	5	Improper user response
P_UI_ERROR_ABANDON	6	The specified leg is disconnected before the send information completed
P_UI_ERROR_NO_OPERATION_ACTIVE	7	There is no active user interaction for the specified leg. Either the application did not start any user interaction or the user interaction was already finished when the `abortAction_Req()` was called.
P_UI_ERROR_NO_SPACE_AVAILABLE	8	There is no more storage capacity to record the message when the `recordMessage()` operation was called
P_UI_ERROR RESOURCE TIMEOUT	9	The request has been accepted by the resource but it did not report a result.

B.9 Type TpUIEventCriteriaResult

TpUIEventCriteriaResultSetRef

Defines a reference to TpUIEventCriteriaResultSet

TPUIEventCriteriaResultSet

Defines a set of TpUIEventCriteriaResult

TPUIEventCriteriaResult

Defines a sequence of data elements that specify a requested event notification criteria with the associated assignmentID.

Structure Element Name	Structure Element Type	Structure Element Description
EventCriteria	TpUIEventCriteria	The event criteria that were specified by the application.
AssignmentID	TpInt32	The associated assignmentID. This can be used to disable the notification.

B.10 TpUITargetObjectType

TpUITargetObjectType

Defines the type of object where user interaction should be performed upon.

Name	Value	Description
P_UI_TARGET_OBJECT_CALL	0	User-interaction will be performed on a complete Call.
P_UI_TARGET_OBJECT_MULTI_PARTY_CALL	1	User-interaction will be performed on a complete Multi-party Call.
P_UI_TARGET_OBJECT_CALL_LEG	2	User-interaction will be performed on a single Call Leg.

TpUITargetObject

Defines the Tagged Choice of Data Elements that specify the object to perform user interaction on.

	Tag Element Type	
	TpUITargetObjectType	

Tag Element Value	Choice Element Type	Choice Element Name
P_UI_TARGET_OBJECT_CALL	TpCallIdentifier	Call
P_UI_TARGET_OBJECT_MULTI_PARTY_CALL	TpMultiPartyCallIdentifier	MultiPartyCall
P_UI_TARGET_OBJECT_CALL_LEG	TpCallLegIdentifier	CallLeg

B.11 TpUIVariableInfo

TpUIVariableInfo

Defines the Tagged Choice of Data Elements that specify the variable parts in the information to send to the user.

	Tag Element Type	
	TpUIVariablePartType	

Tag Element Value	Choice Element Type	Choice Element Name
P_UI_VARIABLE_PART_INT	TpInt32	VariablePartInteger
P_UI_VARIABLE_PART_ADDRESS	TpString	VariablePartAddress
P_UI_VARIABLE_PART_TIME	TpTime	VariablePartTime
P_UI_VARIABLE_PART_DATE	TpDate	VariablePartDate
P_UI_VARIABLE_PART_PRICE	TpPrice	VariablePartPrice

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for approval and placement under Change Control

3GPP TS 29.198-6 V.1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface
Part 6: Mobility
(Release 4)

Keywords
API, OSA, IDL, MM, Mobility, UL, ULC, US

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

This document is part of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA). The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA API's. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

This document specifies the Mobility Service Capability Feature (SCF) aspects of the interface. All aspects of the Mobility SCF are defined here, these being:

- Sequence Diagrams
- Class Diagrams
- Interface specification plus detailed method descriptions
- State Transition diagrams
- Data definitions
- IDL Description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modelling Language (UML).

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

[1] 3GPP TS 29.198-1 "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in TS 29.198-1 [1] apply.

3.2 Symbols

For the purposes of the present document, the symbols in TS 29.198-1 [1] apply.

3.3 Abbreviations

For the purposes of the present document, the abbreviations in TS 29.198-1 [1] apply.

4 Mobility SCF

The following sections describe each aspect of the Mobility Service Capability Feature (SCF).

The order is as follows:

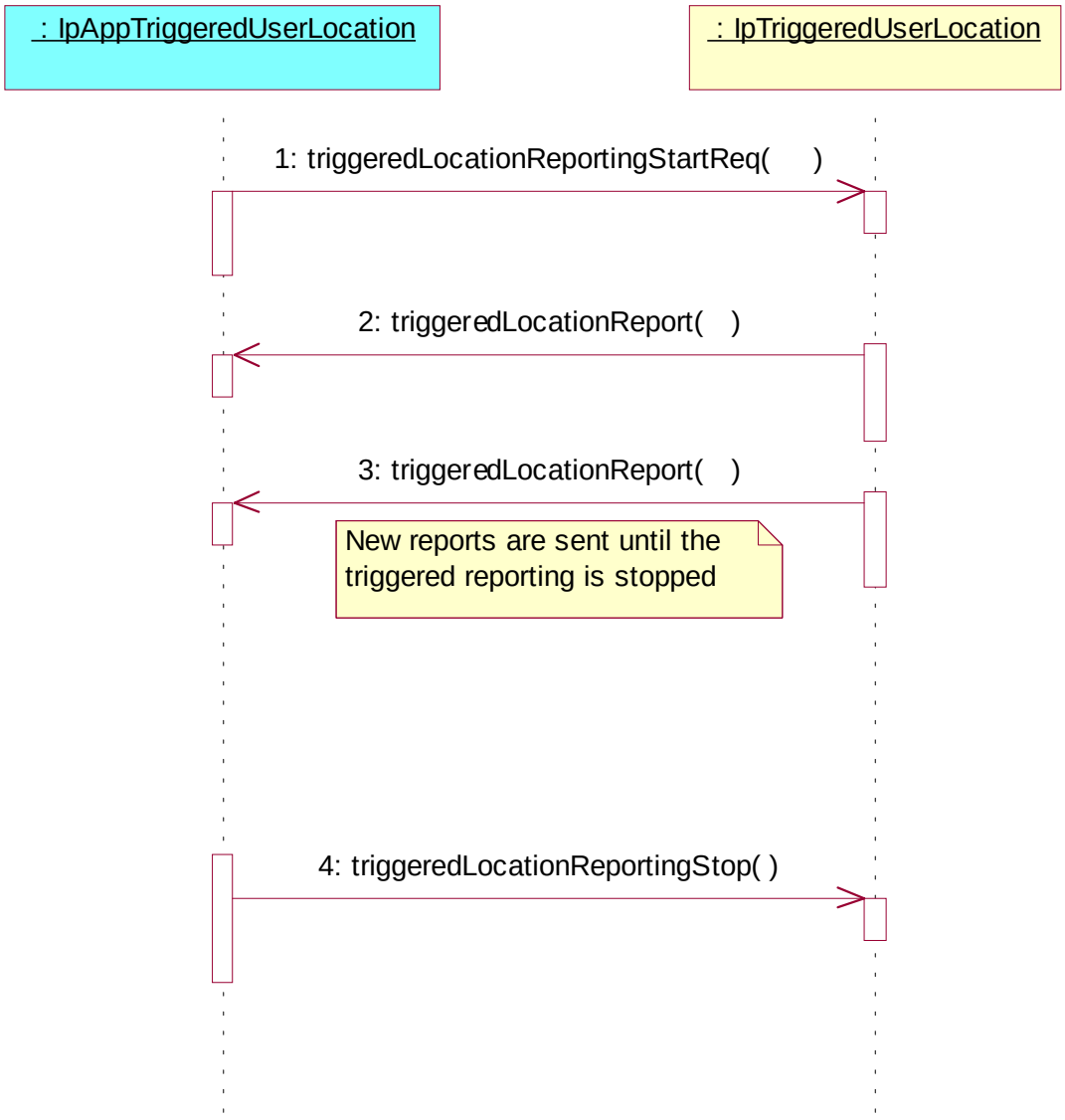
- The Sequence diagrams give the reader a practical idea of how each of the service capability feature is implemented.
- The Class relationships section show how each of the interfaces applicable to the SCF, relate to one another
- The Interface specification section describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the progression of internal processes either in the application, or Gateway.
- The Data definitions section show a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.

5 Sequence Diagrams

5.1 User Location Sequence Diagrams

5.1.1 User Location Interrogation - Triggered Request

The following sequence diagram shows how an application requests triggered location reports from the User Location service. When users location changes, the service reports this to the application.



1: This message is used to start triggered location reporting for one or several users.

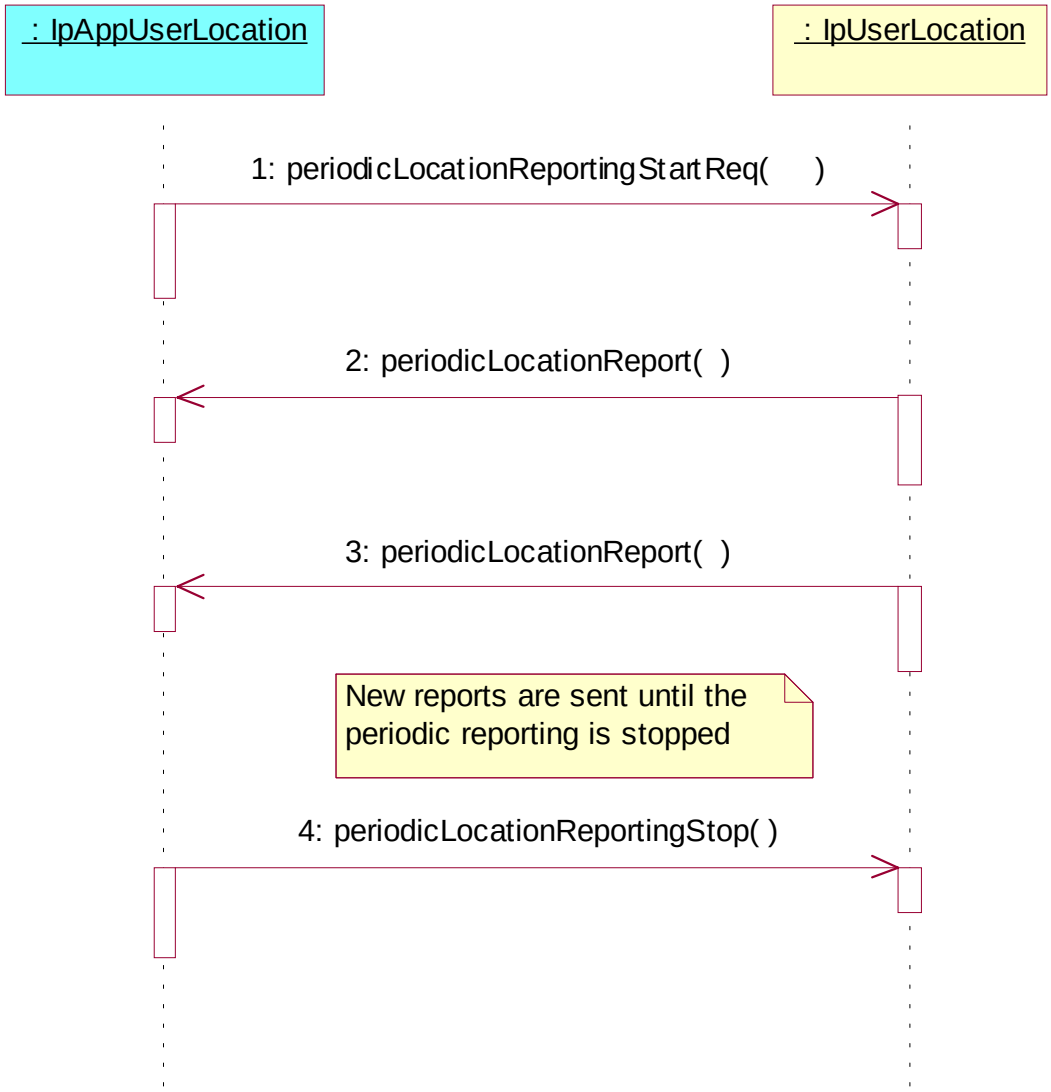
2: When the trigger condition is fulfilled then this message passes the location of the affected user to its callback object.

3: This is repeated until the application stops triggered location reporting (see next message).

4: This message is used to stop triggered location reporting.

5.1.2 User Location Interrogation - Periodic Request

The following sequence diagram shows how an application requests periodic location reports from the User Location service.



1: This message is used to start periodic location reporting for one or several users.

2: This message passes the location of one or several users to its callback object.

3: This message passes the location of one or several users to its callback object.

This is repeated at regular intervals until the application stops periodic location reporting (see next message).

4: This message is used to stop periodic location reporting.

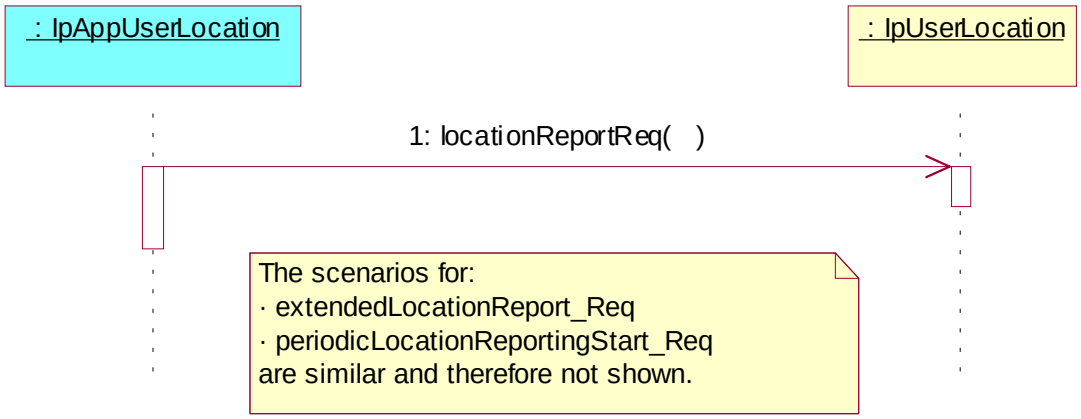
5.1.3 User Location Interrogation - Parameter Error

The following sequence diagram show a scenario where the application is requesting a location report from the User Location service but there is at least one error in the parameters that is detected by the service. The scenarios for:

· extendedLocationReportReq

· periodicLocationReportingStartReq

are similar and therefore not shown.



1: This message is used to request the location of one or several users, but the service returns an error and the execution of the request is aborted.

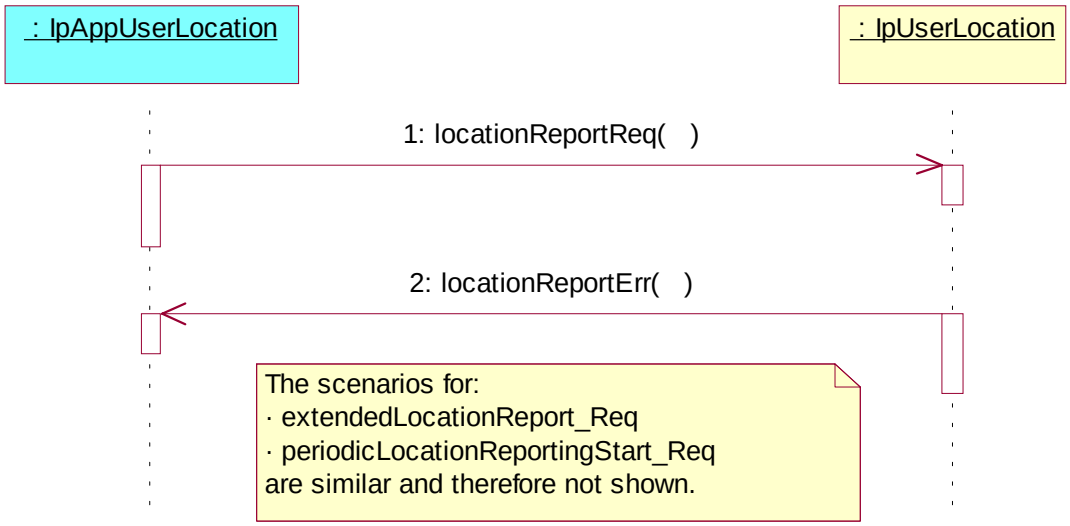
5.1.4 User Location Interrogation - Network Error

The following sequence diagram shows a scenario where the application is requesting a location report from the User Location service, but a network error occurs. The scenarios for:

· extendedLocationReportReq

· periodicLocationReportingStartReq

are similar and therefore not shown.

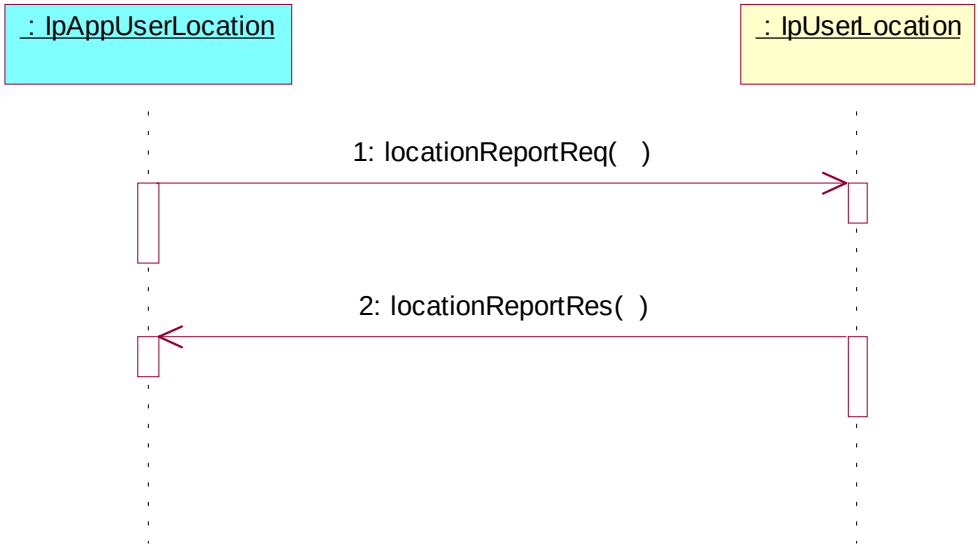


1: This message is used to request the location of one or several users.

2: This message passes information about the error in the location request from the network to the callback object.

5.1.5 User Location Interrogation - Interactive Request

The following sequence diagram shows how an application requests a location report from the User Location service.



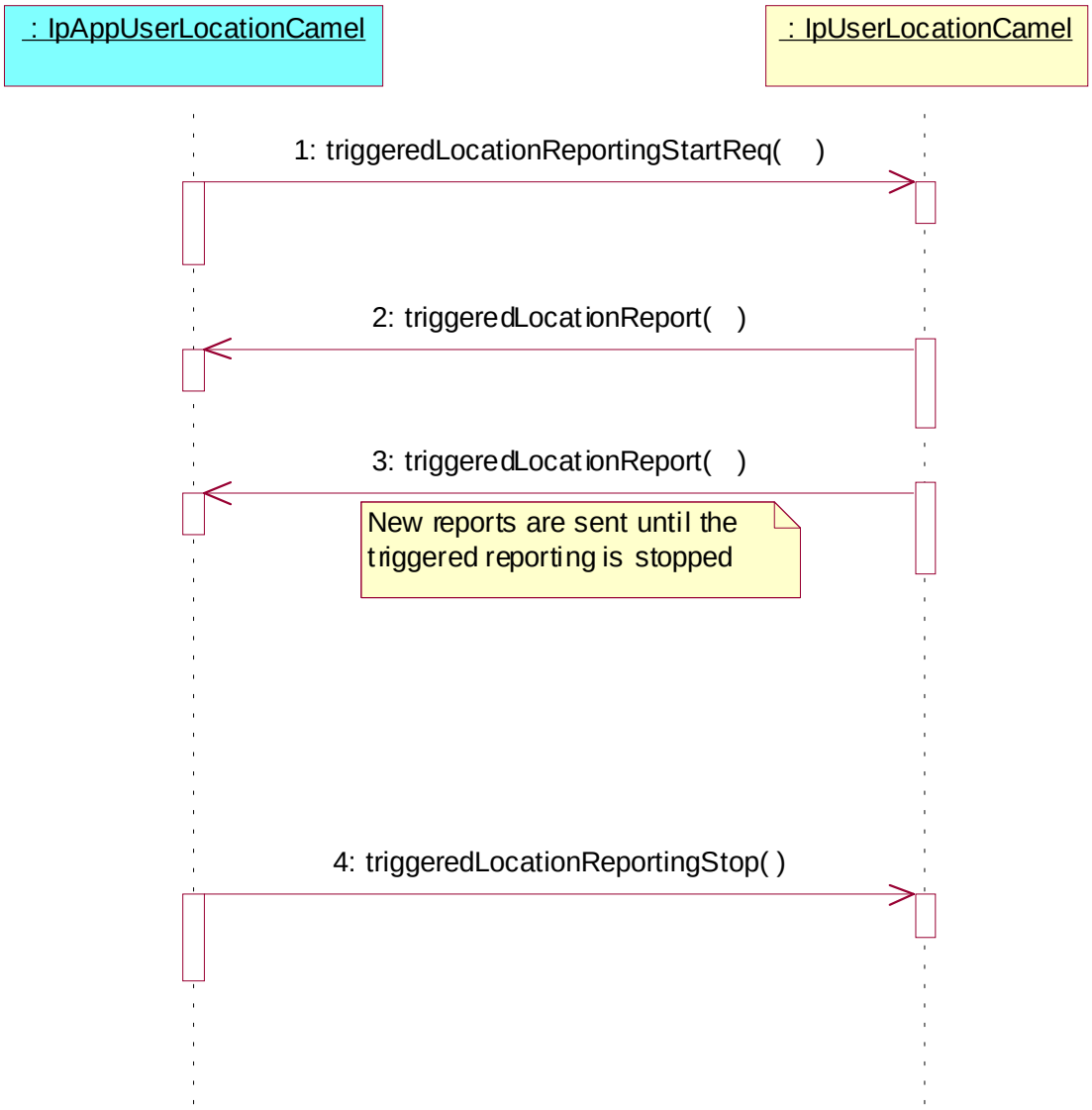
1: This message is used to request the location of one or several users.

2: This message passes the result of the location request for one or several users to its callback object.

5.2 User Location Camel Sequence Diagrams

5.2.1 User Location Camel Interrogation - Triggered Request

The following sequence diagram shows how an application requests triggered location reports from the User Location Camel service. When users location changes, the service reports this to the application.



1: This message is used to start triggered location reporting for one or several users.

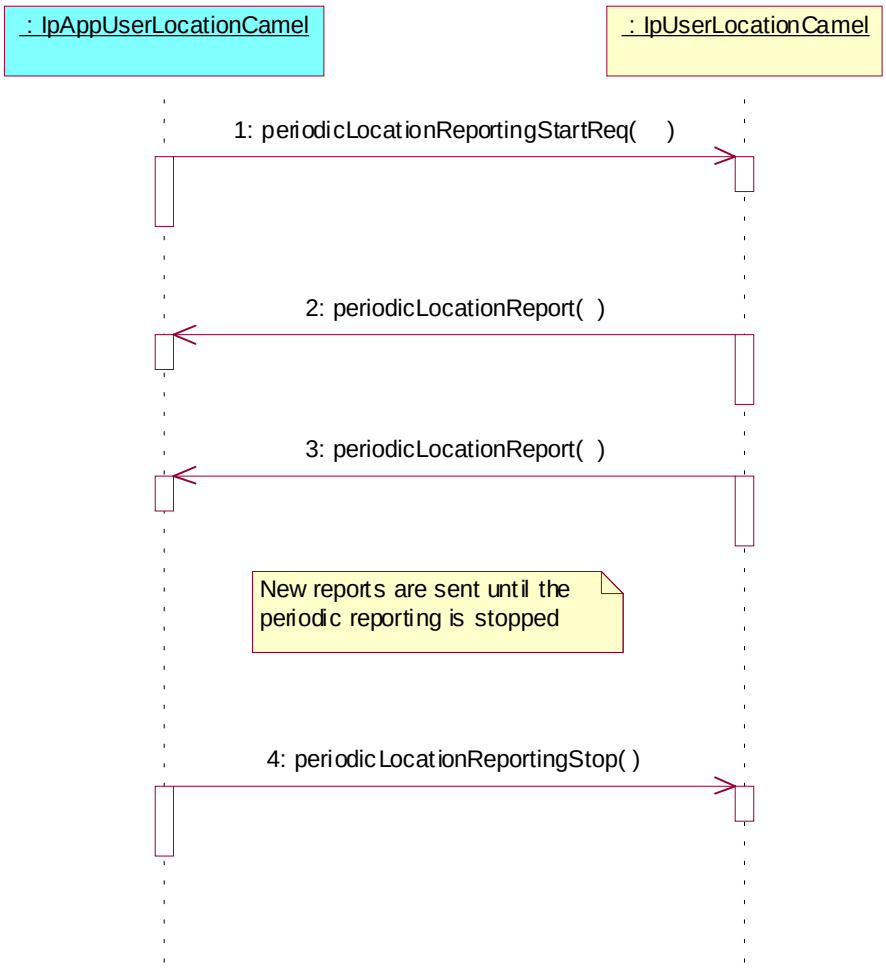
2: When the trigger condition is fulfilled then this message passes the location of the affected user to its callback object.

3: This is repeated until the application stops triggered location reporting (see next message).

4: This message is used to stop triggered location reporting.

5.2.2 User Location Camel Interrogation - Periodic Request

The following sequence diagram shows how an application requests periodic location reports from the User Location Camel service.



1: This message is used to start periodic location reporting for one or several users.

2: This message passes the location of one or several users to its callback object.

3: This message passes the location of one or several users to its callback object.

This is repeated at regular intervals until the application stops periodic location reporting (see next message).

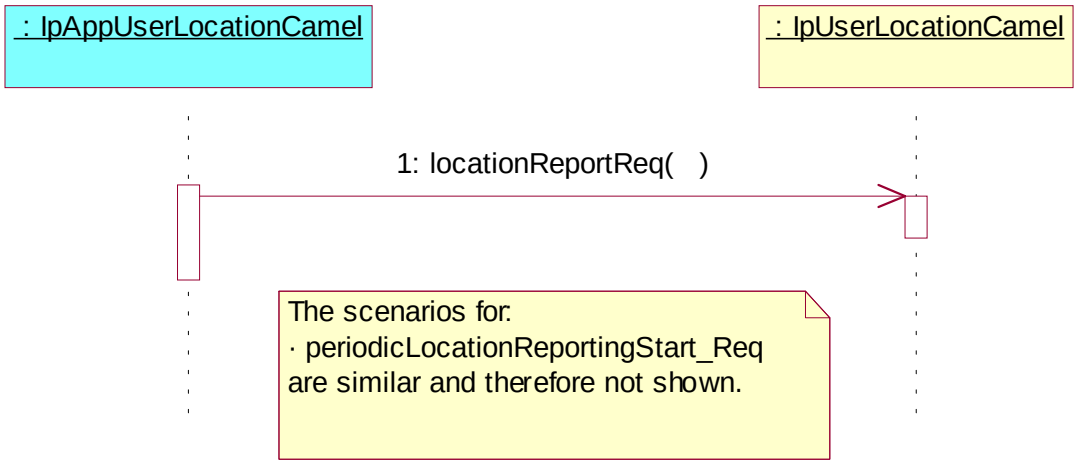
4: This message is used to stop periodic location reporting.

5.2.3 User Location Camel Interrogation - Parameter Error

The following sequence diagram show a scenario where the application is requesting a location report from the User Location Camel service but there is at least one error in the parameters that is detected by the service. The scenarios for:

· periodicLocationReportingStartReq

are similar and therefore not shown.



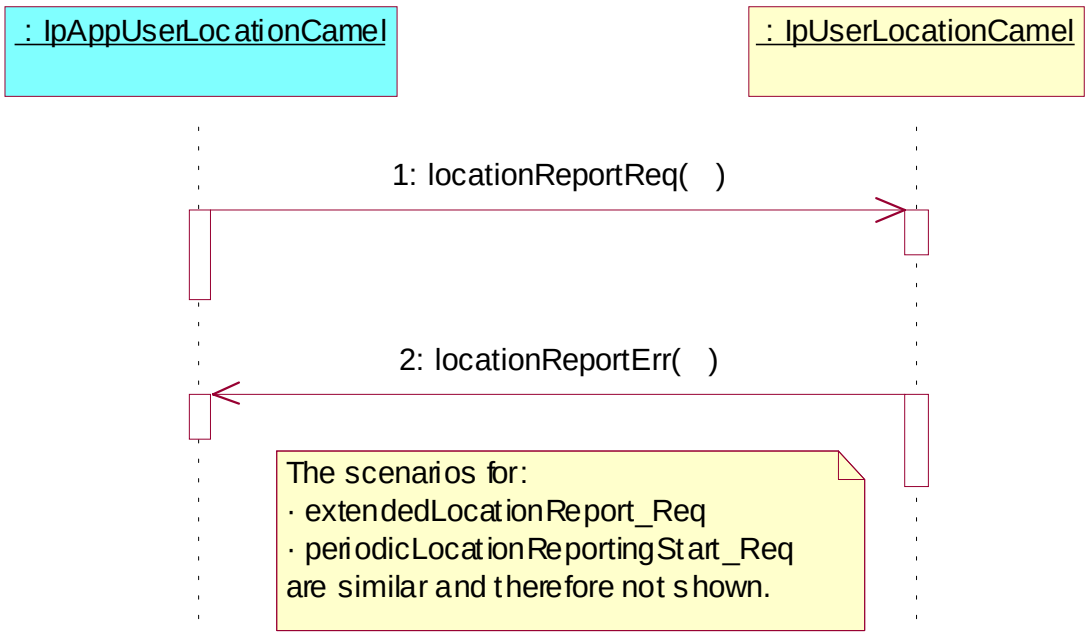
1: This message is used to request the location of one or several users, but the service returns an error and the execution of the request is aborted.

5.2.4 User Location Camel Interrogation - Network Error

The following sequence diagram shows a scenario where the application is requesting a location report from the User Location Camel service, but a network error occurs. The scenarios for:

· periodicLocationReportingStartReq

are similar and therefore not shown.

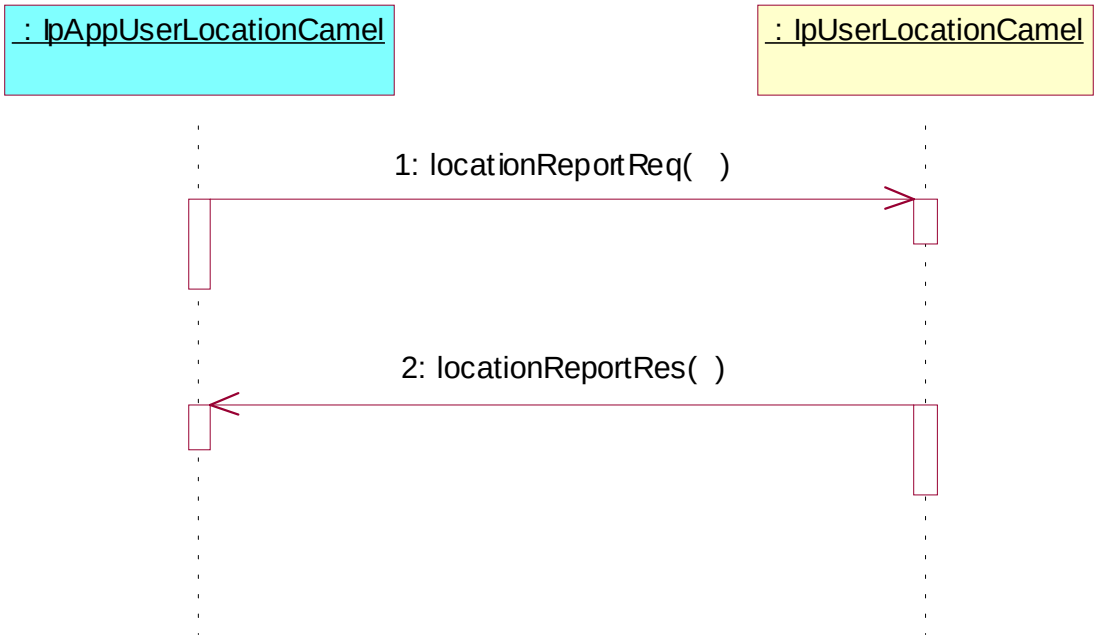


1: This message is used to request the location of one or several users.

2: This message passes information about the error in the location request from the network to the callback object.

5.2.5 User Location Camel Interrogation - Interactive Request

The following sequence diagram shows how an application requests a location report from the User Location Camel service.



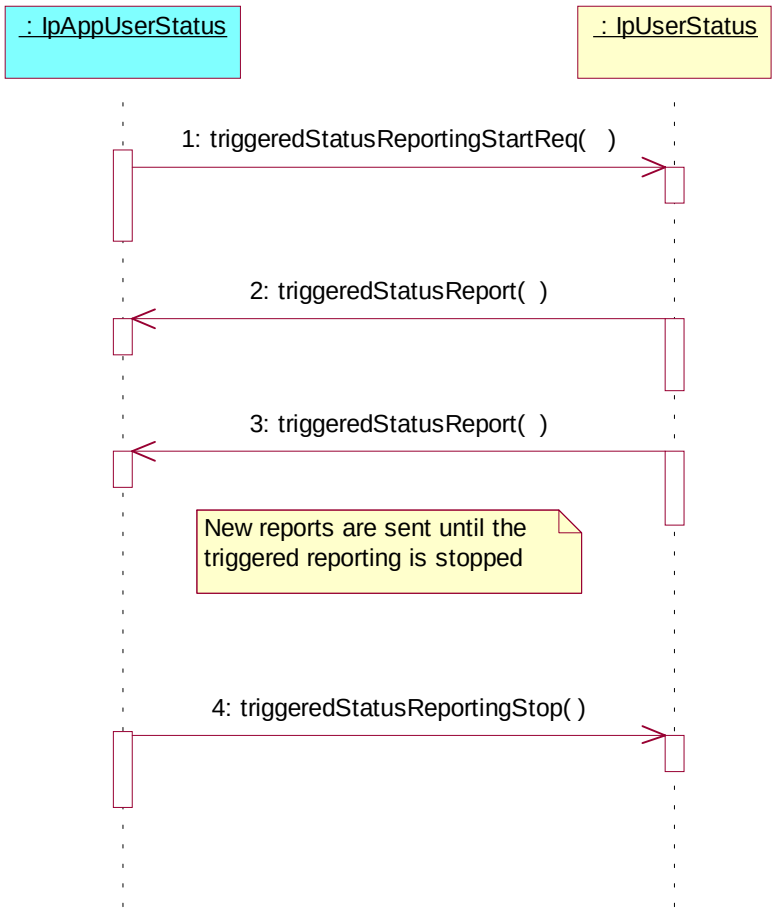
1: This message is used to request the location of one or several users.

2: This message passes the result of the location request for one or several users to its callback object.

5.3 User Status Sequence Diagrams

5.3.1 Triggered Reporting

The following sequence diagram shows how an application requests triggered status reports from the Status Location service. When user's status changes, the service reports this to the application.



1: This message is used to start triggered status reporting for one or several users.

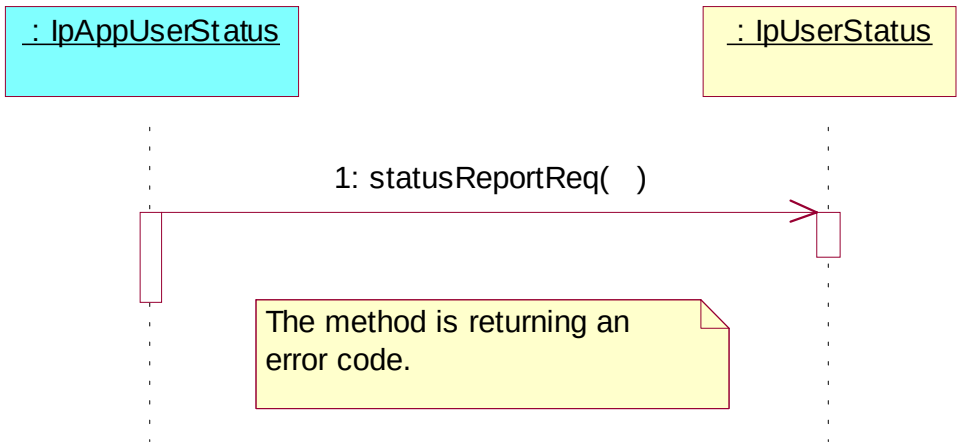
2: When a user's status changes, this message passes the status to its callback object.

3: This is repeated until the application stops triggered status reporting (see next message).

4: This message is used to stop triggered status reporting.

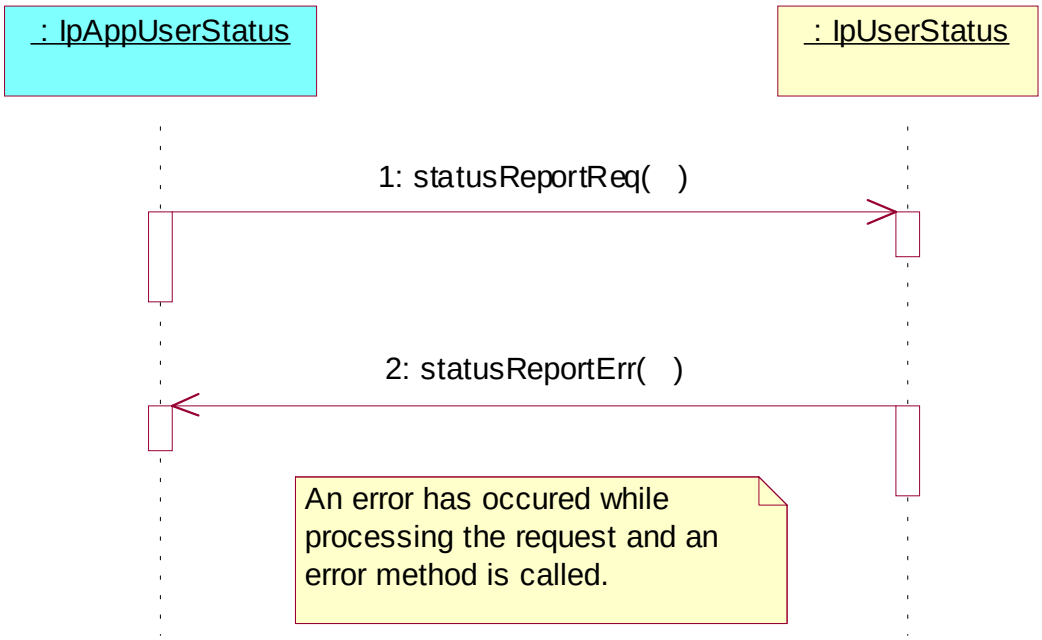
5.3.2 Interactive Request Parameter Error

The following sequence diagram shows, how an application requests a status report from the User Status service, but the service discovers an error and returns an error code.



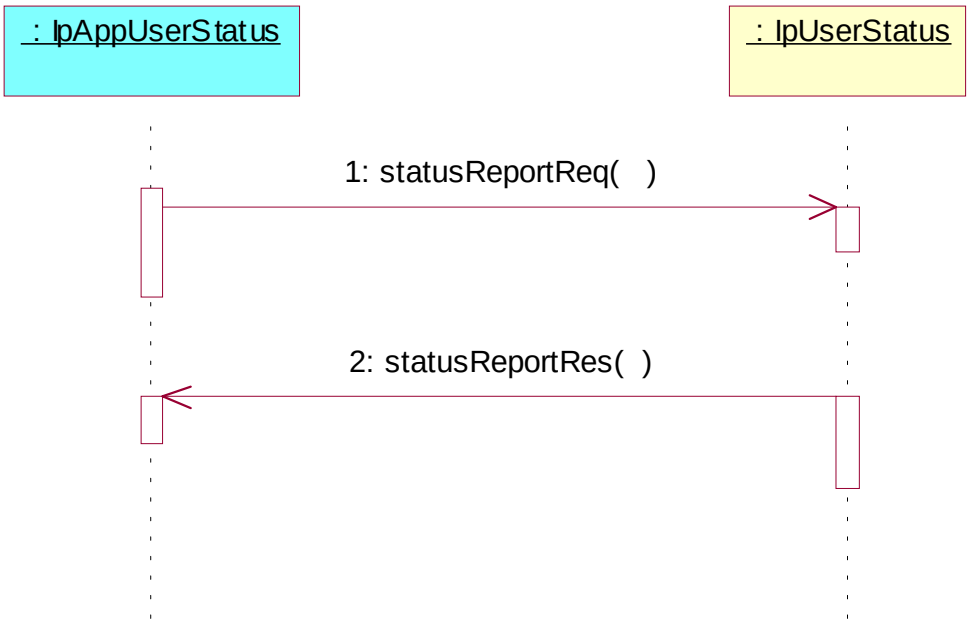
5.3.3 Interactive Request Network Error

The following sequence diagram shows, how an application requests a status report from the User Status service, but later, when the request is processed, the service discovers an error and calls an error method.



5.3.4 Interactive Request

The following sequence diagram shows how an application requests a status report from the User Status service.



1: This message is used to request the status of one or several users.

2: This message passes the result of the status request to its callback object.

6 Class Diagrams

6.1 User Location Class Diagrams

This class diagram shows the relationship between the interfaces in the User Location service. IpTriggeredUserLocation inherits from IpUserLocation, and IpAppTriggeredUserLocation inherits from IpAppUserLocation.

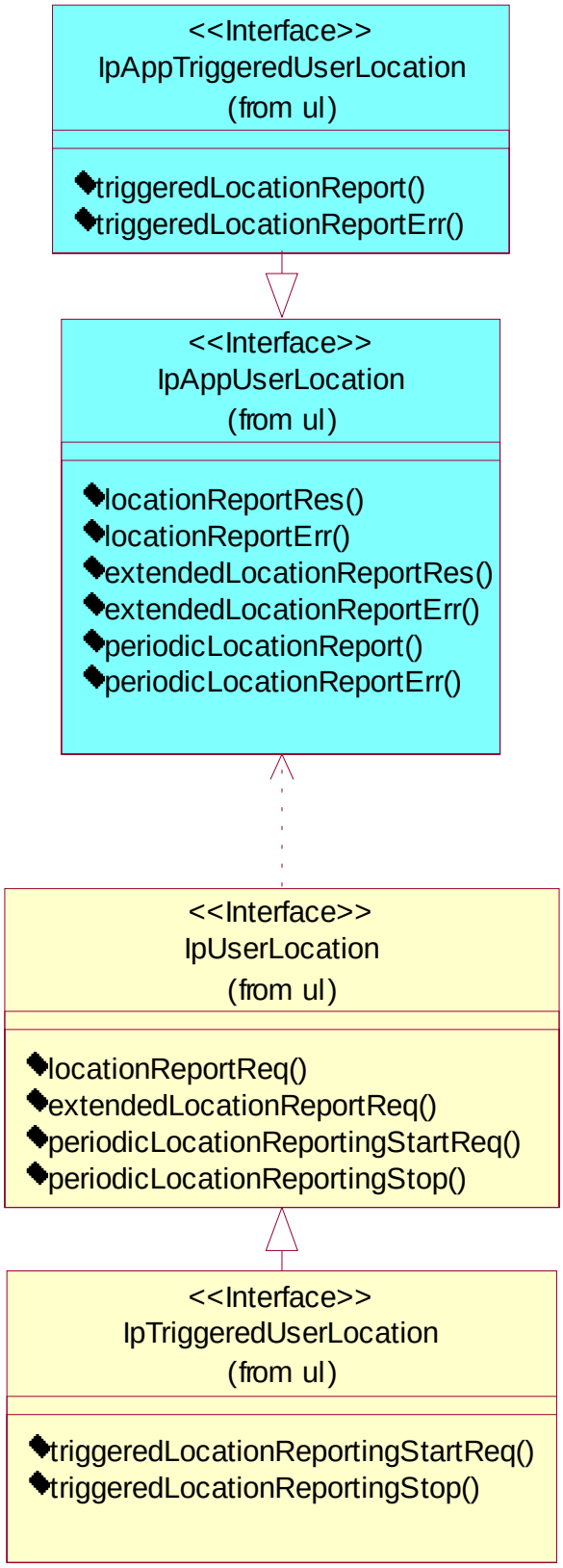


Figure 1: User Location Class Diagram

6.2 User Location Camel Class Diagrams

This class diagram shows the interfaces for the User Location Camel service.

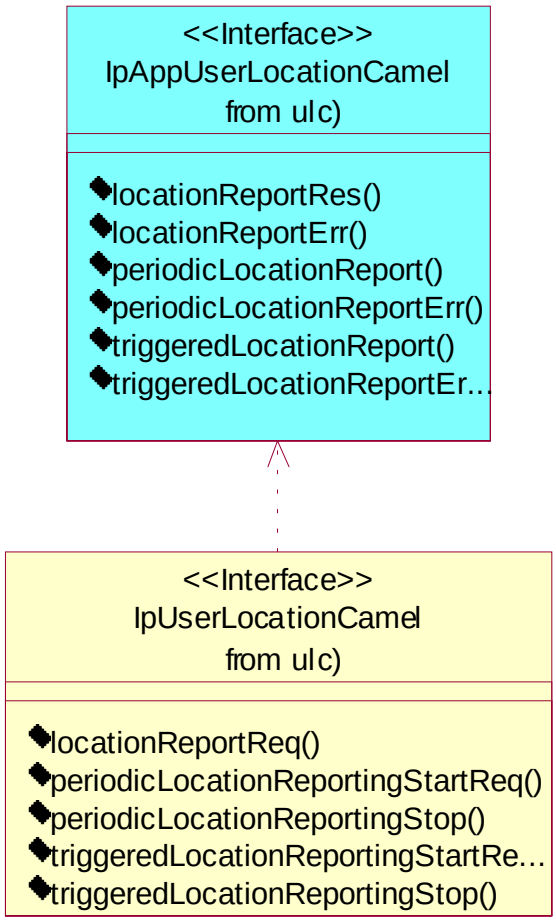


Figure 2: User Location Camel Class Diagram

6.3 User Status Class Diagrams

This class diagram shows the interfaces for the User Status service.

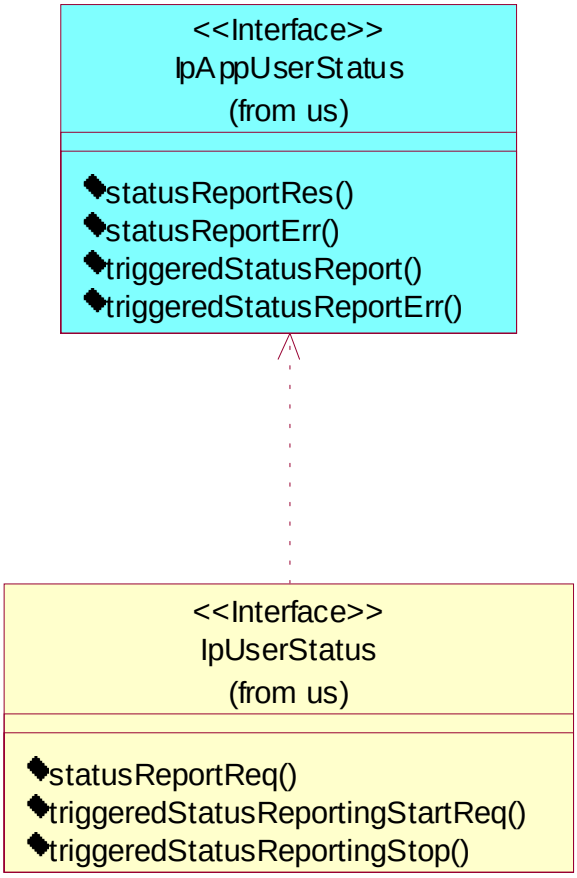


Figure 3: User Status Class Diagram

7 The Service Interface Specifications

7.1 Interface Specification Format

This section defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

7.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for enterprise-based client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>

7.1.2 Method descriptions

Each method (API method “call”) is described. All methods in the API return a value of type `TpResult`, indicating, amongst other things, if the method invocation was sucessfully executed or not.

Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a '`Req`' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a '`Res`' or '`Err`' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name> or IpSvc<name>` interfaces to provide the callback mechanism.

7.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.

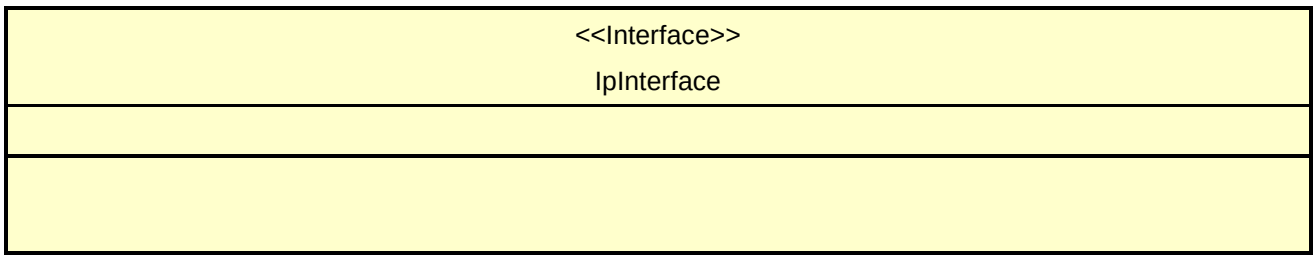
7.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

7.2 Base Interface

7.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.



7.3 Service Interfaces

7.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

7.4 Generic Service Interface

7.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.

<<Interface>> IpService
setCallback (appInterface : in IpInterfaceRef) : TpResult setCallbackWithSessionID (appInterface : in IpInterfaceRef, sessionID : in TpSessionID) : TpResult

Method
`setCallback()`

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application.

Parameters

`appInterface : in IpInterfaceRef`

Specifies a reference to the application interface, which is used for callbacks

Raises

`TpGeneralException`

Method
`setCallbackWithSessionID()`

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg.

Parameters

`appInterface : in IpInterfaceRef`

Specifies a reference to the application interface, which is used for callbacks

`sessionID : in TpSessionID`

Specifies the session for which the service can invoke the application's callback interface.

Raises

`TpGeneralException`

8 Mobility Interface Classes

8.1 User Location Interface Classes

The User Location service (UL) provides a general geographic location service. UL has functionality to allow applications to obtain the geographical location and the status of fixed, mobile and IP based telephony users.

UL is supplemented by User Location Camel service (ULC) to provide information about network related information. There is also some specialised functionality to handle emergency calls in the User Location Emergency service (ULE).

The UL service provides the IpUserLocation and IpTriggeredUserLocation interfaces. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppUserLocation and IpAppTriggeredUserLocation interfaces to provide the callback mechanism.

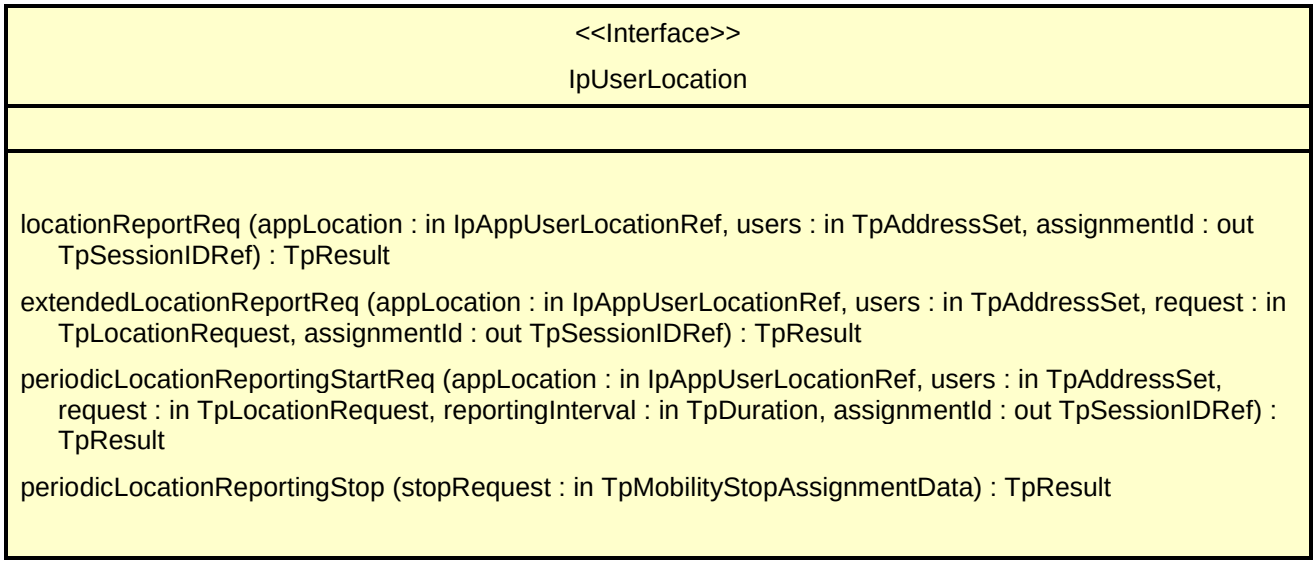
When periodic or triggered location reporting is used, errors may be reported either when the recurrent reporting is requested, as an error per user in reports or in the corresponding err-method when the error concerns all subscribers in an assignment.

8.1.1 Interface Class IpUserLocation

Inherits from: IpService.

This interface is the 'service manager' interface for the User Location Service.

The user location interface provides the management functions to the user location service. The application programmer can use this interface to obtain the geographical location of users.



Method
`locationReportReq()`

Request of a report on the location for one or several users.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

P_RESOURCES_UNAVAILABLE

The required resources in the network are not available. The application may try to invoke the method at a later time.

P_USER_NOT_SUBSCRIBED

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_USER_PRIVACY

The requests violates the end-user's privacy setting.

Parameters

appLocation : in IpAppUserLocationRef

Specifies the application interface for callbacks from the User Location service.

users : in TpAddressSet

Specifies the user(s) for which the location shall be reported.

assignmentId : out TpSessionIDRef

Specifies the assignment ID of the location-report request.

Raises

TpGeneralException

Method

extendedLocationReportReq()

Advanced request of report on the location for one or several users.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

P_RESOURCES_UNAVAILABLE

The required resources in the network are not available. The application may try to invoke the method at a later time.

P_USER_NOT_SUBSCRIBED

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_USER_PRIVACY

The requests violates the end-user's privacy setting.

Parameters

appLocation : in IpAppUserLocationRef

Specifies the application interface for callbacks from the User Location service.

users : in TpAddressSet

Specifies the user(s) for which the location shall be reported

request : in TpLocationRequest

Specifies among others the requested location type, accuracy, response time and priority.

assignmentId : out TpSessionIDRef

Specifies the assignment ID of the extended location-report request.

Raises

TpGeneralException

Method

periodicLocationReportingStartReq()

Request of periodic reports on the location for one or several users.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

P_RESOURCES_UNAVAILABLE

The required resources in the network are not available. The application may try to invoke the method at a later time.

P_USER_NOT_SUBSCRIBED

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_USER_PRIVACY

The requests violates the end-user's privacy setting.

Parameters

appLocation : in IpAppUserLocationRef

Specifies the application interface for callbacks from the User Location service.

users : in TpAddressSet

Specifies the user(s) for which the location shall be reported.

request : in TpLocationRequest

Specifies among others the requested location type, accuracy, response time and priority.

reportingInterval : in TpDuration

Specifies the requested interval in seconds between the reports.

assignmentId : out TpSessionIDRef

Specifies the assignment ID of the periodic location-reporting request.

Raises

TpGeneralException

Method

periodicLocationReportingStop()

Termination of periodic reports on the location for one or several users.

Raises the following exceptions:

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

stopRequest : in TpMobilityStopAssignmentData

Specifies how the assignment shall be stopped, i.e. if whole or just parts of the assignment should be stopped.

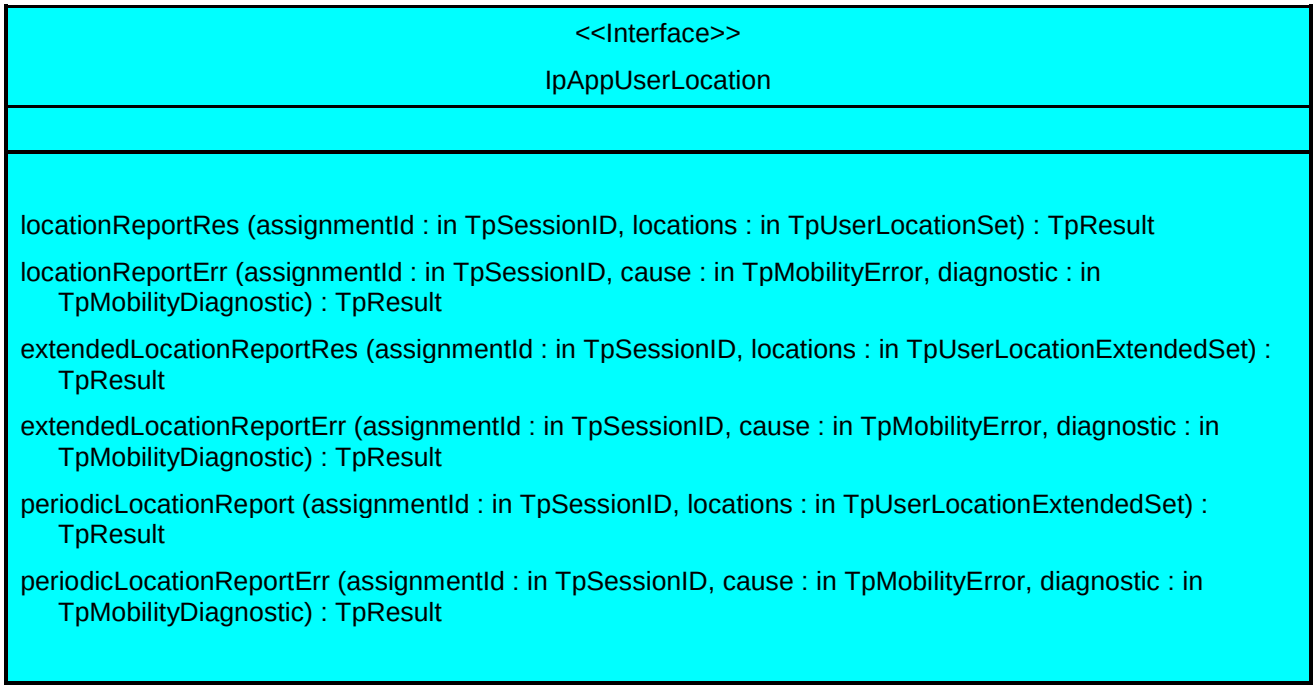
Raises

TpGeneralException

8.1.2 Interface Class IpAppUserLocation

Inherits from: IpInterface.

The user-location application interface is implemented by the client application developer and is used to handle user location request responses.



Method

`locationReportRes()`

A report containing locations for one or several users is delivered.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

`assignmentId : in TpSessionID`

Specifies the assignment ID of the location-report request.

`locations : in TpUserLocationSet`

Specifies the location(s) of one or several users.

Raises

`TpGeneralException`

Method

locationReportErr()

This method indicates that the location report request has failed.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the failed location report request.

cause : in TpMobilityError

Specifies the error that led to the failure.

diagnostic : in TpMobilityDiagnostic

Specifies additional information about the error that led to the failure.

Method

extendedLocationReportRes()

A report containing extended location information for one or several users is delivered.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the extended location-report request.

locations : in TpUserLocationExtendedSet

Specifies the location(s) of one or several users.

Raises

TpGeneralException

Method

extendedLocationReportErr()

This method indicates that the extended location report request has failed.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the failed extended location report request.

cause : in TpMobilityError

Specifies the error that led to the failure.

diagnostic : in TpMobilityDiagnostic

Specifies additional information about the error that led to the failure.

Method

periodicLocationReport()

A report containing periodic location information for one or several users is delivered.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the periodic location-reporting request.

locations : in TpUserLocationExtendedSet

Specifies the location(s) of one or several users.

Raises

TpGeneralException

Method

periodicLocationReportErr()

This method indicates that a requested periodic location report has failed. Note that errors only concerning individual users are reported in the ordinary periodicLocationReport() message.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the failed periodic location reporting start request.

cause : in TpMobilityError

Specifies the error that led to the failure.

diagnostic : in TpMobilityDiagnostic

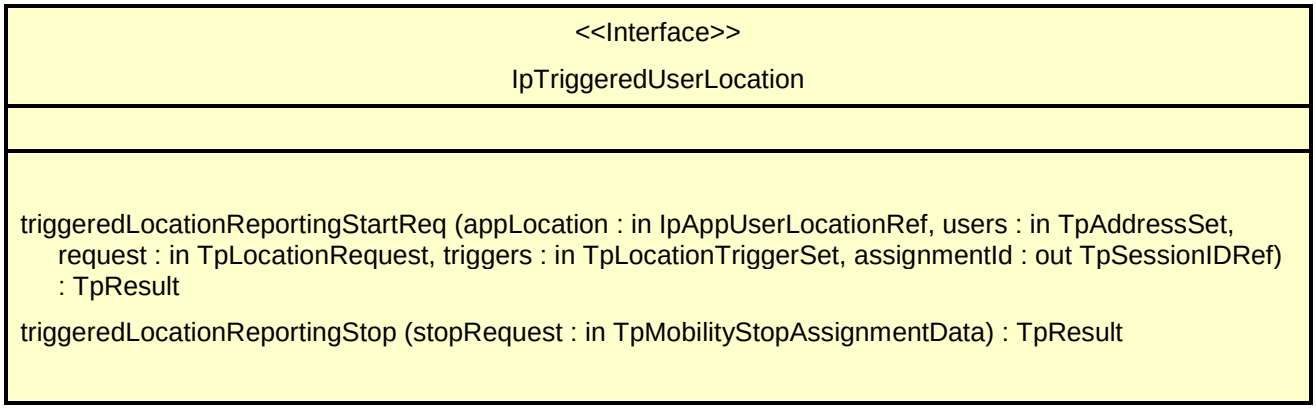
Specifies additional information about the error that led to the failure.

8.1.3 Interface Class IpTriggeredUserLocation

Inherits from: IpUserLocation.

This interface can be used as an extended version of the User Location: Service Interface.

The triggered user location interface represents the interface to the triggered user location functions. The application programmer can use this interface to request user location reports that are triggered by location change.



Method
`triggeredLocationReportingStartReq()`

Request for user location reports when the location is changed (reports are triggered by location change).

Parameters

`appLocation : in IpAppUserLocationRef`

Specifies the application interface for callbacks from the User Location service.

`users : in TpAddressSet`

Specifies the user(s) for which the location shall be reported.

`request : in TpLocationRequest`

Specifies among others the requested location type, accuracy, response time and priority.

`triggers : in TpLocationTriggerSet`

Specifies the trigger conditions.

`assignmentId : out TpSessionIDRef`

Specifies the assignment ID of the triggered location-reporting request.

Raises

`TpGeneralException`

Method
`triggeredLocationReportingStop()`

Stop triggered user location reporting.

Parameters

`stopRequest : in TpMobilityStopAssignmentData`

Specifies how the assignment shall be stopped, i.e. if whole or just parts of the assignment should be stopped.

Raises

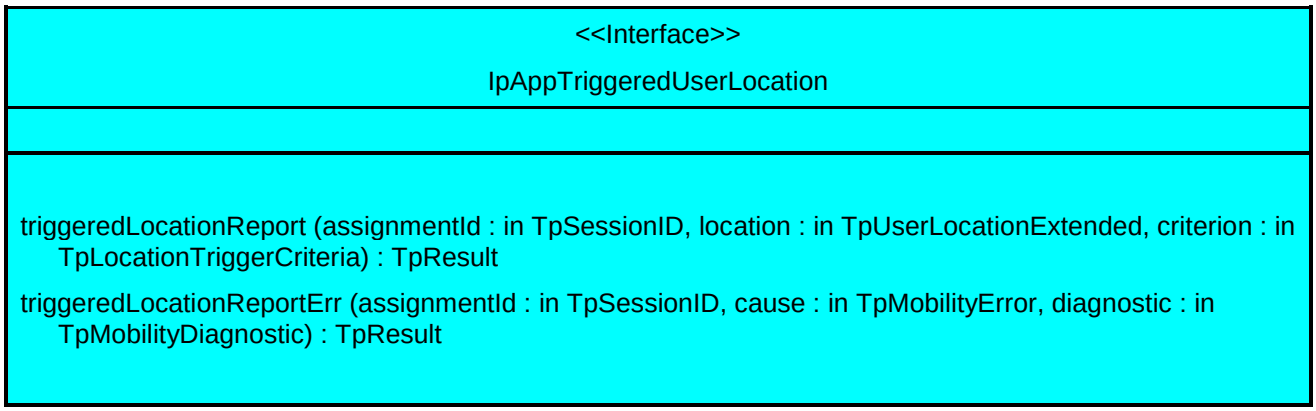
TpGeneralException

8.1.4 Interface Class IpAppTriggeredUserLocation

Inherits from: IpAppUserLocation.

This interface must be used as a specialised version of the User Location: Application Interface if the Triggered User Location: Service Interface is used.

The triggered user location application interface is implemented by the client application developer and is used to handle triggered location reports.



Method

triggeredLocationReport()

A triggered report containing location for a user is delivered.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the triggered location-reporting request.

location : in TpUserLocationExtended

Specifies the location of the user.

criterion : in TpLocationTriggerCriteria

Specifies the criterion that triggered the report.

Raises

TpGeneralException

Method

triggeredLocationReportErr()

This method indicates that a requested triggered location report has failed. Note that errors only concerning individual users are reported in the ordinary triggeredLocationReport() message.

Parameters

`assignmentId : in TpSessionID`

Specifies the assignment ID of the failed triggered location reporting start request.

`cause : in TpMobilityError`

Specifies the error that led to the failure.

`diagnostic : in TpMobilityDiagnostic`

Specifies additional information about the error that led to the failure.

8.2 User Location Camel Interface Classes

The ULC provides location information, based on network-related information, rather than the geographical co-ordinates that can be retrieved via the general User Location Service.

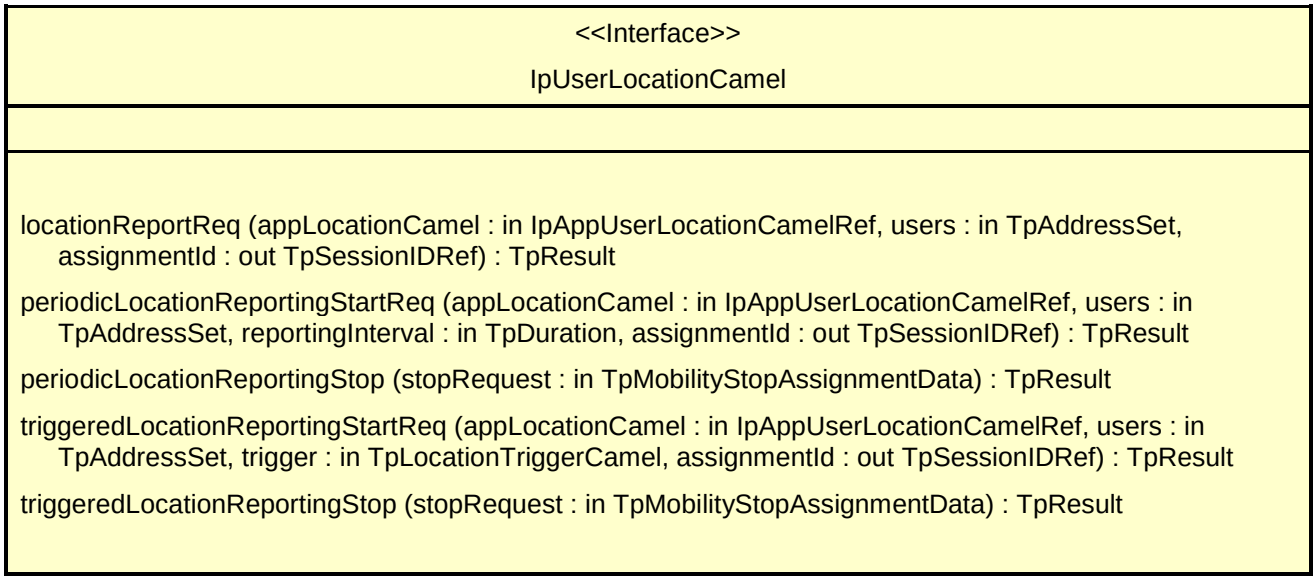
Using the ULC functions, an application programmer can request the VLR Number, the location Area Identification and the Cell Global Identification and other mobile-telephony-specific location information

The ULC provides the IpUserLocationCamel interface. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppUserLocationCamel interface to provide the callback mechanism.

8.2.1 Interface Class IpUserLocationCamel

Inherits from: IpService.

This interface is the 'service manager' interface for ULC.



Method

locationReportReq()

Request for mobile-related location information on one or several camel users.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

P_RESOURCES_UNAVAILABLE

The required resources in the network are not available. The application may try to invoke the method at a later time.

P_USER_NOT_SUBSCRIBED

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_USER_PRIVACY

The requests violates the end-user's privacy setting.

Parameters

appLocationCamel : in IpAppUserLocationCamelRef

Specifies the application interface for callbacks from the User Location Camel service.

users : in TpAddressSet

Specifies the user(s) for which the location shall be reported.

assignmentId : out TpSessionIDRef

Specifies the assignment ID of the location-report request.

Raises

TpGeneralException

Method

periodicLocationReportingStartReq()

Request for periodic mobile location reports on one or several users.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

P_RESOURCES_UNAVAILABLE

The required resources in the network are not available. The application may try to invoke the method at a later time.

P_USER_NOT_SUBSCRIBED

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_USER_PRIVACY

The requests violates the end-user's privacy setting.

Parameters

appLocationCamel : in IpAppUserLocationCamelRef

Specifies the application interface for callbacks from the User Location Camel service.

users : in TpAddressSet

Specifies the user(s) for which the location shall be reported.

reportingInterval : in TpDuration

Specifies the requested interval in seconds between the reports.

assignmentId : out TpSessionIDRef

Specifies the assignment ID of the periodic location-reporting request.

Raises

TpGeneralException

Method

periodicLocationReportingStop()

This method stops the sending of periodic mobile location reports for one or several users.

Raises the following exceptions:

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

stopRequest : in TpMobilityStopAssignmentData

Specifies how the assignment shall be stopped, i.e. if whole or just parts of the assignment should be stopped.

Raises

TpGeneralException

Method

triggeredLocationReportingStartReq()

Request for user location reports, containing mobile related information, when the location is changed (the report is triggered by the location change).

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

P_RESOURCES_UNAVAILABLE

The required resources in the network are not available. The application may try to invoke the method at a later time.

P_USER_NOT_SUBSCRIBED

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_USER_PRIVACY

The requests violates the end-user's privacy setting.

Parameters

appLocationCamel : in IpAppUserLocationCamelRef

Specifies the application interface for callbacks from the User Location Camel service.

users : in TpAddressSet

Specifies the user(s) for which the location shall be reported.

trigger : in TpLocationTriggerCamel

Specifies the trigger conditions.

assignmentId : out TpSessionIDRef

Specifies the assignment ID of the triggered location-reporting request.

Raises

TpGeneralException

Method

triggeredLocationReportingStop()

Request that triggered mobile location reporting should stop.

Raises the following exceptions:

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

stopRequest : in TpMobilityStopAssignmentData

Specifies how the assignment shall be stopped, i.e. if whole or just parts of the assignment should be stopped.

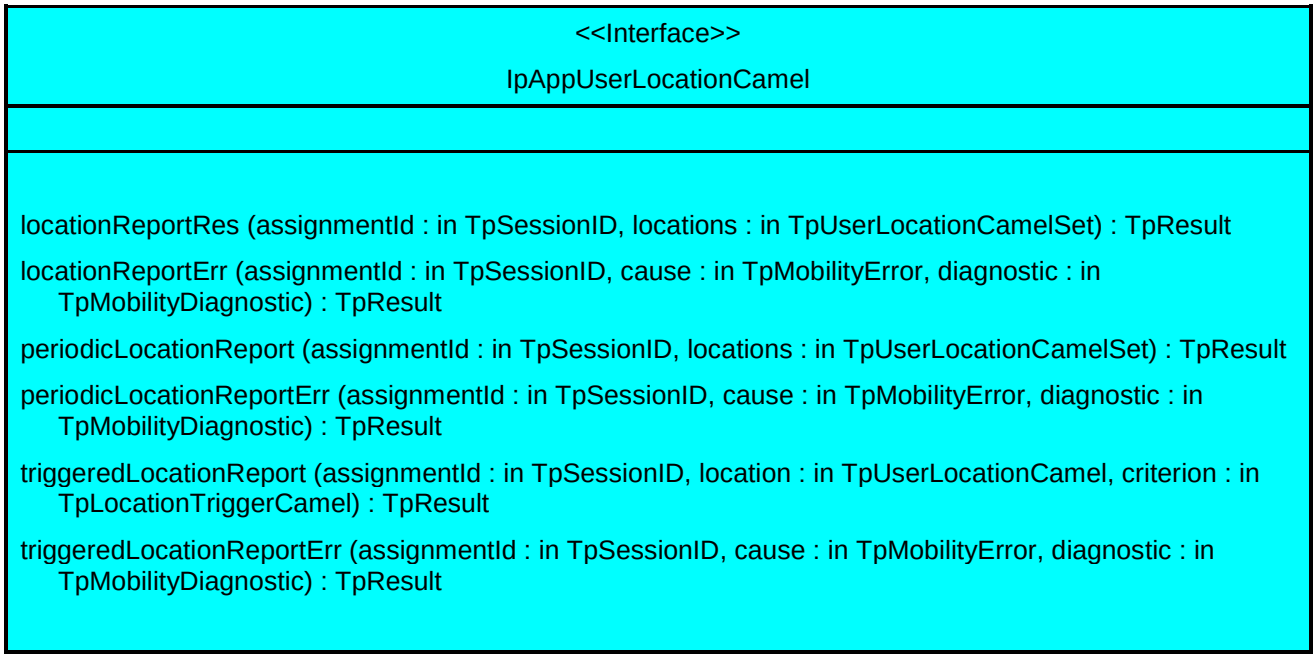
Raises

TpGeneralException

8.2.2 Interface Class IpAppUserLocationCamel

Inherits from: IpInterface.

The user location Camel application interface is implemented by the client application developer and is used to handle location reports that are specific for mobile telephony users.



Method

locationReportRes()

Delivery of a mobile location report. The report is containing mobile-related location information for one or several users.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the location-report request.

locations : in TpUserLocationCamelSet

Specifies the location(s) of one or several users.

Raises

TpGeneralException

Method

locationReportErr()

This method indicates that the location report request has failed.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the failed location report request.

cause : in TpMobilityError

Specifies the error that led to the failure.

diagnostic : in TpMobilityDiagnostic

Specifies additional information about the error that led to the failure.

Method

periodicLocationReport()

Periodic delivery of mobile location reports. The reports are containing mobile-related location information for one or several users.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the periodic location-reporting request.

locations : in TpUserLocationCamelSet

Specifies the location(s) of one or several users.

Raises

TpGeneralException

Method

periodicLocationReportErr()

This method indicates that a requested periodic location report has failed. Note that errors only concerning individual users are reported in the ordinary periodicLocationReport() message.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the failed periodic location reporting start request.

cause : in TpMobilityError

Specifies the error that led to the failure.

diagnostic : in TpMobilityDiagnostic

Specifies additional information about the error that led to the failure.

Method

triggeredLocationReport()

Delivery of a report that is indicating that the user's mobile location has changed.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the triggered location-reporting request.

location : in TpUserLocationCamel

Specifies the location of the user.

criterion : in TpLocationTriggerCamel

Specifies the criterion that triggered the report.

Raises

TpGeneralException

Method

triggeredLocationReportErr()

This method indicates that a requested triggered location report has failed. Note that errors only concerning individual users are reported in the ordinary triggeredLocationReport() message.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the failed triggered location reporting start request.

`cause : in TpMobilityError`

Specifies the error that led to the failure.

`diagnostic : in TpMobilityDiagnostic`

Specifies additional information about the error that led to the failure.

8.3 User Status Interface Classes

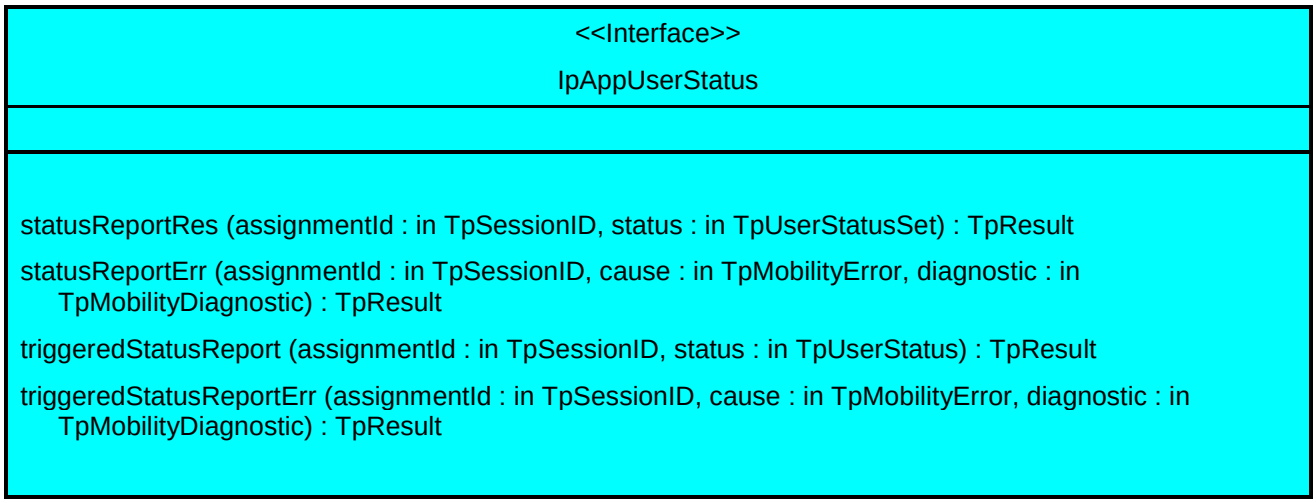
The User Status Service (US) provides a general user status service. US allow applications to obtain the status of fixed, mobile and IP-based telephony users.

The US provides the IpUserStatus interface. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppUserStatus interface to provide the callback mechanism.

8.3.1 Interface Class IpAppUserStatus

Inherits from: IpInterface.

The user-status application interface is implemented by the client application developer and is used to handle user status reports.



Method

`statusReportRes()`

Delivery of a report, that is containing one or several user's status.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the status-report request.

status : in TpUserStatusSet

Specifies the status of one or several users.

Raises

TpGeneralException

Method

statusReportErr()

This method indicates that the status report request has failed.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the failed status report request.

cause : in TpMobilityError

Specifies the error that led to the failure.

diagnostic : in TpMobilityDiagnostic

Specifies additional information about the error that led to the failure.

Method

triggeredStatusReport()

Delivery of a report that is indicating that a user's status has changed.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the triggered status-reporting request.

status : in TpUserStatus

Specifies the status of the user.

Raises

TpGeneralException

Method
triggeredStatusReportErr()

This method indicates that a requested triggered status reporting has failed. Note that errors only concerning individual users are reported in the ordinary triggeredStatusReport() message.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the failed triggered status reporting start request.

cause : in TpMobilityError

Specifies the error that led to the failure.

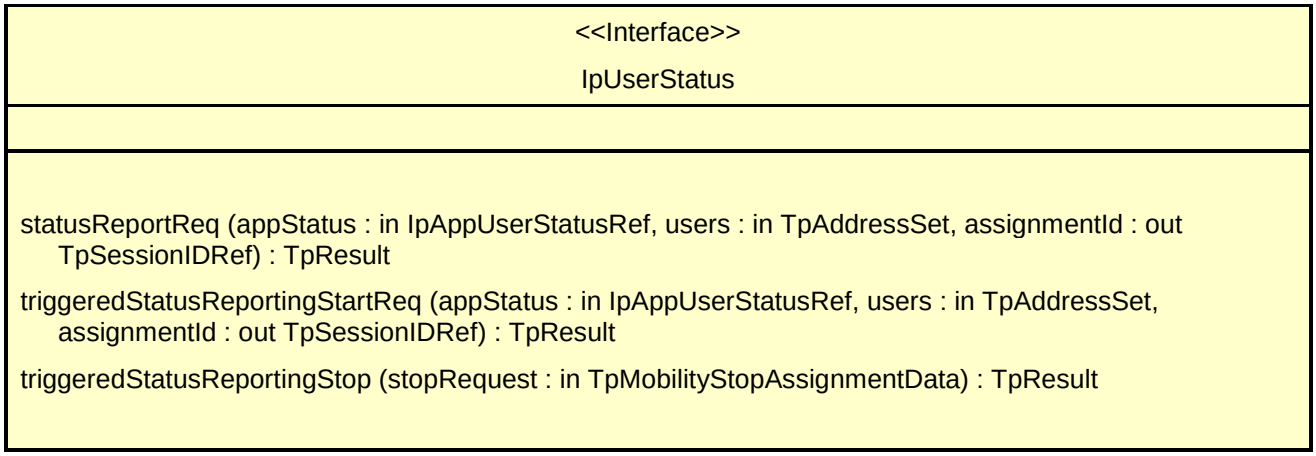
diagnostic : in TpMobilityDiagnostic

Specifies additional information about the error that led to the failure.

8.3.2 Interface Class IpUserStatus

Inherits from: IpService.

The application programmer can use this interface to obtain the status of fixed, mobile and IP-based telephony users.



Method
statusReportReq()

Request for a report on the status of one or several users.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

P_RESOURCES_UNAVAILABLE

The required resources in the network are not available. The application may try to invoke the method at a later time.

Parameters

appStatus : in IpAppUserStatusRef

Specifies the application interface for callbacks from the User Status service.

users : in TpAddressSet

Specifies the user(s) for which the status shall be reported.

assignmentId : out TpSessionIDRef

Specifies the assignment ID of the status-report request.

Raises

TpGeneralException

Method

triggeredStatusReportingStartReq()

Request for triggered status reports when one or several user's status is changed. The user status service will send a report when the status changes.

Raises the following exceptions:

P_INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

P_NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

P_RESOURCES_UNAVAILABLE

The required resources in the network are not available. The application may try to invoke the method at a later time.

Parameters

appStatus : in IpAppUserStatusRef

Specifies the application interface for callbacks from the User Status service.

users : in TpAddressSet

Specifies the user(s) for which the status changes shall be reported.

assignmentId : out TpSessionIDRef

Specifies the assignment ID of the triggered status-reporting request.

Raises

TpGeneralException

Method

triggeredStatusReportingStop()

This method stops the sending of status reports for one or several users.

Raises the following exceptions:

P_INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Parameters

`stopRequest : in TpMobilityStopAssignmentData`

Specifies how the assignment shall be stopped, i.e. if whole or just parts of the assignment should be stopped.

Raises

`TpGeneralException`

9 State Transition Diagrams

9.1 User Location

There are no State Transition Diagrams for User Location.

9.2 User Location Camel

9.2.1 State Transition Diagrams for IpUserLocationCamel

During the signServiceAgreement a new user location interface reference is created, which is user as the initial point of contact for the application.

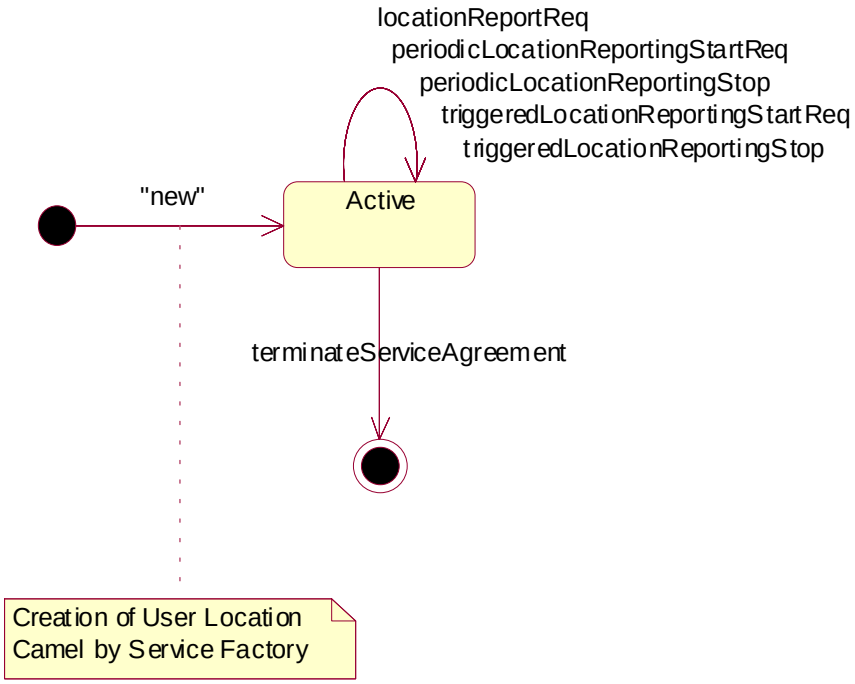


Figure : State Transition Diagram for User Location Camel

9.2.1.1 Active State

In this state, a relation between the Application and the Network User Location Service Capability Feature has been established. It allows the application to request a specific user location reports, subscribe to periodic user location reports or subscribe to triggers that generate location report when a location update occurs inside the current VLR area or when the user moves to another VLR area or both.

9.3 User Status

9.3.1 State Transition Diagrams for IpUserStatus

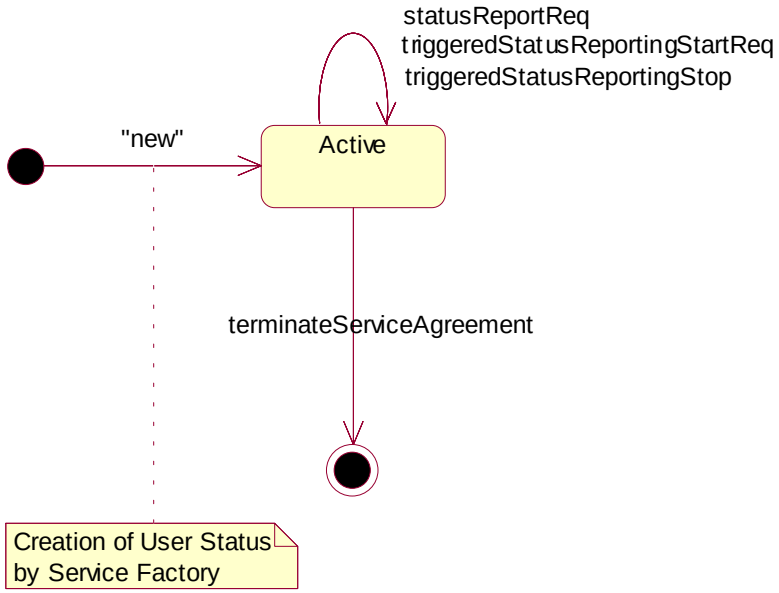


Figure : State Transition Diagram for User Status

9.3.1.1 Active State

In this state, a relation between the Application and the User Status Service Capability Feature has been established. It allows the application to request a specific user status report or subscribe to triggers that generate status reports when the status of one of the monitored user changes.

10 Service Properties

10.1 Mobility Properties

10.1.1 Emergency[1] Application Subtypes

This property contains a list of application subtypes that are permitted to use the service. The possible subtypes are (see definition of 'LCS Client Internal ID' in GSM 09.02 and chapter 6.4.1 in GSM 03.71):

[1] See definition of 'LCS Client Type' in GSM 09.02.

- “Broadcast service”
- “O&M HPLMN service”
- “O&M VPLMN service”
- “Anonymous location”
- “Target MS subscribed service”

10.1.2 Value Added[2] Application Subtypes

This property contains a list of application subtypes that are permitted to use the service. The possible subtypes are (see definition of ‘LCS Client Internal ID’ in GSM 09.02 and chapter 6.4.1 in GSM 03.71):

- “Broadcast service”
- “O&M HPLMN service”
- “O&M VPLMN service”
- “Anonymous location”
- “Target MS subscribed service”

10.1.3 PLMN Operator[3] Application Subtypes

This property contains a list of application subtypes that are permitted to use the service. The possible subtypes are (see definition of ‘LCS Client Internal ID’ in GSM 09.02 and chapter 6.4.1 in GSM 03.71):

- “Broadcast service”
- “O&M HPLMN service”
- “O&M VPLMN service”
- “Anonymous location”
- “Target MS subscribed service”

10.1.4 Lawful Intercept[4] Application Subtypes

This property contains a list of application subtypes that are permitted to use the service. The possible subtypes are (see definition of ‘LCS Client Internal ID’ in GSM 09.02 and chapter 6.4.1 in GSM 03.71):

- “Broadcast service”
- “O&M HPLMN service”
- “O&M VPLMN service”
- “Anonymous location”
- “Target MS subscribed service”

10.1.5 Altitude Obtainable

Indicates whether it is possible to obtain a user’s altitude.

10.1.6 Location Methods

List of supported location methods. Possible values (other values are permitted):

- “Time of Arrival”
- “Timing Advance”
- “GPS”

---

2 See definition of ‘LCS Client Type’ in GSM 09.02.

3 See definition of ‘LCS Client Type’ in GSM 09.02.

4 See definition of ‘LCS Client Type’ in GSM 09.02.

- "User Data Lookup"
- "Any Time Interrogation"

10.1.7 Priorities

List of supported priorities for location requests. Possible values (no other values are permitted):

- "Normal"
- "High"

10.1.8 Max Interactive Requests

The maximum number of parallel outstanding location or status requests allowed per application. It shall be possible to convert the value to a 32-bit integer.

10.1.9 Max Triggered Users

The maximum number of users allowed per application for which triggered location reporting can be requested. It shall be possible to convert the value to a 32-bit integer.

10.1.10 Max Periodic Users

The maximum number of users allowed per application for which periodic location reporting can be requested. It shall be possible to convert the value to a 32-bit integer.

10.1.11 Min Periodic Interval Duration

The minimal time in seconds allowed between two periodic reports. It shall be possible to convert the value to a 32-bit integer.

10.2 User Location Service Properties

A specific User Location service shall set the following properties:

- General Properties applicable to all SCFs (in Framework)
- Permitted application types
- Permitted application subtypes
- Priorities[5]
- Altitude obtainable
- Location methods
- Max interactive requests
- Max triggered users
- Max periodic users
- Min periodic interval duration

[5] See definition of 'LCSClientType' in GSM 09.02.

Example

The example below describes the capabilities of two fictive User Location services:

Property Name	Property Value Service 1	Property Value Service 2
Service instance ID	0x80923AD0	0xF0ED85CB
Service name	UserLocation	UserLocation
Service version	2.1	2.1
Service description	Basic user location service.	Advanced high-performance user location service.
Product name	Find It	Locate.com
Product version	1.3	3.1
Supported interfaces	“IpUserLocation”	“IpUserLocation”
Permitted application types	“Emergency service”, “Value added service”	“Emergency service”, “Value added service”, “Lawful intercept service”
Permitted application subtypes	?	?
Priorities	“Normal”	“Normal”, “High”
Altitude obtainable	False	True
Location methods	“Timing Advance”	“GPS”, “Time Of Arrival”
Max interactive requests	2000	10000
Max triggered users	0	2000
Max periodic users	300	2000
Min periodic interval duration	600	30

10.3 User Location Camel Service Properties

A specific User Location Camel service shall set the following properties:

- General Properties applicable to all SCFs (in Framework)
- Max interactive requests
- Max triggered users
- Max periodic users
- Min periodic interval duration

10.4 User Status Service Properties

A specific User Location service shall set the following properties:

- General Properties applicable to all SCFs (in Framework)
- Max interactive requests
- Max triggered users

11 Data Definitions

11.1 Common Mobility Data Definitions

The following data definitions are used for several of the mobility services.

11.1.1 TpGeographicalPosition

TpGeographicalPosition

Defines the `Sequence of Data Elements` that specify a geographical position.

The horizontal location is defined by an "ellipsoid point with uncertainty shape". The reference system chosen for the coding of locations is the World Geodetic System 1984 (WGS 84).

TypeOfUncertaintyShape describes the type of the uncertainty shape and *Longitude*/*Latitude* defines the position of the uncertainty shape. The following table defines the meaning of the data elements that describe the uncertainty shape for each uncertainty shape type.

Type of uncertainty shape	*Uncertainty Outer Semi Major*	*Uncertainty Outer Semi Minor*	*Uncertainty Inner Semi Major*	*Uncertainty Inner Semi Minor*	*Angle Of Semi Major*	*Segment Start Angle*	*Segment End Angle*
None	-	-	-	-	-	-	-
Circle	radius of circle	-	-	-	-	-	-
Circle Sector	radius of circle	-	-	-	-	start angle of circle segment	end angle of circle segment
Circle Arc Stripe	radius of outer circle	-	radius of inner circle	-	-	start angle of circle arc stripe	end angle of circle arc stripe
Ellipse	length of semi-major axis	length of semi-minor axis	-	-	rotation of ellipse measured clockwise from north	-	-
Ellipse Sector	length of semi-major axis	length of semi-minor axis	-	-	rotation of ellipse measured clockwise from north	start angle of ellipse segment	end angle of ellipse segment
Ellipse Arc Stripe	length of semi-major axis, outer ellipse	length of semi-minor axis, outer ellipse	length of semi-major axis, inner ellipse	length of semi-minor axis, inner ellipse	rotation of ellipse measured clockwise from north	start angle of ellipse arc stripe	end angle of ellipse arc stripe

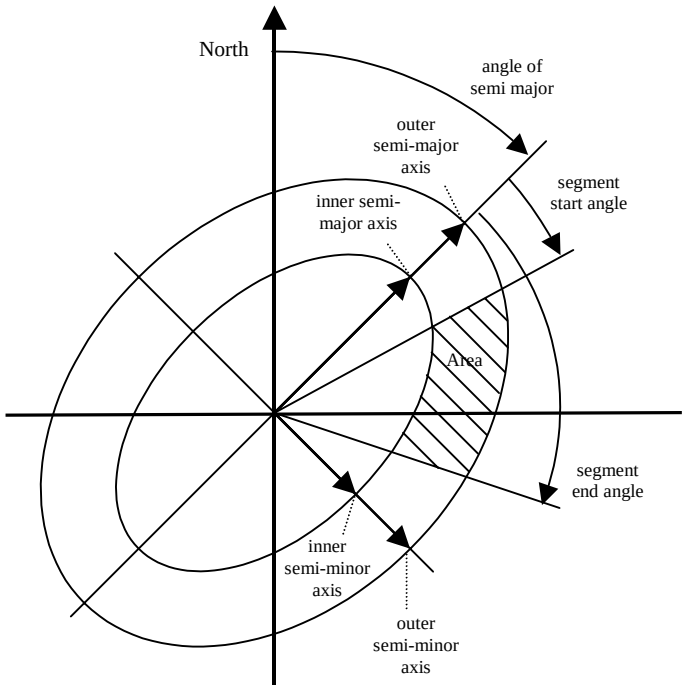


Figure 4 Description of an Ellipse Arc

TpGeographicalPosition:

Sequence Element Name	Sequence Element Type
Longitude	TpFloat
Latitude	TpFloat
TypeOfUncertaintyShape	TpLocationUncertaintyShape
UncertaintyInnerSemiMajor	TpFloat
UncertaintyOuterSemiMajor	TpFloat
UncertaintyInnerSemiMinor	TpFloat
UncertaintyOuterSemiMinor	TpFloat
AngleOfSemiMajor	TpInt32
SegmentStartAngle	TpInt32
SegmentEndAngle	TpInt32

11.1.2 TpLocationPriority

TpLocationPriority

Defines the priority of a location request.

Name	Value	Description
P_M_NORMAL	0	Normal
P_M_HIGH	1	High

11.1.3 TpLocationRequest

TpLocationRequest

Defines the Sequence of Data Elements that specify a location request.

Sequence Element Name	Sequence Element Type	Description
RequestedAccuracy	TpFloat	Requested accuracy in meters.
RequestedResponseTime	TpLocationResponseTime	Requested response time as a classified reqirement or as an absolute timer.
AltitudeRequested	TpBoolean	Altitude request flag.
Type	TpLocationType	The kind of location that is requested.
Priority	TpLocationPriority	Priority of location request.
RequestedLocationMethod	TpString	The kind of location method that is requested.

11.1.4 TpLocationResponseIndicator

TpLocationResponseIndicator

Defines a response time requirement.

Name	Value	Description
P_M_NO_DELAY	0	No delay: return either initial or last known location of the user.
P_M_LOW_DELAY	1	Low delay: return the current location with minimum delay. The mobility service shall attempt to fulfil any accuracy requirement, but in doing so shall not add any additional delay.
P_M_DELAY_TOLERANT	2	Delay tolerant: obtain the current location with regard to fulfilling the accuracy requirement.
P_M_USE_TIMER_VALUE	3	Use timer value: obtain the current location with regard to fulfilling the response time requirement.

11.1.5 TpLocationResponseTime

TpLocationResponseTime

Defines the Sequence of Data Elements that specify the application's requirements on the mobility service's response time.

Sequence Element Name	Sequence Element Type	Description
ResponseTime	TpLocationResponseIndicator	Indicator for which kind of response time that is required, see TpLocationResponseIndicator.
TimerValue	TpInt32	Optional timer used in combination when ResponseTime equals P_M_USE_TIMER_VALUE.

11.1.6 TpLocationType

TpLocationType

Defines the type of location requested.

Name	Value	Description
P_M_CURRENT	0	Current location
P_M_CURRENT_OR_LAST_KNOWN	1	Current or last known location
P_M_INITIAL	2	Initial location for an emergency services call

11.1.7 TpLocationUncertaintyShape

TpLocationUncertaintyShape

Defines the type of uncertainty shape.

Name	Value	Description
P_M_SHAPE_NONE	0	No uncertainty shape present.
P_M_SHAPE_CIRCLE	1	Uncertainty shape is a circle.
P_M_SHAPE_CIRCLE_SECTOR	2	Uncertainty shape is a circle sector.
P_M_SHAPE_CIRCLE_ARC_STRIPE	3	Uncertainty shape is a circle arc stripe.
P_M_SHAPE_ELLIPSE	4	Uncertainty shape is an ellipse.
P_M_SHAPE_ELLIPSE_SECTOR	5	Uncertainty shape is an ellipse sector.
P_M_SHAPE_ELLIPSE_ARC_STRIPE	6	Uncertainty shape is an ellipse arc stripe.

11.1.8 TpMobilityDiagnostic

TpMobilityDiagnostic

Defines a diagnostic value that is reported in addition to an error by one of the mobility services.

Name	Value	Description
P_M_NO_INFORMATION	0	No diagnostic information present. Valid for all type of errors.
P_M_APPL_NOT_IN_PRIV_EXCEPT_LST	1	Application not in privacy exception list. Valid for 'Unauthorised Application' error.
P_M_CALL_TO_USER_NOT_SETUP	2	Call to user not set-up. Valid for 'Unauthorised Application' error.
P_M_PRIVACY_OVERRIDE_NOT_APPLIC	3	Privacy override not applicable. Valid for 'Unauthorised Application' error.
P_M_DISALL_BY_LOCAL_REGULAT_REQ	4	Disallowed by local regulatory requirements. Valid for 'Unauthorised Application' error.
P_M_CONGESTION	5	Congestion. Valid for 'Position Method Failure' error.
P_M_INSUFFICIENT_RESOURCES	6	Insufficient resources. Valid for 'Position Method Failure' error.
P_M_INSUFFICIENT_MEAS_DATA	7	Insufficient measurement data. Valid for 'Position Method Failure' error.
P_M_INCONSISTENT_MEAS_DATA	8	Inconsistent measurement data. Valid for 'Position Method Failure' error.
P_M_LOC_PROC_NOT_COMPLETED	9	Location procedure not completed. Valid for 'Position Method Failure' error.
P_M_LOC_PROC_NOT_SUPP_BY_USER	10	Location procedure not supported by user. Valid for 'Position Method Failure' error.
P_M_QOS_NOT_ATTAINABLE	11	Quality of service not attainable. Valid for 'Position Method Failure' error.

11.1.9 TpMobilityError

TpMobilityError

Defines an error that is reported by one of the mobility services.

Name	Value	Description	Fatal
P_M_OK	0	No error occurred while processing the request.	N/A
P_M_SYSTEM_FAILURE	1	System failure. The request can not be handled because of a general problem in the mobility service or the underlying network.	Yes
P_M_UNAUTHORIZED_NETWORK	2	Unauthorised network, The requesting network is not authorised to obtain the user's location or status.	No
P_M_UNAUTHORIZED_APPLICATION	3	Unauthorised application. The application is not authorised to obtain the user's location or status.	Yes
P_M_UNKNOWN_SUBSCRIBER	4	Unknown subscriber. The user is unknown, i.e. no such subscription exists.	Yes
P_M_ABSENT_SUBSCRIBER	5	Absent subscriber. The user is currently not reachable.	No
P_M_POSITION_METHOD_FAILURE	6	Position method failure. The mobility service failed to obtain the user's position.	No

11.1.10 TpMobilityStopAssignmentData

TpMobilityStopAssignmentData

Defines the Sequence of Data Elements that specify a request to stop whole or parts of an assignment. Assignments are used for periodic or triggered reporting of a user's location or status.

Note that the parameter 'Users' is optional. If the parameter 'StopScope' is set to P_M_ALL_IN_ASSIGNMENT the parameter 'Users' is undefined. If the parameter 'StopScope' is set to P_M_SPECIFIED_USERS, then the assignment shall be stopped only for those users specified in the 'Users' list.

Sequence Element Name	Sequence Element Type	Description
AssignmentId	TpSessionID	Identity of the session that shall be stopped.
StopScope	TpMobilityStopScope	Specify if only a part of the assignment or if all the assignment shall be stopped.
Users	TpAddressSet	Optional parameter describing which users a stop request is addressing, when only a part of an assignment is to be stopped.

11.1.11 TpMobilityStopScope

TpMobilityStopScope

This enumeration is used in requests to stop mobility reports that are sent from a mobility service to an application.

Name	Value	Description
P_M_ALL_IN_ASSIGNMENT	0	The request concerns all users in an assignment.
P_M_SPECIFIED_USERS	1	The request concerns only the users that are explicitly specified in a list.

11.1.12 TpTerminalType

TpTerminalType

Defines which kind of terminal is used.

Name	Value	Description
P_M_FIXED	0	Fixed terminal.
P_M_MOBILE	1	Mobile terminal.
P_M_IP	2	IP terminal.

11.2 User Location Data Definitions

11.2.1 TpUlExtendedData

TpUlExtendedData

Defines the Sequence of Data Elements that specify a location (extended format).

The optional vertical location is defined by the data element *Altitude*, which contains the altitude in meters above sea level, and the data element *AltitudeAccuracy*, which contains the accuracy of the altitude.

Sequence Element Name	Sequence Element Type	Description
GeographicalPosition	TpGeographicalPosition	Specification of a position and an area of uncertainty.
TerminalType	TpTerminalType	Kind of terminal.
AltitudePresent	TpBoolean	Flag indicating if the altitude is present.
Altitude	TpFloat	Decimal altitude in meters.
UncertaintyAltitude	TpFloat	Uncertainty of the altitude.
TimestampPresent	TpBoolean	Flag indicating if the timestamp is present.
Timestamp	TpDateAndTime	Timestamp indicating when the position was measured.
UsedLocationMethod	TpString	Specifying which location method was used.

11.2.2 TpUlExtendedDataSet

TpUlExtendedDataSet

Defines a Numbered Set of Data Elements of TpUlExtendedData.

11.2.3 TpUserLocationExtended

TpUserLocationExtended

Defines the Sequence of Data Elements that specify the identity and location(s) of a user (extended format). In general the data element *Locations* will contain only one location, but in case of IP-telephony users this data element might continue several locations (the locations of all communication end-points, where the user is currently registered).

Sequence Element Name	Sequence Element Type	Description
UserID	TpAddress	The address of the user.
StatusCode	TpMobilityError	Indicator of error.

Locations	TpUlExtendedDataSet	Optional list of locations. If StatusCode is indicating an error, this value is undefined.

11.2.4 TpUserLocationExtendedSet

TpUserLocationExtendedSet

Defines a Numbered Set of Data Elements of TpUserLocationExtended.

11.2.5 TpLocationTrigger

TpLocationTrigger

Defines the Sequence of Data Elements that specify the criteria for a triggered location report to be generated. The area is defined by an ellipse.

Sequence Element Name	Sequence Element Type	Description
Longitude	TpFloat	Longitude of the position used in the trigger.
Latitude	TpFloat	Latitude of the position used in the trigger.
AreaSemiMajor	TpFloat	Semi major of ellipse area used in the trigger.
AreaSemiMinor	TpFloat	Semi minor of ellipse area used in the trigger.
AngleOfSemiMajor	TpInt32	Angle of the semi major of the ellipse area used in the trigger.
Criterion	TpLocationTriggerCriteria	Trigger criteria with regard to the ellipse area.
ReportingInterval	TpDuration	Duration between generated location reports.

11.2.6 TpLocationTriggerSet

TpLocationTriggerSet

Defines a Numbered Set of Data Elements of TpLocationTrigger

11.2.7 TpLocationTriggerCriteria

TpLocationTriggerCriteria

Defines the criteria that trigger a location report.

Name	Value	Description
P_UL_ENTERING_AREA	0	User enters the area
P_UL_LEAVING_AREA	1	User leaves the area

11.2.8 TpUserLocation

TpUserLocation

Defines the Sequence of Data Elements that specify the identity and location of a user (basic format).

Sequence Element Name	Sequence Element Type	Description
UserID	TpAddress	The address of the user.
StatusCode	TpMobilityError	Indicator of error.
GeographicalPosition	TpGeographicalPosition	Specification of a position and an area of uncertainty. If StatusCode is indicating an error, this value is undefined.

11.2.9 TpUserLocationSet

TpUserLocationSet

Defines a Numbered Set of Data Elements of TpUserLocation.

11.3 User Location Camel Data Definitions

11.3.1 TpLocationCellIDOrLAI

TpLocationCellIDOrLAI

This data type is identical to a TpString. It specifies the Cell Global Identification or the Location Area Identification (LAI).
The Cell Global Identification (CGI) is defined as a string of characters in the following format:

MCC-MNC-LAC-CI

where:

MCC Mobile Country Code (three decimal digits)

MNC Mobile Network Code (two or three decimal digits)

LAC Location Area Code (four hexadecimal digits)

CI Cell Identification (four hexadecimal digits)

The Location Area Identification (LAI) is defined as a string of characters in the following format:

MCC-MNC-LAC

where:

MCC Mobile Country Code (three decimal digits)

MNC Mobile Network Code (two or three decimal digits)

LAC Location Area Code (four hexadecimal digits)

The length of the parameter indicates, which format is used. See 3G TS 29.002 for the detailed coding.

11.3.2 TpLocationTriggerCamel

TpLocationTriggerCamel

Defines the Sequence of Data Elements that specify the criteria for a triggered location report to be generated.

Sequence Element Name	Sequence Element Type	Description
UpdateInsideVlr	TpBoolean	Generate location report, when a location update occurs inside the current VLR area.
UpdateOutsideVlr	TpBoolean	Generate location report, when the user moves to another VLR area.

11.3.3 TpUserLocationCamel

TpUserLocationCamel

Defines the Sequence of Data Elements that specify the location of a mobile telephony user. Note that if the StatusCode is indicating an error , then neither GeographicalPosition, Timestamp, VlrNumber, LocationNumber, CellIdOrLai nor their associated presence flags are defined.

Sequence Element Name	Sequence Element Type	Description
UserID	TpAddress	The address of the user.
StatusCode	TpMobilityError	Indicator of error.
GeographicalPositionPresent	TpBoolean	Flag indicating if the geographical position is present.
GeographicalPosition	TpGeographicalPosition	Specification of a position and an area of uncertainty.
TimestampPresent	TpBoolean	Flag indicating if the timestamp is present.
Timestamp	TpDateAndTime	Timestamp indicating when the request was processed.
VlrNumberPresent	TpBoolean	Flag indicating if the VLR number is present.
VlrNumber	TpAddress	Current VLR number for the user.
LocationNumberPresent	TpBoolean	Flag indicating if the location number is present.
LocationNumber[6]	TpAddress	Current location number.
CellIdOrLaiPresent	TpBoolean	Flag indicating if cell-id or LAI of the user is present.
CellIdOrLai	TpLocationCellIDOrLAI	Cell-id or LAI of the user.

11.3.4 TpUserLocationCamelSet

TpUserLocationCamelSet

Defines a Numbered Set of Data Elements of TpUserLocationCamel.

11.4 User Location Emergency Data Definitions

11.4.1 TpIMEI

TpIMEI

This data type is identical to a TpString. It specifies the International Mobile Equipment Identity (IMEI).

11.4.2 TpNaESRD

TpNaESRD

This data type is identical to a TpString. It specifies the North American Emergency Services Routing Digits (NA-ESRD).
NA-ESRD is a telephone number in the North American Numbering Plan that can be used to identify a North American emergency services provider and its associated Location Services client. The NA-ESRD also identifies the base station, cell site or sector from which a North American emergency call originates.

---

[6] The location number is the number to the MSC or in rare cases the roaming number.

11.4.3 TpNaESRK

TpNaESRK

This data type is identical to a `TpString`. It specifies the North American Emergency Services Routing Key (NA-ESRK).
NA-ESRK is a telephone number in the North American Numbering Plan that is assigned to an emergency services call for the duration of the call. The NA-ESRK is used to identify (e.g. route to) both, the emergency services provider and the switch, currently serving the emergency caller. During the lifetime of an emergency services call, the NA-ESRK also identifies the calling subscriber.

11.4.4 TpUserLocationEmergencyRequest

TpUserLocationEmergencyRequest

Defines the `Sequence of Data Elements` that specify the request for the location of an emergency service user. The emergency service user is identified by a combination of *user address*, *NaESRD*, *NaESRK* and *IMEI*. *NaESRD*, *NaESRK* and *IMEI* may be provided, if the emergency service user has originated the emergency service call in North America.

Sequence Element Name	Sequence Element Type	Description
UserAddressPresent	TpBoolean	Flag indicating if the user address is present.
UserAddress	TpAddress	The address of the user.
NaEsrdPresent	TpBoolean	Flag indicating if the NaESRD is present.
NaEsrd	TpNaESRD	Current NaESRD for the user.
NaEsrkPresent	TpBoolean	Flag indicating if the NaESRK is present.
NaEsrk	TpNaESRK	Current NaESRK for the user.
ImeiPresent	TpBoolean	Flag indicating if the IMEI is present.
Imei	TpIMEI	IMEI for the user.
LocationReq	TpLocationRequest	The actual location request.

11.4.5 TpUserLocationEmergency

TpUserLocationEmergency

Defines the `Sequence of Data Elements` that specify the identity and location of an emergency service user. The emergency service user is identified by a combination of *UserID*, *NaESRD*, *NaESRK* and *IMEI*.
NaESRD, *NaESRK* and *IMEI* may be provided, if the emergency service user has originated the emergency service call in North America.
The horizontal location is defined by an “ellipsoid point with uncertainty ellipse” (see TpUlExtendedData).

Sequence Element Name	Sequence Element Type	Description
StatusCode	TpMobilityError	Indicator of error.
UserIdPresent	TpBoolean	Flag indicating if the user address is present.
UserId	TpAddress	The user address.
NaEsrdPresent	TpBoolean	Flag indicating if the NaESRD is present.
NaEsrd	TpNaESRD	Current NaESRD for the user.
NaEsrkPresent	TpBoolean	Flag indicating if the NaESRK is present.
NaEsrk	TpNaESRK	Current NaESRK for the user.

ImeiPresent	TpBoolean	Flag indicating if the IMEI is present.
Imei	TpIMEI	IMEI for the user.
TriggeringEvent	TpUserLocationEmergencyTrigger	The reason for this location report.
GeographicalPositionPresent	TpBoolean	Flag indicating if the geographical position is present.
GeographicalPosition	TpGeographicalPosition	Specification of a position and an area of uncertainty.
AltitudePresent	TpBoolean	Flag indicating if the altitude is present.
Altitude	TpFloat	Decimal altitude in meters.
UncertaintyAltitude	TpFloat	Uncertainty of the altitude.
TimestampPresent	TpBoolean	Flag indicating if a timestamp is present.
Timestamp	TpDateAndTime	Timestamp indicating when the request was processed.
UsedLocationMethod	TpString	Specifying which location method was used.

11.4.6 TpUserLocationEmergencyTrigger

TpUserLocationEmergencyTrigger

Defines which event triggered the emergency user location report.

Name	Value	Description
P_ULE_CALL_ORIGINATION	0	An emergency service user originated an emergency call.
P_ULE_CALL_RELEASE	1	An emergency service user released an emergency call.
P_ULE_LOCATION_REQUEST	2	The report is a response to an emergency location report request.

11.5 User Status Data Definitions

11.5.1 TpUserStatus

TpUserStatus

Defines the Sequence of Data Elements that specify the identity and status of a user.

Sequence Element Name	Sequence Element Type	Description
UserID	TpAddress	The user address.
StatusCode	TpMobilityError	Indicator of error.
Status	TpUserStatusIndicator	The current status of the user.
TerminalType	TpTerminalType	The kind of terminal used by the user.

11.5.2 TpUserStatusSet

TpUserStatusSet

Defines a `Numbered Set of Data Elements` of TpUserStatus.

11.5.3 TpUserStatusIndicator

`TpUserStatusIndicator`

Defines the status of a user.

Name	Value	Description
`P_US_REACHABLE`	0	User is reachable
`P_US_NOT_REACHABLE`	1	User is not reachable
`P_US_BUSY`[7]	2	User is busy (only applicable for interactive user status request, not when triggers are used)

[7] Only applicable to mobile (Camel) telephony users.

11.6 Units and Validations of Parameters

This section describes the units that shall be used for data elements, where this is not obvious.

Altitude
Unit: Metric meter

Angle
Unit: Degrees

Value constraint: 0 ≤ 'Angle' ≤ 360

AreaSemiMajor and AreaSemiMinor
Unit: Metric meter

Value constraint: 0 ≤ 'AreaSemi…'

ReportingInterval
Unit: Seconds

Value constraint: 0 < 'ReportingInterval'

UncertaintyAltitude
Unit: Metric meter

Value constraint: 0 ≤ 'UncertaintyAltitude'

Semantic: (Altitude – UncertaintyAltitude) ≤ 'Terminal actual altitude' ≤ ('Altitude' + 'UncertaintyAltitude')

UncertaintyInnerSemiMajor and UncertaintyInnerSemiMinor
Unit: Metric meter

Value constraint: 0 ≤ 'UncertaintyInner…'

UncertaintyOuterSemiMajor and UncertaintyOuterSemiMinor
Unit: Metric meter

Value constraint: 0 ≤ 'UncertaintyInner…'

UsedLocationMethod
Predefined strings are listed in section Location Methods.

Annex A (normative): OMG IDL Description of Mobility SCF

The OMG IDL representation of this interface specification is contained in a text file (mm.idl contained in archive 2919806IDL.ZIP) which accompanies the present document.

Annex B (informative): Differences between this draft and 3GPP 29.198 R99

No differences recorded to methods, parameters or data types for those interfaces which are common (User Location Camel and User Status). User Location interfaces added.

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for approval and placement under Change Control

3GPP TS 29.198-7 V1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface;
Part 7: Terminal Capabilities;
(Release 4)

Keywords
API, OSA, IDL, TERMCAP, Terminal Capabilities

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

This document is part of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA). The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA API's. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

This document specifies the Terminal Capabilities Service Capability Feature (SCF) aspects of the interface. All aspects of the Terminal Capabilities SCF are defined here, these being:

- Sequence Diagrams
- Class Diagrams
- Interface specification plus detailed method descriptions
- State Transition diagrams
- Data definitions
- IDL Description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modelling Language (UML).

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

[1] 3GPP TS 29.198-1 "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

[4] World Wide Web Consortium Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation (www.w3.org)

[5] Wireless Application Protocol (WAP), Version 1.2, UAProf Specification (www.wapforum.org)

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in TS 29.198-1 [1] apply.

3.2 Symbols

For the purposes of the present document, the symbols in TS 29.198-1 [1] apply.

3.3 Abbreviations

For the purposes of the present document, the abbreviations in TS 29.198-1 [1] apply.

4 Terminal Capabilities SCF

The following sections describe each aspect of the Terminal Capability Feature (SCF).

The order is as follows:

- The Sequence diagrams give the reader a practical idea of how each of the service capability feature is implemented.
- The Class relationships section show how each of the interfaces applicable to the SCF, relate to one another
- The Interface specification section describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the progression of internal processes either in the application, or Gateway.
- The Data definitions section show a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.

5 Sequence Diagrams

There are no Sequence Diagrams for the Terminal Capabilities SCF.

6 Class Diagrams

Terminal Capabilities Class Diagram:

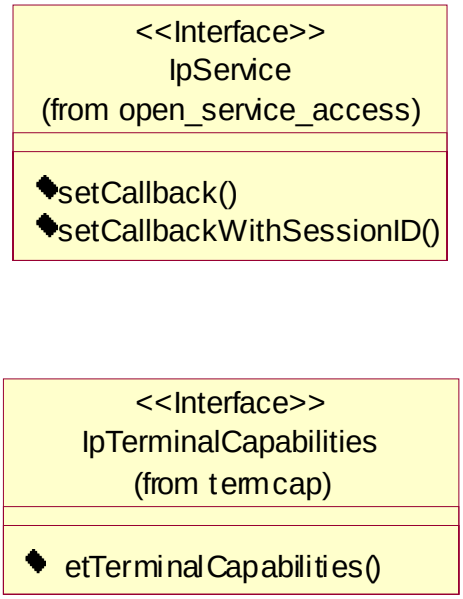


Figure: Package Overview

7 The Service Interface Specifications

7.1 Interface Specification Format

This section defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

7.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for enterprise-based client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>.

7.1.2 Method descriptions

Each method (API method "call") is described. All methods in the API return a value of type `TpResult`, indicating, amongst other things, if the method invocation was sucessfully executed or not.

Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a '`Req`' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a '`Res`' or '`Err`' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name> or IpSvc<name>` interfaces to provide the callback mechanism.

7.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as "in" represent those that must have a value when the method is called. Those described as "out" are those that contain the return result of the method when the method returns.

7.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

7.2 Base Interface

7.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.

<<Interface>> IpInterface

7.3 Service Interfaces

7.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

The interfaces that are implemented by the services are denoted as "Service Interface". The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as "Application Interface".

7.4 Generic Service Interface

7.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.

<<Interface>> IpService
setCallback (appInterface : in IpInterfaceRef) : TpResult setCallbackWithSessionID (appInterface : in IpInterfaceRef, sessionID : in TpSessionID) : TpResult

Method
`setCallback()`

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

Raises

TpGeneralException

Method

setCallbackWithSessionID()

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

sessionID : in TpSessionID

Specifies the session for which the service can invoke the application's callback interface.

Raises

TpGeneralException

8 Terminal Capabilities Interface Classes

The Terminal Capabilities SCF enables the application to retrieve the terminal capabilities of the specified terminal. The Terminal Capabilities service provides a SCF interface that is called IpTerminalCapabilities. There is no need for an application interface, since IpTerminalCapabilities only contains the synchronous method getTerminalCapabilities.

8.1 Interface Class IpTerminalCapabilities

Inherits from: IpInterface.

The Terminal Capabilities SCF interface IpTerminalCapabilities contains the synchronous method getTerminalCapabilities. The application has to provide the terminaIdentity as input to this method. The result indicates whether or not the terminal capabilities are available in the network and, in case they are, it will return the terminal capabilities (see the data definition of TpTerminalCapabilities for more information).

<<Interface>> IpTerminalCapabilities
getTerminalCapabilities (terminalIdentity : in TpString, result : out TpTerminalCapabilitiesRef) : TpResult

Method

`getTerminalCapabilities()`

This method is used by an application to get the capabilities of a user's terminal. Direction: Application to Network.

Parameters

`terminalIdentity : in TpString`

Identifies the terminal. It may be a logical address known by the WAP Gateway/PushProxy.

`result : out TpTerminalCapabilitiesRef`

Specifies the latest available capabilities of the user´s terminal.

This information, if available, is returned as CC/PP headers as specified in W3C [1] and adopted in the WAP UAProf specification [2]. It contains URLs; terminal attributes and values, in RDF format; or a combination of both.

Raises

`TpTermCapException,TpGeneralException`

9 State Transition Diagrams

There are no State Transition Diagrams for the Terminal Capabilities SCF.

10 Terminal Capabilities Data Definitions

The constants and types defined in the following sections are defined in the *org.osa.termcap* package.

`terminalIdentity`

Identifies the terminal.

Name	Type	Documentation
`terminalIdentity`	TpString	Identifies the terminal. It may be a logical address known by the WAP Gateway/PushProxy.

`TpTerminalCapabilities`

This data type is a Sequence_of_Data_Elements that describes the terminal capabilities. It is a structured type that consists of:

Sequence Element Name	Sequence Element Type	Documentation
`StatusCode`	TpBoolean	Indicates whether or not the terminalCapabilities are available.
`TerminalCapabilities`	TpString	Specifies the latest available capabilities of the user´s terminal. This information, if available, is returned as CC/PP headers as specified in W3C [12] and adopted in the WAP UAProf specification [13]. It contains URLs; terminal attributes and values, in RDF format; or a combination of both.

TpTerminalCapabilitiesError

Defines an error that is reported by the Terminal Capabilities SCF.

Name	Value	Description
P_TERMCAP_ERROR_UNDEFINED	0	Undefined.
P_TERMCAP_INVALID_TERMINALID	1	The request can not be handled because the terminal id specified is not valid.
P_TERMCAP_SYSTEM_FAILURE	2	System failure. The request cannot be handled because of a general problem in the terminal capabilities service or the underlying network.

Annex A (normative): OMG IDL Description of Terminal Capabilities SCF

The OMG IDL representation of this interface specification is contained in a text file (termcap.idl contained in archive 2919807IDL.ZIP) which accompanies the present document.

Annex B (informative):
Differences between this draft and 3GPP 29.198 R99

None Recorded

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for approval and placement under Change Control

3GPP TS 29.198-8 V1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface
Part 8: Data Session Control
(Release 4)

Keywords
API, OSA, IDL, DSC, Data Session Control

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

This document is part of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA). The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA API's. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

This document specifies the Data Session Control Service Capability Feature (SCF) aspects of the interface. All aspects of the Data Session Control SCF are defined here, these being:

- Sequence Diagrams
- Class Diagrams
- Interface specification plus detailed method descriptions
- State Transition diagrams
- Data definitions
- IDL Description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modeling Language (UML).

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

[1] 3GPP TS 29.198-1 "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in TS 29.198-1 [1] apply.

3.2 Symbols

For the purposes of the present document, the symbols in TS 29.198-1 [1] apply.

3.3 Abbreviations

For the purposes of the present document, the abbreviations in TS 29.198-1 [1] apply.

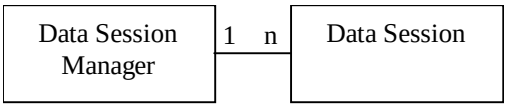
4 Data Session Control SCF

The Data Session control network service capability feature consists of two interfaces:

1) Data Session manager, containing management functions for data session related issues;

2) Data Session, containing methods to control a session.

A session can be controlled by one Data Session Manager only. Data Session Manager can control several sessions.



NOTE: The term "data session" is used in a broad sense to describe a data connection/session. For example, it comprises a PDP context in GPRS.

Figure 1: Data Session control interfaces usage relationship

The Data Session Control service capability features are described in terms of the methods in the Data Session Control interfaces. Table 1 gives an overview of the Data Session Control methods and to which interfaces these methods belong.

Table 1: Overview of Data Session Control interfaces and their methods

Data Session Manager	Data Session
createNotification	connectReq
destroyNotification	connectRes
dataSessionNotificationInterrupted	connectErr
dataSessionNotificationContinued	release
reportNotification	superviseDataSessionReq
dataSessionAborted	superviseDataSessionRes
getNotification	superviseDataSessionErr
changeNotification	dataSessionFaultDetected
	setAdviceofCharge
	setDataSessionChargePlan

The session manager interface provides the management functions to the data session service capability features. The application programmer can use this interface to enable or disable data session-related event notifications.

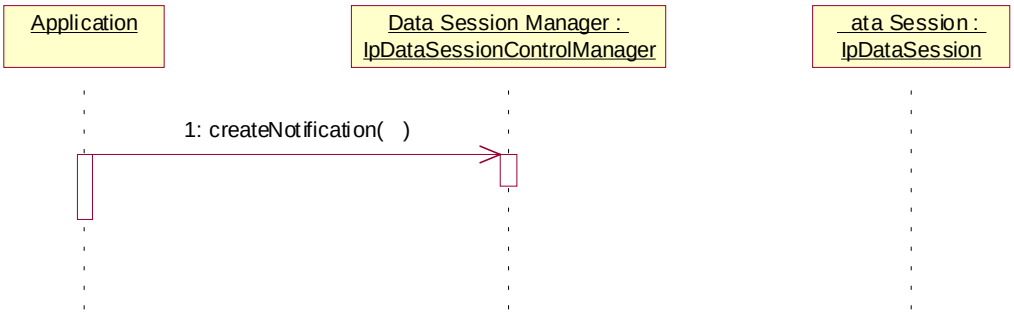
The following sections describe each aspect of the Data Session Control Service Capability Feature (SCF).

The order is as follows:

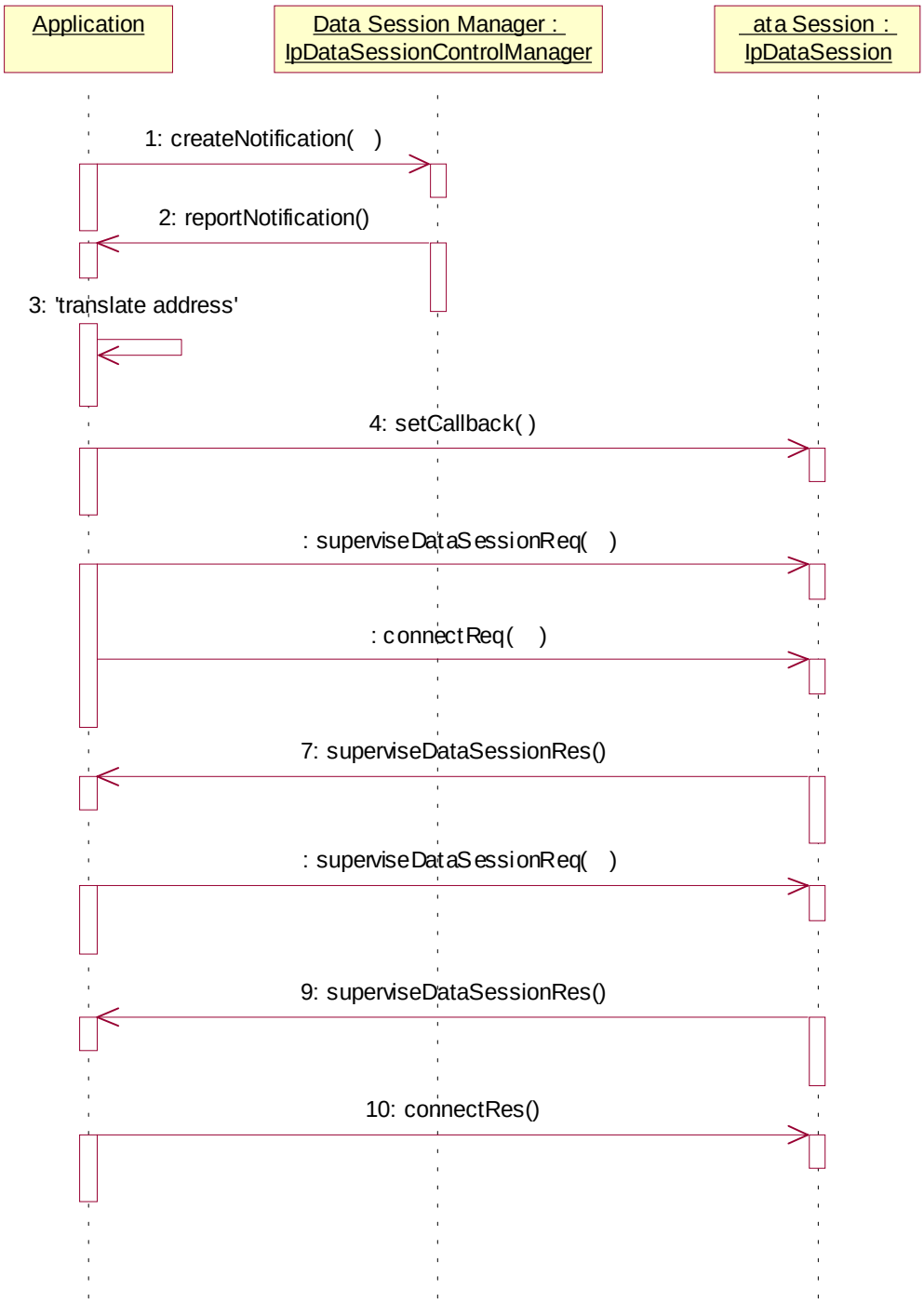
- the Sequence diagrams give the reader a practical idea of how each of the service capability feature is implemented;
- the Class relationships section show how each of the interfaces applicable to the SCF, relate to one another;
- the Interface specification section describes in detail each of the interfaces shown within the Class diagram part;
- the State Transition Diagrams (STD) show the progression of internal processes either in the application, or Gateway;
- the Data definitions section show a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.

5 Sequence Diagrams

5.1 Enable Data Session Notification



5.2 Address Translation With Charging


Application
Data Session Manager : IpDataSessionControlManager
ata Session : IpDataSession
1: createNotification( )
2: reportNotification()
3: 'translate address'
4: setCallback( )
: superviseDataSessionReq( )
: connectReq( )
7: superviseDataSessionRes()
: superviseDataSessionReq( )
9: superviseDataSessionRes()
10: connectRes()

6 Class Diagrams

Data Session Control Class Diagram:

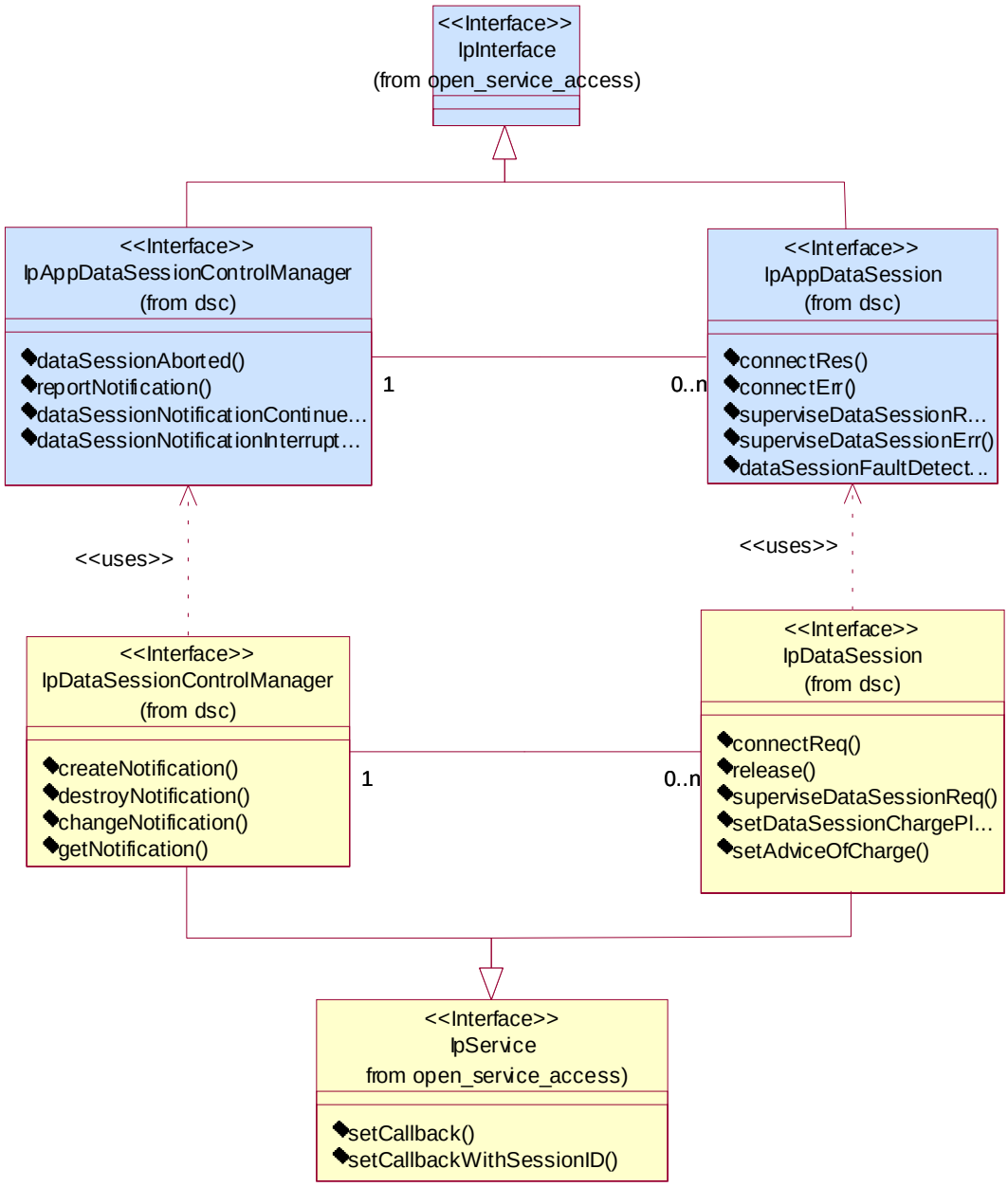


Figure: Package Overview

7 The Service Interface Specifications

7.1 Interface Specification Format

This section defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

7.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for enterprise-based client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>

7.1.2 Method descriptions

Each method (API method "call") is described. All methods in the API return a value of type `TpResult`, indicating, amongst other things, if the method invocation was sucessfully executed or not.

Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a '`Req`' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a '`Res`' or '`Err`' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name> or IpSvc<name>` interfaces to provide the callback mechanism.

7.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.

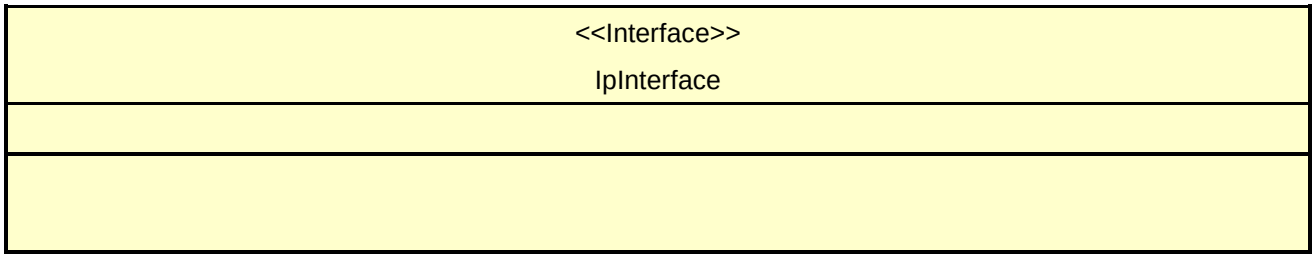
7.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

7.2 Base Interface

7.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.



7.3 Service Interfaces

7.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

The interfaces that are implemented by the services are denoted as "Service Interface". The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as "Application Interface".

7.4 Generic Service Interface

7.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.

<<Interface>> IpService
setCallback (appInterface : in IpInterfaceRef) : TpResult setCallbackWithSessionID (appInterface : in IpInterfaceRef, sessionID : in TpSessionID) : TpResult

Method
`setCallback()`

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application.

Parameters

`appInterface: in IpInterfaceRef`

Specifies a reference to the application interface, which is used for callbacks

Raises

`TpGeneralException`

Method
`setCallbackWithSessionID()`

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg.

Parameters

`appInterface: in IpInterfaceRef`

Specifies a reference to the application interface, which is used for callbacks

`sessionID: in TpSessionID`

Specifies the session for which the service can invoke the application's callback interface.

Raises

`TpGeneralException`

8 Data Session Control Interface Classes

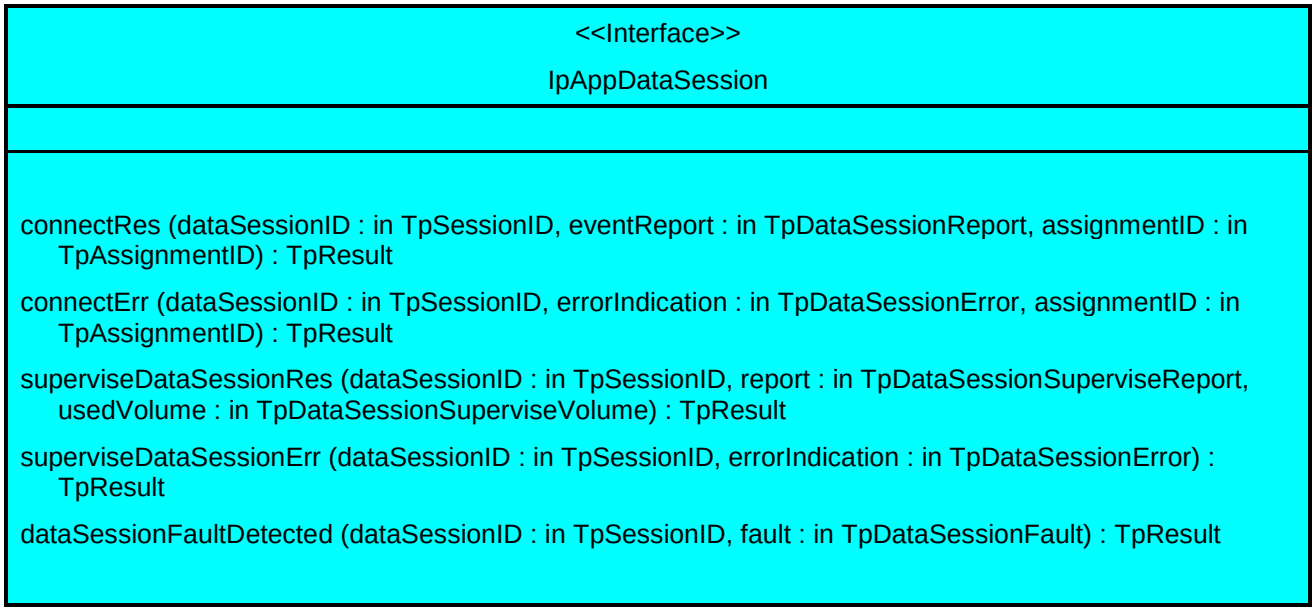
The Data Session Control provides a means to control per data session basis the establishment of a new data session. This means espcially in the GPRS context that the establishment of a PDP session is modelled not the attach/detach mode. Change of terminal location is assumed to be managed by the underlying network and is therefore not part of the model. The underlying assumption is that a terminal initiates a data session and the application can reject the request for data session establishment, can continue the establishment or can continue and change the destination as requested by the terminal.

The modelling is hold similar to the Generic Call Control but assuming a simpler underlying state model. An IpDataSessionManager and IpData Session object are the interfaces used by the application, whereas the IpAppDataSessionManager and the IpAppDataSession interfaces are implemented by the application.

8.1 Interface Class IpAppDataSession

Inherits from: IpInterface.

The application side of the data session interface is used to handle data session request responses and state reports.



Method

`connectRes()`

This asynchronous method indicates that the request to connect a data session with the destination party was successful, and indicates the response of the destination party (e.g. connected, disconnected).

Parameters

dataSessionID: in TpSessionID

Specifies the session ID of the data session.

eventReport: in TpDataSessionReport

Specifies the result of the request to connect the data session. It includes the network event, date and time, monitoring mode and event specific information such as release cause.

assignmentID: in TpAssignmentID

Raises

TpDSCSException,TpGeneralException

Method

connectErr()

This asynchronous method indicates that the request to connect a data session with the destination party was unsuccessful, e.g. an error detected in the network or the data session was abandoned.

Parameters

dataSessionID: in TpSessionID

Specifies the session ID.

errorIndication: in TpDataSessionError

Specifies the error which led to the original request failing.

assignmentID: in TpAssignmentID

Raises

TpDSCSException,TpGeneralException

Method

superviseDataSessionRes()

This asynchronous method reports a data session supervision event to the application.

Parameters

dataSessionID: in TpSessionID

Specifies the data session.

report: in TpDataSessionSuperviseReport

Specifies the situation, which triggered the sending of the data session supervision response.

usedVolume: in TpDataSessionSuperviseVolume

Specifies the used volume for the data session supervision (in the same unit as specified in the request).

Raises

TpDSCSException,TpGeneralException

Method

superviseDataSessionErr()

This asynchronous method reports a data session supervision error to the application.

Parameters

dataSessionID: in TpSessionID

Specifies the data session ID.

errorIndication: in TpDataSessionError

Specifies the error which led to the original request failing.

Raises

TpDSCSException,TpGeneralException

Method

dataSessionFaultDetected()

This method indicates to the application that a fault in the network has been detected which can't be communicated by a network event, e.g., when the user aborts before any establishment method is called by the application.

The system purges the Data Session object. Therefore, the application has no further control of data session processing. No report will be forwarded to the application.

Parameters

dataSessionID: in TpSessionID

Specifies the data session ID of the Data Session object in which the fault has been detected

fault: in TpDataSessionFault

Specifies the fault that has been detected.

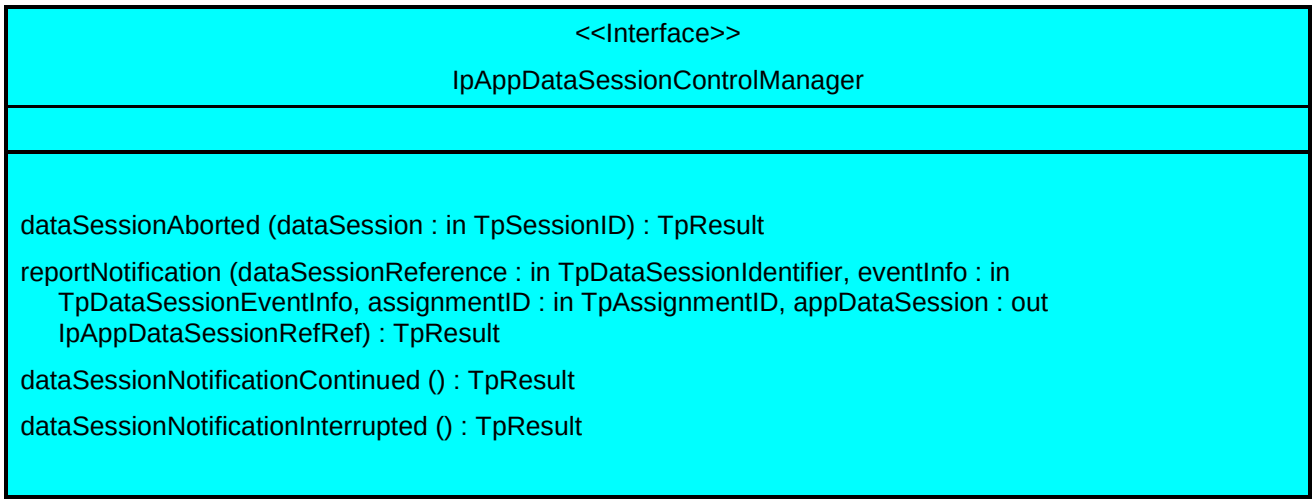
Raises

TpDSCSException,TpGeneralException

8.2 Interface Class IpAppDataSessionControlManager

Inherits from: IpInterface.

The data session control manager application interface provides the application data session control management functions to the data session control SCF.



Method

dataSessionAborted()

This method indicates to the application that the Data Session object has aborted or terminated abnormally. No further communication will be possible between the Data Session object and the application.

Parameters

dataSession: in TpSessionID

Specifies the session ID of the data session that has aborted or terminated abnormally.

Raises

TpDSCSException,TpGeneralException

Method

reportNotification()

This method notifies the application of the arrival of a data session-related event.

Parameters

dataSessionReference: in TpDataSessionIdentifier

Specifies the session ID and the reference to the Data Session object to which the notification relates.

eventInfo: in TpDataSessionEventInfo

Specifies data associated with this event. This data includes the destination address provided by the end-user.

assignmentID: in TpAssignmentID

Specifies the assignment id which was returned by the createNotification() method. The application can use assignment ID to associate events with event-specific criteria and to act accordingly.

appDataSession: out IpAppDataSessionRefRef

Specifies a reference to the application object which implements the callback interface for the new data session.

Raises

TpDSCSException,TpGeneralException

Method

dataSessionNotificationContinued()

This method indicates to the application that all event notifications are resumed.

Parameters

No Parameters were identified for this method

Raises

TpDSCSException,TpGeneralException

Method

dataSessionNotificationInterrupted()

This method indicates to the application that event notifications will no longer be sent (for example, due to faults detected).

Parameters

No Parameters were identified for this method

Raises

TpDSCSException,TpGeneralException

8.3 Interface Class IpDataSession

Inherits from: IpService.

The Data Session interface provides basic methods for applications to control data sessions.

<<Interface>> IpDataSession
connectReq (dataSessionID : in TpSessionID, responseRequested : in TpDataSessionReportRequestSet, targetAddress : in TpAddress, assignmentID : out TpAssignmentIDRef) : TpResult release (dataSessionID : in TpSessionID, cause : in TpDataSessionReleaseCause) : TpResult superviseDataSessionReq (dataSessionID : in TpSessionID, treatment : in TpDataSessionSuperviseTreatment, bytes : in TpDataSessionSuperviseVolume) : TpResult setDataSessionChargePlan (dataSessionID : in TpSessionID, dataSessionChargePlan : in TpDataSessionChargePlan) : TpResult setAdviceOfCharge (dataSessionID : in TpSessionID, aoCInfo : in TpAoCInfo, tariffSwitch : in TpDuration) : TpResult

Method

`connectReq()`

This asynchronous method requests the connection of a data session with the destination party (specified in the parameter TargetAddress). The Data Session object is not automatically deleted if the destination party disconnects from the data session.

Parameters

`dataSessionID: in TpSessionID`

Specifies the session ID.

`responseRequested: in TpDataSessionReportRequestSet`

Specifies the set of observed data session events that will result in a connectRes() being generated.

`targetAddress: in TpAddress`

Specifies the address of destination party.

`assignmentID: out TpAssignmentIDRef`

Specifies the ID assigned to the request. The same ID will be returned in the connectRes or Err. This allows the application to correlate the request and the result.

Raises

`TpDSCSException,TpGeneralException`

Method

release()

This method requests the release of the data session and associated objects.

Parameters

dataSessionID: in TpSessionID

Specifies the session.

cause: in TpDataSessionReleaseCause

Specifies the cause of the release.

Raises

TpDSCSException,TpGeneralException

Method

superviseDataSessionReq()

The application calls this method to supervise a data session. The application can set a granted data volume for this data session. If an application calls this function before it calls a connectReq() or a user interaction function the time measurement will start as soon as the data session is connected. The Data Session object will exist after the data session has been terminated if information is required to be sent to the application at the end of the data session

Parameters

dataSessionID: in TpSessionID

Specifies the data session.

treatment: in TpDataSessionSuperviseTreatment

Specifies how the network should react after the granted data volume has been sent.

bytes: in TpDataSessionSuperviseVolume

Specifies the granted number of bytes that can be transmitted for the data session.

Raises

TpDSCSException,TpGeneralException

Method

setDataSessionChargePlan()

Allows an application to include charging information in network generated CDR.

Parameters

dataSessionID: in TpSessionID

Specifies the session ID of the data session.

dataSessionChargePlan: in TpDataSessionChargePlan

Specifies the charge plan used.

Raises

TpDSCSException,TpGeneralException

Method

setAdviceOfCharge()

This method allows the application to determine the charging information that will be send to the end-users terminal.

Parameters

dataSessionID: in TpSessionID

Specifies the session ID of the data session.

aoCInfo: in TpAoCInfo

Specifies two sets of Advice of Charge parameter according to GSM.

tariffSwitch: in TpDuration

Specifies the tariff switch that signifies when the second set of AoC parameters becomes valid.

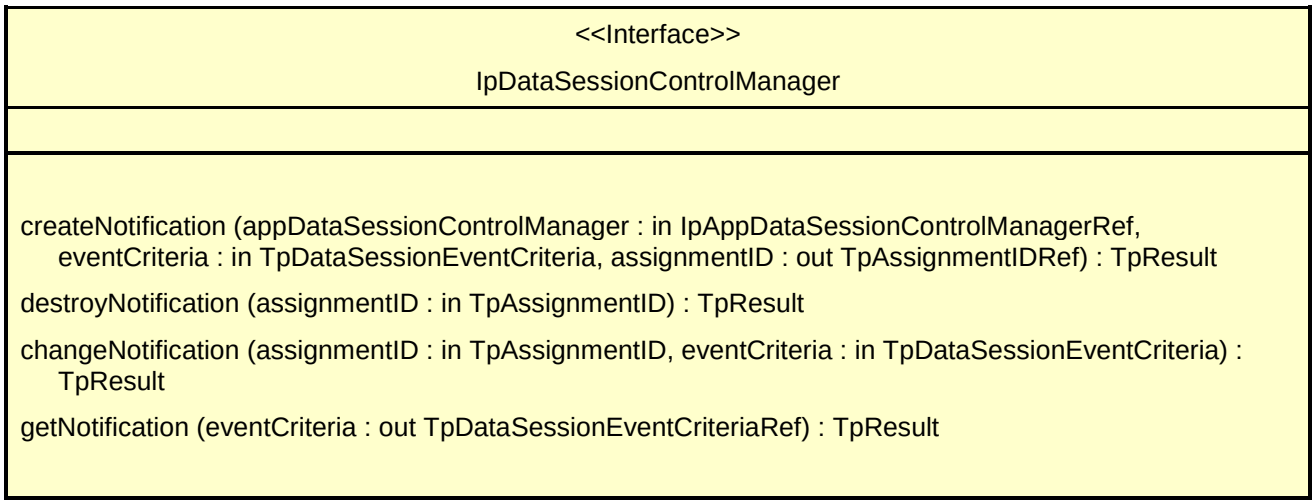
Raises

TpDSCSException,TpGeneralException

8.4 Interface Class IpDataSessionControlManager

Inherits from: IpService.

This interface is the SCF manager' interface for Data Session Control.



Method
createNotification()

This method is used to enable data session notifications.

Parameters

appDataSessionControlManager: in IpAppDataSessionControlManagerRef

If this parameter is set (i.e. not NULL) it specifies a reference to the application interface which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the setCallback() method.

eventCriteria: in TpDataSessionEventCriteria

Specifies the event specific criteria used by the application to define the event required. Individual addresses or address ranges may be specified for destination and/or origination. Examples of events are "Data Session set up".

assignmentID: out TpAssignmentIDRef

Specifies the ID assigned by the Data Session Manager object for this newly-enabled event notification.

Raises

TpDSCSException,TpGeneralException

Method
destroyNotification()

This method is used by the application to disable data session notifications.

Parameters

assignmentID: in TpAssignmentID

Specifies the assignment ID given by the data session manager object when the previous createNotification() was done.

Raises

TpDSCSException,TpGeneralException

Method
changeNotification()

This method is used by the application to change the event criteria introduced with the createNotification method. Any stored notification request associated with the specified assignmentID will be replaced with the specified events requested.

Parameters

assignmentID: in TpAssignmentID

Specifies the ID assigned by the manager interface for the event notification.

eventCriteria: in TpDataSessionEventCriteria

Specifies the enw set of event criteria used by the application to define the event required. Only events that meet these criteria are reported.

Raises

`TpDSCSException,TpGeneralException`

Method

`getNotification()`

This method is used by the application to query the event criteria set with createNotification or changeNotification.

Parameters

`eventCriteria: out TpDataSessionEventCriteriaRef`

Specifies the event criteria used by the application to define the event required. Only events that meet these requirements are reported.

9 State Transition Diagrams

9.1 State Transition Diagrams for IpDataSession

The state transition diagram shows the application view on the Data Session object. This diagram shows only the part of the state transition diagram valid for 3GPP (UMTS) release 99.

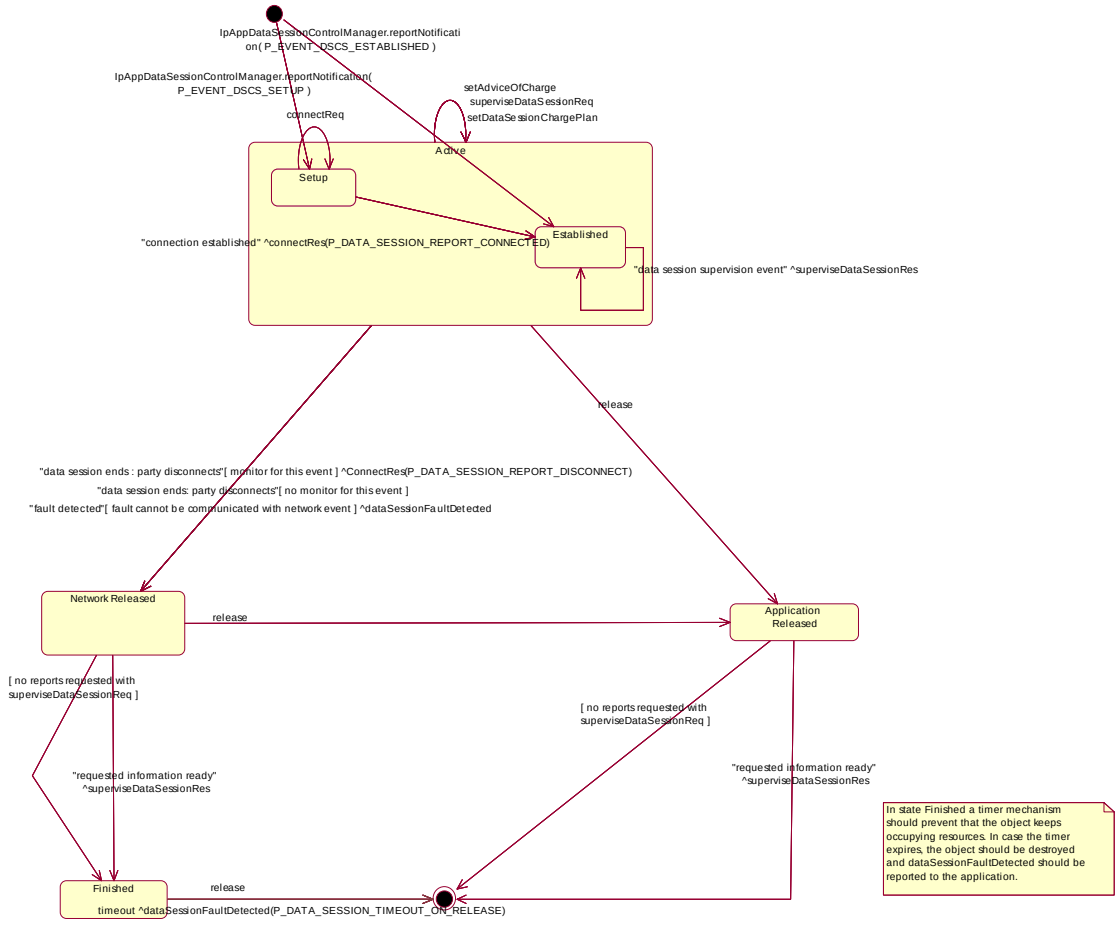


Figure: Application view on the Data Session object

9.1.1 Network Released State

In this state the data session has ended. In the case on a normal user disconnection the transition to this state is indicated to the application by the disconnect report of connectRes(). But this will only happen if the application requested monitoring of the disconnect event before. An abnormal disconnection is indicated by dataSessionFaultDetected(). The application may wait for outstanding superviseDataSessionRes().

9.1.2 Finished State

In this state the data session has ended and no further data session related information is to be send to the application. The application can only release the data session object. If the application fails to invoke release() within a certain period of time the gateway should automatically release the object and send a timeout indication to the application.

9.1.3 Application Released State

In this state the application has released the data session object. If supervision has been requested the gateway will collect the information and send superviseDataRes() to the application.

9.1.4 Active State

In this state a data connection between two parties is being setup or established (refer to the substates for more details). The application can request the gateway for a certain type of charging by calling setDataSessionChargePlan(), send advice of charge information by calling setAdviceOfCharge(), and request supervision of the data session by calling superviseDataSessionReq().

9.1.5 Setup State

The Setup state is reached after a reportNotification() indicates to the application that a data session is interested in being connected. If the application is going to connect the two parties by invoking connectReq() it may call the charging or supervision methods before.

9.1.6 Established State

In this state the data connection is established. If supervision has been requested the application expects the corresponding superviseDataSessionRes().

10 Data Definitions

10.1 Data Session Control Data Definitions

IpAppDataSession

Defines the address of an `IpAppDataSession` Interface.

IpAppDataSessionRef

Defines a Reference to type `IpAppDataSession`

IpAppDataSessionRefRef

Defines a Reference to type `IpAppDataSessionRef`.

IpAppDataSessionControlManager

Defines the address of an `IpAppDataSessionControlManager` Interface.

IpAppDataSessionControlManagerRef

Defines a Reference to type IpAppDataSessionControlManager.

IpDataSession

Defines the address of an IpDataSession Interface.

IpDataSessionRef

Defines a Reference to type IpDataSession.

IpDataSessionRefRef

Defines a Reference to type IpDataSessionRef.

IpDataSessionControlManager

Defines the address of an IpDataSessionManager Interface.

IpDataSessionManagerRef

Defines a Reference to type IpDataSessionControlManager.

10.2 Event Notification data definitions

TpDataSessionEventName

Defines the names of events being notified with a new call request. The following events are supported. The values may be combined by a logical 'OR' function when requesting the notifications. Additional events that can be requested / received during the call process are found in the TpDataSessionReportType data-type.

Name	Value	Description
P_EVENT_NAME_UNDEFINED	0	Undefined
P_EVENT_DSCS_SETUP	1	The data session is going to be setup.
P_EVENT_DSCS_ESTABLISHED	2	The data session is established by the network.

TpDataSessionMonitorMode

Defines the mode that the call will monitor for events, or the mode that the call is in following a detected event.

Name	Value	Description
P_DATA_SESSION_MONITOR_MODE_INTERRUPT	0	The data session event is intercepted by the data session control service and data session establishment is interrupted. The application is notified of the event and data session establishement resumes following an appropriate API call or network event (such as a data session release)
P_DATA_SESSION_MONITOR_MODE_NOTIFY	1	The data session event is detected by the data session control service but not intercepted. The application is notified of the event and data session establishment continues
P_DATA_SESSION_MONITOR_MODE_DO_NOT_MONITOR	2	Do not monitor for the event

TpDataSessionEventCriteria

Defines the Sequence of Data Elements that specify the criteria for a event notification.

Of the addresses only the Plan and the AddrString are used for the purpose of matching the notifications against the criteria.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddressRange	Defines the destination address or address range for which the notification is requested.
OriginatingAddress	TpAddressRange	Defines the origination address or a address range for which the notification is requested.
DataSessionEventName	TpDataSessionEventName	Name of the event(s)
MonitorMode	TpDataSessionMonitorMode	Defines the mode that the Data Session is in following the notification. Monitor mode P_DATA_SESSION_MONITOR_MODE_DO_NOT_MONITOR is not a legal value here.

TpDataSessionEventInfo

Defines the Sequence of Data Elements that specify the information returned to the application in a Data Session event notification.

Sequence Element Name	Sequence Element Type
DestinationAddress	TpAddress
OriginatingAddress	TpAddress
DataSessionEventName	TpDataSessionEventName
MonitorMode	TpDataSessionMonitorMode

TpDataSessionChargePlan

Defines the Sequence of Data Elements that specify the charge plan for the call.

Sequence Element Name	Sequence Element Type	Description
ChargeOrderType	TpDataSessionChargeOrder	Charge order
Currency	TpString	Currency unit according to ISO-4217:1995
AdditionalInfo	TpString	Descriptive string which is sent to the billing system without prior evaluation. Could be included in the ticket.

Valid Currencies are:

ADP, AED, AFA, ALL, AMD, ANG, AON, AOR, ARS, ATS, AUD, AWG, AZM, BAM,

BBD, BDT, BEF, BGL, BGN, BHD, BIF, BMD, BND, BOB, BOV, BRL, BSD, BTN,

BWP, BYB, BZD, CAD, CDF, CHF, CLF, CLP, CNY, COP, CRC, CUP, CVE, CYP,

CZK, DEM, DJF, DKK, DOP, DZD, ECS, ECV, EEK, EGP, ERN, ESP, ETB, EUR,

FIM, FJD, FKP, FRF, GBP, GEL, GHC, GIP, GMD, GNF, GRD, GTQ, GWP, GYD,

HKD, HNL, HRK, HTG, HUF, IDR, IEP, ILS, INR, IQD, IRR, ISK, ITL, JMD,

JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KYD, KZT, LAK, LBP, LKR,

LRD, LSL, LTL, LUF, LVL, LYD, MAD, MDL, MGF, MKD, MMK, MNT, MOP, MRO,

MTL, MUR, MVR, MWK, MXN, MXV, MYR, MZM, NAD, NGN, NIO, NLG, NOK, NPR,

NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PTE, PYG, QAR, ROL, RUB, RUR,

RWF, SAR, SBD, SCR, SDD, SEK, SGD, SHP, SIT, SKK, SLL, SOS, SRG, STD,

SVC, SYP, SZL, THB, TJR, TMM, TND, TOP, TPE, TRL, TTD, TWD, TZS, UAH,

UGX, USD, USN, USS, UYU, UZS, VEB, VND, VUV, WST, XAF, XAG, XAU, XBA,

XBB, XBC, XBD, XCD, XDR, XFO, XFU, XOF, XPD, XPF, XPT, XTS, XXX, YER,

YUM, ZAL, ZAR, ZMK, ZRN, ZWD.

XXX is used for transactions where no currency is involved.

TpDataSessionChargeOrder

Defines the Tagged Choice of Data Elements that specify the charge plan for the call.

	Tag Element Type	
	TpDataSessionChargeOrderCategory	

Tag Element Value	Choice Element Type	Choice Element Name
P_DATA_SESSION_CHARGE_PER_VOLUME	TpChargePerVolume	ChargePerVolume
P_DATA_SESSION_CHARGE_NETWORK	TpString	NetworkCharge

TpDataSessionChargeOrderCategory

Name	Value	Description
P_DATA_SESSION_CHARGE_PER_VOLUME	0	Charge per volume
P_DATA_SESSION_CHARGE_NETWORK	1	Operator specific charge plan specification, e.g. charging table name / charging table entry

TpChargePerVolume

Defines the Sequence of Data Elements that specify the time based charging information. The volume is the sum of uplink and downlink transfer data volumes.

Sequence Element Name	Sequence Element Type	Description
InitialCharge	TpInt32	Initial charge amount (in currency units * 0.0001)
CurrentChargePerKilobyte	TpInt32	Current tariff (in currency units * 0.0001)
NextChargePerKilobyte	TpInt32	Next tariff (in currency units * 0.0001) after tariff switch. Only used in setAdviceOfCharge()

TpDataSessionIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Data Session object

Sequence Element Name	Sequence Element Type	Sequence Element Description
DataSessionReference	IpDataSessionRef	This element specifies the interface reference for the Data Session object.

DataSessionSessionID	TpSessionID	This element specifies the data session ID of the Data Session.

TpDataSessionError

Defines the Sequence of Data Elements that specify the additional information relating to acall error.

Sequence Element Name	Sequence Element Type
ErrorTime	TpDateAndTime
ErrorType	TpDataSessionErrorType
AdditionalErrorInfo	TpDataSessionAdditionalErrorInfo

TpDataSessionAdditionalErrorInfo

Defines the Tagged Choice of Data Elements that specify additional Data Session error and Data Session error specific information.

	Tag Element Type	
	TpDataSessionErrorType	

Tag Element Value	Choice Element Type	Choice Element Name
P_DATA_SESSION_ERROR_UNDEFINED	NULL	Undefined
P_DATA_SESSION_ERROR_INVALID_ADDRESS	TpAddressError	DataSessionErrorInvalidAddress
P_DATA_SESSION_ERROR_INVALID_STATE	NULL	Undefined

TpDataSessionErrorType

Defines a specific Data Session error.

Name	Value	Description
P_DATA_SESSION_ERROR_UNDEFINED	0	Undefined; the method failed or was refused, but no specific reason can be given.
P_DATA_SESSION_ERROR_INVALID_ADDRESS	1	The operation failed because an invalid address was given
P_DATA_SESSION_ERROR_INVALID_STATE	2	The data session was not in a valid state for the requested operation

TpDataSessionFault

Defines the cause of the data session fault detected.

Name	Value	Description
P_DATA_SESSION_FAULT_UNDEFINED	0	Undefined
P_DATA_SESION_USER_ABORTED	1	User has finalised the data session before any message could be sent by the application
P_DATA_SESSION_TIMEOUT_ON_RELEASE	2	This fault occurs when the final report has been sent to the application, but the application did not explicitly release data session object, within a specified time. The timer value is operator specific.
P_DATA_SESSION_TIMEOUT_ON_INTERRUPT	3	This fault occurs when the application did not instruct the gateway how to handle the call within a specified time, after the gateway

		reported an event that was requested by the application in interrupt mode. The timer value is operator specific.

TpDataSessionReleaseCause

Defines the Sequence of Data Elements that specify the cause of the release of a data session.

Sequence Element Name	Sequence Element Type
Value	TpInt32
Location	TpInt32

NOTE: the Value and Location are specified as in ITU-T recommendation Q.850.

TpDataSessionSuperviseVolume

Defines the Sequence of Data Elements that specify the amount of volume that is allowed to be transmitted for the specific connection.

Sequence Element Name	Sequence Element Type	Sequence Element Description
VolumeQuantity	TpInt32	This data type is identical to a TpInt32, and defines the quantity of the granted volume that can be transmitted for the specific connection. The volume specifies the sum of uplink and downlink transfer data volumes.
VolumeUnit	TpInt32	In Order to enlarge the range of the volume quantity value the exponent of a scaling factor (10^VolumeUnit) is provided. When the unit is for example in kilobytes, VolumeUnit must be set to 3.

TpDataSessionSuperviseReport

Defines the responses from the data session control service for calls that are supervised. The values may be combined by a logical 'OR' function.

Name	Value	Description
P_DATA_SESSION_SUPERVISE_VOLUME_REACHED	01h	The maximum volume has been reached.
P_DATA_SESSION_SUPERVISE_DATA_SESSION_ENDED	02h	The data session has ended, either due to data session party to reach of maximum volume or calling or called release.
P_DATA_SESSION_SUPERVISE_MESSAGE_SENT	04h	A warning message has been sent.

TpDataSessionSuperviseTreatment

Defines the treatment of the call by the data session control service when the supervised volume is reached. The values may be combined by a logical 'OR' function.

Name	Value	Description
P_DATA_SESSION_SUPERVISE_RELEASE	01h	Release the data session when the data session supervision volume is reached.
P_DATA_SESSION_SUPERVISE_RESPOND	02h	Notify the application when the call supervision volume is reached.
P_DATA_SESSION_SUPERVISE_INFORM	04h	Send a warning message to the originating party when the maximum volume is reached. If data session release is requested, then the data

		session will be released following the message after an administered time period

TpDataSessionReport

Defines the Sequence of Data Elements that specify the data session report specific information.

Sequence Element Name	Sequence Element Type
MonitorMode	TpDataSessionMonitorMode
DataSessionEventTime	TpDateAndTime
DataSessionReportType	TpDataSessionReportType
AdditionalReportInfo	TpDataSessionAdditionalReportInfo

TpDataSessionAdditionalReportInfo

Defines the Tagged Choice of Data Elements that specify additional data session report information for certain types of reports.

	Tag Element Type	
	TpDataSessionReportType	

Tag Element Value	Choice Element Type	Choice Element Name
P_DATA_SESSION_REPORT_UNDEFINED	NULL	Undefined
P_DATA_SESSION_REPORT_CONNECTED	NULL	Undefined
P_DATA_SESSION_REPORT_DISCONNECT	TpDataSessionReleaseCause	DataSessionDisconnect

TpDataSessionReportRequest

Defines the Sequence of Data Elements that specify the criteria relating to data session report requests.

Sequence Element Name	Sequence Element Type
MonitorMode	TpDataSessionMonitorMode
DataSessionReportType	TpDataSessionReportType

TpDataSessionReportRequestSet

Defines a Numbered Set of Data Elements of TpDataSessionReportRequest.

TpDataSessionReportType

Defines a specific data session event report type.

Name	Value	Description
P_DATA_SESSION_REPORT_UNDEFINED	0	Undefined
P_DATA_SESSION_REPORT_CONNECTED	1	Data session established.
P_DATA_SESSION_REPORT_DISCONNECT	2	Data session disconnect requested by data session party

Annex A (normative):
OMG IDL Description of Data Session Control SCF

The OMG IDL representation of this interface specification is contained in a text file (dsc.idl contained in archive 2919808IDL.ZIP) which accompanies the present document.

Annex B (informative): Differences between this draft and 3GPP 29.198 R99

C.1 Interface IpAppDataSessionControlManager

reportNotification~~dataSessionEventNotify~~ (dataSessionReference : in TpDataSessionIdentifier, eventInfo : in TpDataSessionEventInfo, assignmentID : in TpAssignmentID, app~~Interface~~DataSession : out IpAppDataSessionRefRef) : TpResult

C.2 Interface IpDataSessionControlManager

createNotification~~enableDataSessionNotification~~ (appDataSessionControlManager~~Interface~~ : in IpAppDataSessionControlManagerRef, eventCriteria : in TpDataSessionEventCriteria, assignmentID : out TpAssignmentIDRef) : TpResult

destroyNotification~~disableDataSessionNotification~~ (assignmentID : in TpAssignmentID) : TpResult

changeNotification (assignmentID : in TpAssignmentID, eventCriteria : in TpDataSessionEventCriteria) : TpResult

getNotification (eventCriteria : out TpDataSessionEventCriteriaRef) : TpResult

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for approval and placement under Change Control

3GPP TS 29.198-11 V1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface
Part 11: Account Management
(Release 4)

Keywords
API, OSA, IDL, AM, Account Management

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

This document is part of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA). The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA API's. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127[2].

This document specifies the Account Management Service Capability Feature (SCF) aspects of the interface. All aspects of the Account Management SCF are defined here, these being:

- Sequence Diagrams
- Class Diagrams
- Interface specification plus detailed method descriptions
- State Transition diagrams
- Data definitions
- IDL Description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modeling Language (UML).

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

[1] 3GPP TS 29.198-1 "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in TS 29.198-1 [1] apply.

3.2 Symbols

For the purposes of the present document, the symbols in TS 29.198-1 [1] apply.

3.3 Abbreviations

For the purposes of the present document, the abbreviations in TS 29.198-1 [1] apply.

4 Account Management SCF

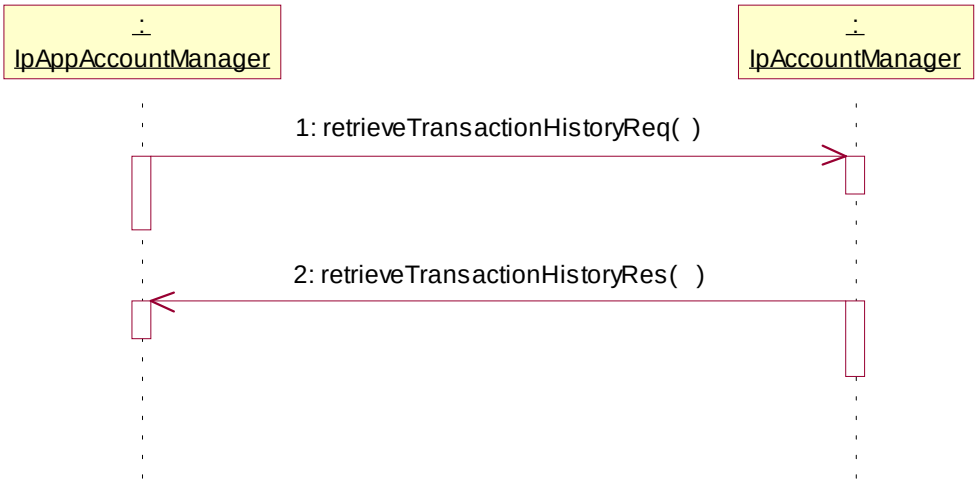
The following sections describe each aspect of the Account Management Service Capability Feature (SCF).

The order is as follows:

- The Sequence diagrams give the reader a practical idea of how each of the service capability features is implemented.
- The Class relationships section show how each of the interfaces applicable to the SCF, relate to one another
- The Interface specification section describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the progression of internal processes either in the application, or Gateway.
- The Data definitions section shows a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.

5 Sequence Diagrams

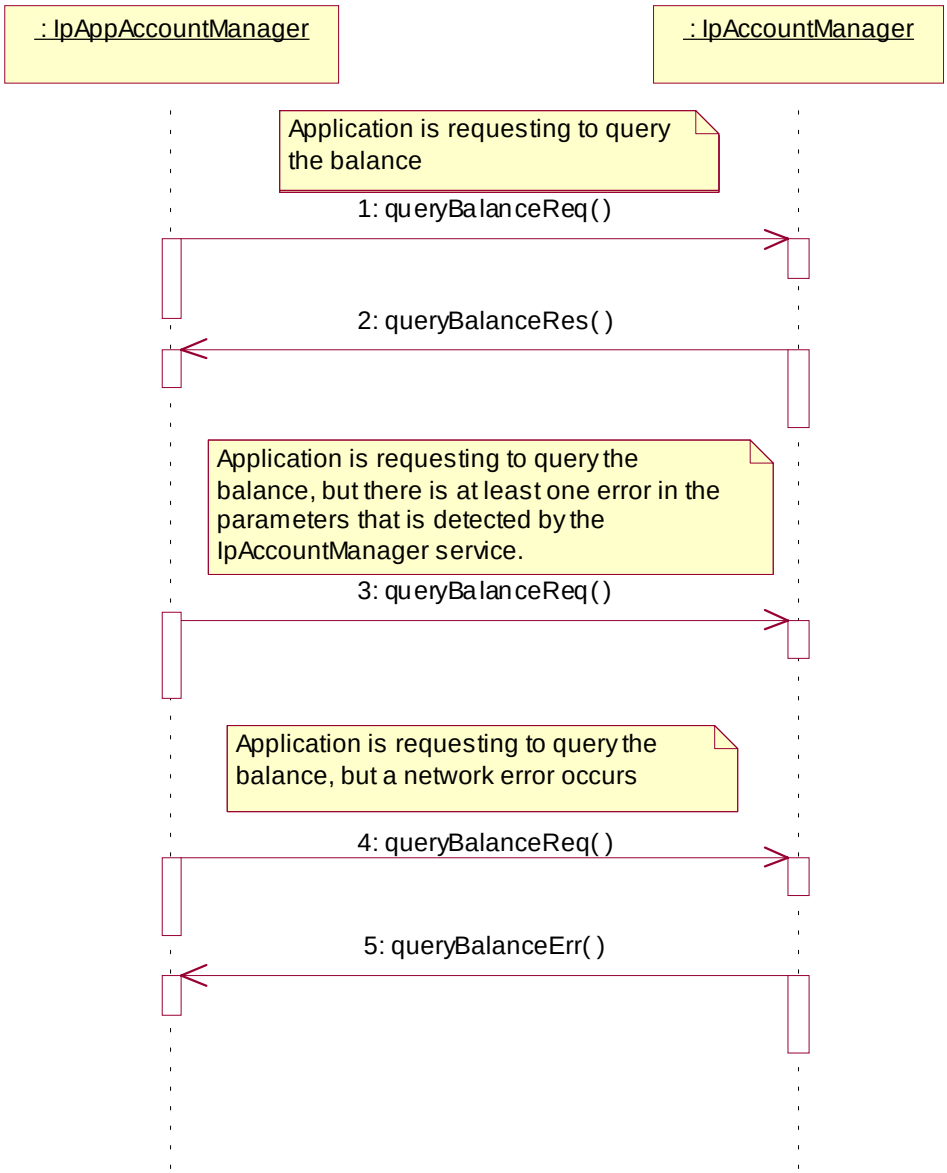
5.1 Standard Transaction History Retrieval



1: This message is used by the application to retrieve a transaction history for a certain subscriber's account.

2: This method passes the result of the transaction history retrieval request for a specific user to its callback object.

5.2 Standard Query Handling



1: This message is used to query the balance of the account of one or several users.

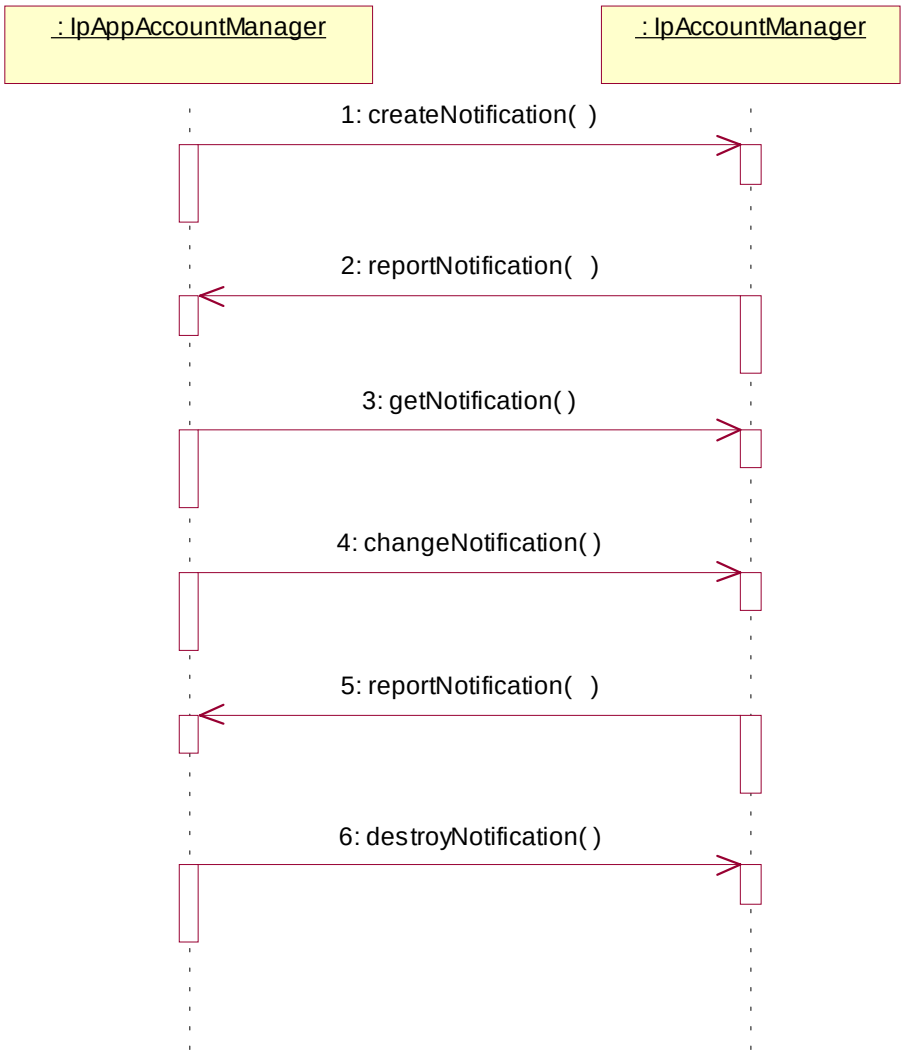
2: This message passes the result of the balance query for one or several users to its callback object.

3: This scenario shows the case where at least one error in the parameters of the message is detected by the IpAccountManager object. An exception will be thrown.

4: This scenario shows the case where a network error occurs.

5: This message passes the error of the balance query. No exception is thrown.

5.3 Standard Notification handling



1: This message is used by the application to request notifications from the IpAccountManager service on certain criteria for one or several users.

2: This message is used by the IpAccountManager service to report a charging event that meets the criteria set in the createNotification message.

3: The application can request the current criteria set in the IpAccountManager service by invoking the getNotification method.

4: This message is used by the application to change the criteria initially created by createNotification, and previously obtained by getNotification.

5: This message is used by the IpAccountManager service to report a charging event that meets the new criteria.

6: This method is used by the application to disable the charging notifications.

6 Class Diagrams

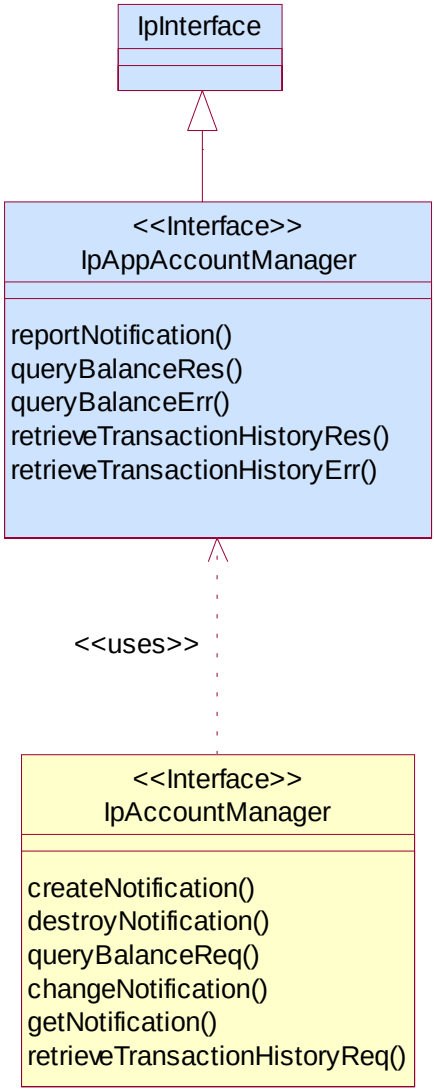


Figure 1: Application Interfaces

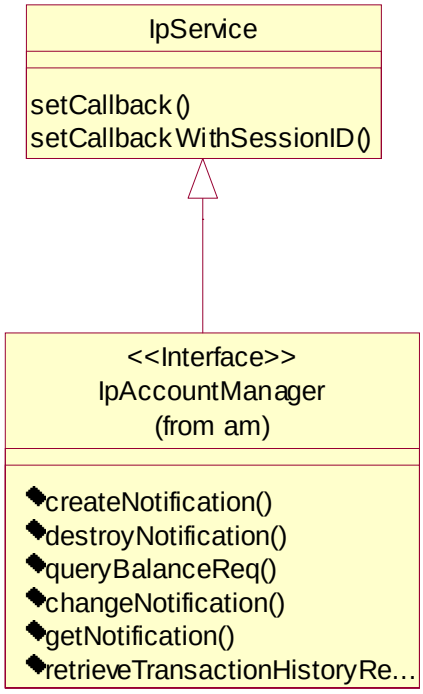


Figure 2: Service Interfaces

7 The Service Interface Specifications

7.1 Interface Specification Format

This section defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

7.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for enterprise-based client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>

7.1.2 Method descriptions

Each method (API method "call") is described. All methods in the API return a value of type `TpResult`, indicating, amongst other things, if the method invocation was sucessfully executed or not.

Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a '`Req`' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a '`Res`' or '`Err`' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name> or IpSvc<name>` interfaces to provide the callback mechanism.

7.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.

7.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

7.2 Base Interface

7.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.

<<Interface>> IpInterface

7.3 Service Interfaces

7.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

7.4 Generic Service Interface

7.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.

<<Interface>> IpService
setCallback (appInterface : in IpInterfaceRef) : TpResult setCallbackWithSessionID (appInterface : in IpInterfaceRef, sessionID : in TpSessionID) : TpResult

Method
setCallback()

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

Raises

TpGeneralException

Method
setCallbackWithSessionID()

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

sessionID : in TpSessionID

Specifies the session for which the service can invoke the application's callback interface.

Raises

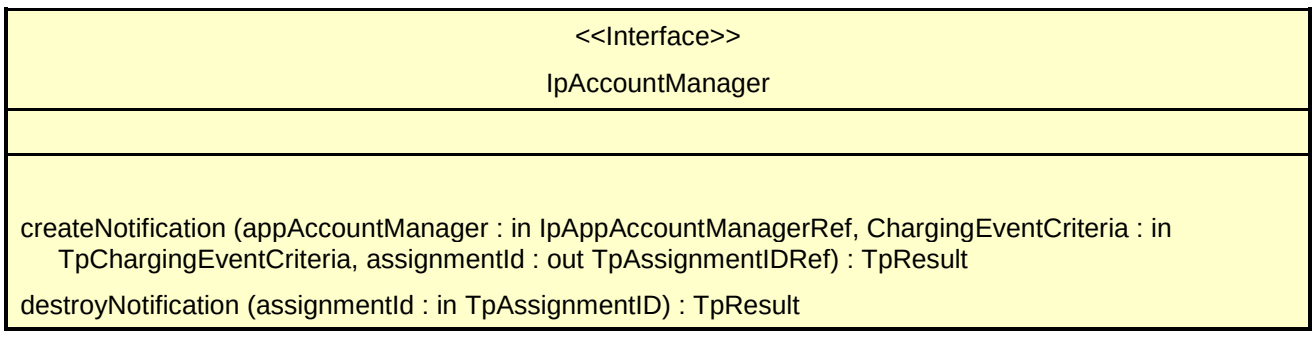
TpGeneralException

8 Account Management Interface Classes

8.1 Interface Class IpAccountManager

Inherits from: IpService.

The account manager interface provides methods for monitoring accounts. Applications can use this interface to enable or disable charging-related event notifications and to query account balances.



queryBalanceReq (users : in TpAddressSet, queryId : out TpSessionIDRef) : TpResult

changeNotification (assignmentID : in TpAssignmentID, eventCriteria : in TpChargingEventCriteria) : TpResult

getNotification (eventCriteria : out TpChargingEventCriteriaRef) : TpResult

retrieveTransactionHistoryReq (user : in TpAddress, transactionInterval : in fw::TpTimeInterval, retrievalID : out TpSessionIDRef) : TpResult

Method
createNotification()

This method is used by the application to enable charging event notifications to be sent to the application.

Parameters

appAccountManager : in IpAppAccountManagerRef

If this parameter is set (i.e. not NULL), it specifies a reference to the application interface that is used for callbacks. If it is set to NULL, the application interface defaults to the interface specified via the setCallback() method.

ChargingEventCriteria : in TpChargingEventCriteria

Specifies the event specific criteria used by the application to define the charging event required. Individual addresses or address ranges may be specified for subscriber accounts. Example of events are "charging" and "recharging".

assignmentId : out TpAssignmentIDRef

Specifies the ID assigned by the account management object for this newly enabled event notification.

Raises

TpAMException,TpGeneralException

Method
destroyNotification()

This method is used by the application to disable charging notifications.

Parameters

assignmentId : in TpAssignmentID

Specifies the assignment ID that was given by the account management object when the application enabled the charging notification.

Raises

TpAMException,TpGeneralException

Method
queryBalanceReq()

This method is used by the application to query the balance of an account for one or several users.

Parameters

users : in TpAddressSet

Specifies the user(s) for which the balance is queried.

queryId : out TpSessionIDRef

Specifies the ID of the balance query request.

Raises

TpAMException,TpGeneralException

Method

changeNotification()

This method is used by the application to change the event criteria introduced with createNotification. Any stored criteria associated with the specified assignementID will be replaced with the specified criteria.

Parameters

assignmentID : in TpAssignmentID

Specifies the ID assigned by the manager interface for the event notification.

eventCriteria : in TpChargingEventCriteria

Specifies the new set of event criteria used by the application to define the event required. Only events that meet these criteria are reported

Raises

TpAMException,TpGeneralException

Method

getNotification()

This method is used by the application to query the event criteria set with createNotification or changeNotification.

Parameters

eventCriteria : out TpChargingEventCriteriaRef

Specifies the event criteria used by the application to define the event required. Only events that meet these criteria are reported.

Raises

TpAMException,TpGeneralException

Method

retrieveTransactionHistoryReq()

This asynchronous method is used by the application to retrieve a transaction history of a subscriber's account. The history is a set of Detailed Records.

Parameters

user : in TpAddress

Specifies the subscriber for whose account the transaction history is to be retrieved.

transactionInterval : in fw::TpTimeInterval

Specifies the time interval for which the application history is to be retrieved.

retrievalID : out TpSessionIDRef

Specifies the retrieval ID of the transaction history retrieval request.

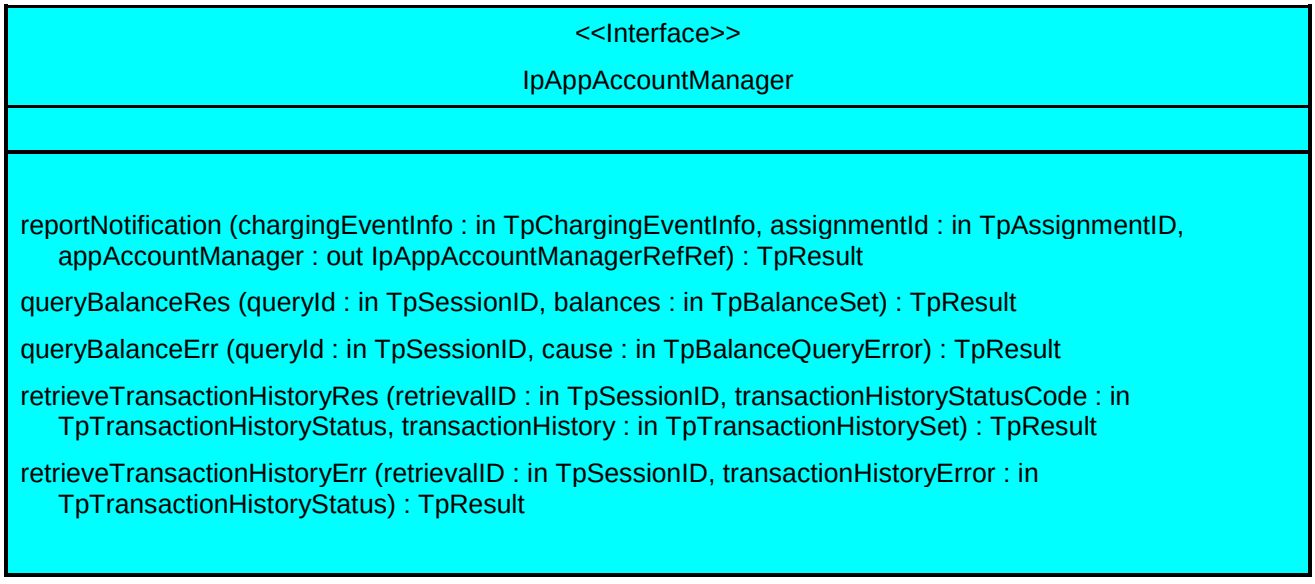
Raises

TpAMException,TpGeneralException

8.2 Interface Class IpAppAccountManager

Inherits from: IpInterface.

The account manager application interface is implemented by the client application developer and is used to handle charging event notifications and query balance responses.



Method

reportNotification()

This method is used to notify the application of a charging event.

Parameters

chargingEventInfo : in TpChargingEventInfo

Specifies data associated with this charging event. These data include the charging event being notified, the current value of the balance after the notified event occurred, and the time at which the charging event occurred.

assignmentId : in TpAssignmentID

Specifies the assignment ID that was returned by the createNotification() method. The application can use the assignment ID to associate events with event-specific criteria and to act accordingly.

appAccountManager : out IpAppAccountManagerRefRef

Specifies a reference to the application object, which implements the callback interface for the new charging session.

Raises

TpAMException,TpGeneralException

Method

queryBalanceRes()

This method indicates that the request to query the balance was successful and it reports the requested balance of an account to the application.

Parameters

queryId : in TpSessionID

Specifies the ID of the balance query request.

balances : in TpBalanceSet

Specifies the balance for one or more user accounts.

Raises

TpAMException,TpGeneralException

Method

queryBalanceErr()

This method indicates that the request to query the balance failed and it reports the cause of failure to the application.

Parameters

queryId : in TpSessionID

Specifies the ID of the balance query request.

cause : in TpBalanceQueryError

Specifies the error that led to the failure.

Raises

TpAMException,TpGeneralException

Method

retrieveTransactionHistoryRes()

This method indicates that the request to retrieve the transaction history was successful and it returns the requested transaction history.

Parameters

retrievalID : in TpSessionID

Specifies the retrievalID of the transaction history retrieval request.

transactionHistoryStatusCode : in TpTransactionHistoryStatus

Specifies the status code for retrieving the transaction history.

transactionHistory : in TpTransactionHistorySet

Specifies the requested transaction history.

Raises

TpAMException,TpGeneralException

Method

retrieveTransactionHistoryErr()

This method indicates that the request to retrieve the transaction history failed and it reports the cause of failure to the application.

Parameters

retrievalID : in TpSessionID

Specifies the retrievalID of the transaction history retrieval request.

transactionHistoryError : in TpTransactionHistoryStatus

Specifies the error that occured while retrieving the transaction history.

Raises

`TpAMException,TpGeneralException`

9 State Transition Diagrams

9.1 State Transition Diagrams for IpAccountManager

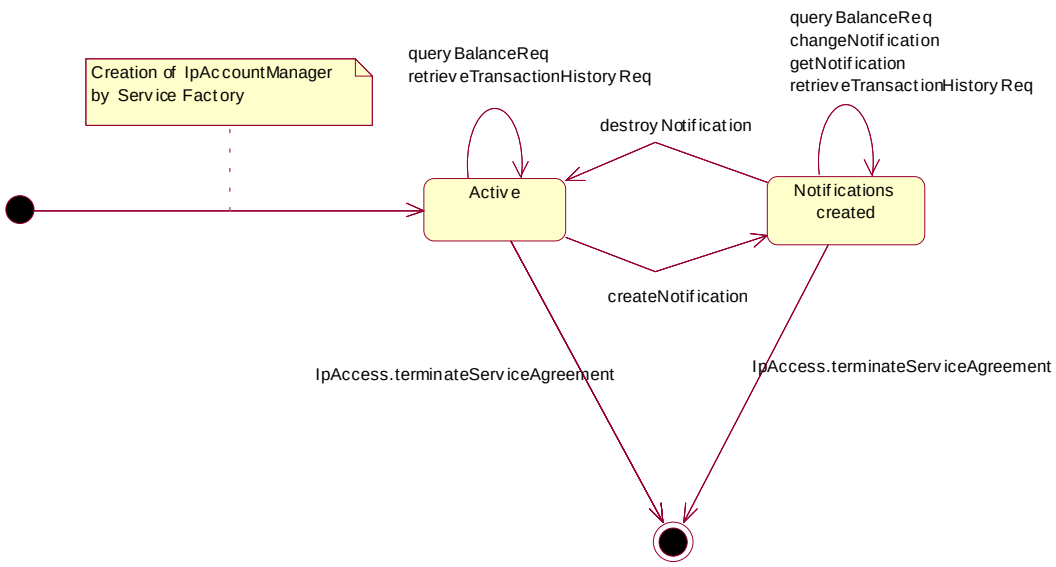


Figure 3: Application view on the IpAccountManager

9.1.1 Active State

In this state a relation between the Application and the Account Management has been established. The state allows the application to indicate that it is interested in charging related events, by calling createNotification. In case such an event occurs, Account Manager will inform the application by invoking the operation reportNotification() on the IpAppAccountManager interface. The application can also indicate it is no longer interested in certain charging related events by calling destroyNotification().

9.1.2 Notifications created State

When the Account Manager is in the Notifications created state, events requested with createNotification() will be forwarded to the application. In this state the application can request to change the notifications or query the Account Manager for the notifications currently set.

10 Data Definitions

10.1 Account Management Data Definitions

This section provides the Account Management specific data definitions necessary to support the OSA interface specification.

The general format of a data definition specification is the following:

- Data type, that shows the name of the data type.
- Description, that describes the data type.
- Tabular specification, that specifies the data types and values of the data type.
- Example, if relevant, shown to illustrate the data type.

10.1.1 TpBalanceQueryError

Defines an error that is reported by the Charging service capability feature as a result of a balance query request.

Name	Value	Description
P_BALANCE_QUERY_OK	0	No error occurred while processing the request
P_BALANCE_QUERY_ERROR_UNDEFINED	1	General error, unspecified
P_BALANCE_QUERY_UNKNOWN_SUBSCRIBER	2	Subscriber for which balance is queried is unknown
P_BALANCE_QUERY_UNAUTHORIZED_APPLICATION	3	Application is not authorized to query balance
P_BALANCE_QUERY_SYSTEM_FAILURE	4	System failure. The request could not be handled

10.1.2 TpChargingEventName

Defines the charging event for which notifications can be requested by the application.

Name	Value	Description
P_AM_CHARGING	0	End user's account has been charged by an application
P_AM_RECHARGING	1	End user has recharged the account
P_AM_ACCOUNT_LOW	2	Account balance is below the balance threshold
P_AM_ACCOUNT_ZERO	3	Account balance is at zero
P_AM_ACCOUNT_DISABLED	4	Account has been disabled

10.1.3 TpBalanceInfo

Defines the structure of data elements that specifies detailed balance info.

Structured Member Name	Structured Member Type	Description

Currency	TpString	Currency unit according to ISO-4217:1995
ValuePartA	TpInt32	This data type is identical to a TpInt32 and specifies the most significant part of the composed value. A currency amount is composed as follows: ((ValuePartA*2^{32}+ ValuePartB) * 0.0001)
ValuePartB	TpInt32	This data type is identical to a TpInt32 and specifies the least significant part of the composed value.
Exponent	TpInt32	Specifies the position of the decimal point in the currency amount made up of the unitPart and the fractionPart, as described above. E.g. an exponent of 4 means a pure integer value, whereas an exponent of 2 means an accuracy of 0.01.
AdditionalInfo	TpString	Descriptive string, containing additional information, which is sent to the application without prior evaluation.

As an example, the currency amount composed of a Currency of EUR, a ValuePartA of 0, a ValuePartB of 10000, and an exponent of 2 yields a currency amount of € 100.00.

Valid Currencies are:

ADP, AED, AFA, ALL, AMD, ANG, AON, AOR, ARS, ATS, AUD, AWG, AZM, BAM, BBD, BDT, BEF, BGL, BGN, BHD, BIF, BMD, BND, BOB, BOV, BRL, BSD, BTN, BWP, BYB, BZD, CAD, CDF, CHF, CLF, CLP, CNY, COP, CRC, CUP, CVE, CYP, CZK, DEM, DJF, DKK, DOP, DZD, ECS, ECV, EEK, EGP, ERN, ESP, ETB, EUR, FIM, FJD, FKP, FRF, GBP, GEL, GHC, GIP, GMD, GNF, GRD, GTQ, GWP, GYD, HKD, HNL, HRK, HTG, HUF, IDR, IEP, ILS, INR, IQD, IRR, ISK, ITL, JMD, JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KYD, KZT, LAK, LBP, LKR, LRD, LSL, LTL, LUF, LVL, LYD, MAD, MDL, MGF, MKD, MMK, MNT, MOP, MRO, MTL, MUR, MVR, MWK, MXN, MXV, MYR, MZM, NAD, NGN, NIO, NLG, NOK, NPR, NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PTE, PYG, QAR, ROL, RUB, RUR, RWF, SAR, SBD, SCR, SDD, SEK, SGD, SHP, SIT, SKK, SLL, SOS, SRG, STD, SVC, SYP, SZL, THB, TJR, TMM, TND, TOP, TPE, TRL, TTD, TWD, TZS, UAH, UGX, USD, USN, USS, UYU, UZS, VEB, VND, VUV, WST, XAF, XAG, XAU, XBA, XBB, XBC, XBD, XCD, XDR, XFO, XFU, XOF, XPD, XPF, XPT, XTS, XXX, YER, YUM, ZAL, ZAR, ZMK, ZRN, ZWD.

XXX is used for transactions where no currency is involved.

10.1.4 TpChargingEventInfo

Defines the structure of data elements that specifies charging event information.

Structured Member Name	Structured Member Type	Description

ChargingEventName	TpChargingEventName	The charging event for which notifications can be requested by the application
CurrentBalanceInfo	TpBalanceInfo	The current balance of the user's account
ChargingEventTime	TpTime	The time at which the charging event occurred.

10.1.5 TpChargingEventCriteria

Defines the structure of data elements that specifies charging event criteria.

Structured Member Name	Structured Member Type	Description
Users	TpAddressSet	Specifies the user(s) for which the charging events are requested to be reported.
ChargingEventName	TpChargingEventName	Specifies the specific charging event criteria used by the application to define the event required.

10.1.6 TpBalance

Defines the structure of data elements that specifies a balance.

Structured Member Name	Structured Member Type	Description
UserID	TpAddress	Specifies the user to whom the account belongs to.
StatusCode	TpBalanceQueryError	Specifies the status code for the balance query request.
BalanceInfo	TpBalanceInfo	Specifies the balance information for the user.

10.1.7 TpBalanceSet

Defines a collection of TpBalance elements.

10.1.8 TpTransactionHistory

This data type is a sequence of data elements that describes the transaction history.

Sequence Element Name	Sequence Element Type	Description
TransactionID	TpAssignmentID	Specifies the ID of the specific transaction
TimeStamp	TpDateAndTime	Specifies the date and time when the specific transaction was processed.
AdditionalInfo	TpString	Specifies a free format string providing additional information on the specific transaction. This could be the applicationDescription provided with the actual transaction.

10.1.9 TpTransactionHistorySet

Defines a collection of TpTransactionHistory elements.

10.1.10 TpTransactionHistoryStatus

Defines a status code that is reported by the Account Manager service capability feature as a result of a transaction history retrieval request.

Name	Value	Description
P_AM_TRANSACTION_OK	0	No error occurred while processing the request
P_AM_TRANSACTION_ERROR_UNSPECIFIED	1	General error, unspecified
P_AM_TRANSACTION_INVALID_INTERVAL	2	An invalid interval for the transaction history was specified.
P_AM_TRANSACTION_UNKNOWN_ACCOUNT	3	No account for the specified user is known.
P_AM_TRANSACTION_UNAUTHORIZED_APPLICATION	4	Application is not authorized to query balance.
P_AM_TRANSACTION_PROCESSING_ERROR	5	A processing error occurred while compiling the transaction history.
P_AM_TRANSACTION_SYSTEM_FAILURE	6	System failure. The request could not be handled

Annex A (normative): OMG IDL Description of Account Management SCF

The OMG IDL representation of this interface specification is contained in a text file (am.idl contained in archive 2919811IDL.ZIP) which accompanies the present document.

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for Information

3GPP TS 29.198-12 V1.0.0 (2001-03)

Technical Specification

3rd Generation Partnership Project;
Technical Specification Group Core Network;
Open Service Access;
Application Programming Interface;
Part 12: Charging;
(Release 4)

Keywords
API, OSA, IDL, CS, Charging

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

This document is part of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA). The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA API's. The concepts and the functional architecture for the Open Service Access (OSA) are described by 3GPP TS 23.127 [3]. The requirements for OSA are defined in 3GPP TS 22.127 [2].

This document specifies the Charging Service Capability Feature (SCF) aspects of the interface. All aspects of the Charging SCF are defined here, these being:

- Sequence Diagrams
- Class Diagrams
- Interface specification plus detailed method descriptions
- State Transition diagrams
- Data definitions
- IDL Description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modeling Language (UML).

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.

[1] 3GPP TS 29.198-1 "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

[4] World Wide Web Consortium Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation (www.w3.org)

[5] Wireless Application Protocol (WAP), Version 1.2, UAProf Specification (www.wapforum.org)

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the definitions in TS 29.198-1 [1] apply.

3.2 Symbols

For the purposes of the present document, the symbols in TS 29.198-1 [1] apply.

3.3 Abbreviations

For the purposes of the present document, the abbreviations in TS 29.198-1 [1] apply.

4 Charging SCF

The following sections describe each aspect of the Charging Service Capability Feature (SCF).

The order is as follows:

- The Sequence diagrams give the reader a practical idea of how each of the service capability feature is implemented.
- The Class relationships section show how each of the interfaces applicable to the SCF, relate to one another
- The Interface specification section describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the progression of internal processes either in the application, or Gateway.
- The Data definitions section show a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.

5 Sequence Diagrams

5.1 Reservation / payment in parts

The sequence diagram illustrates how to request a reservation and how to charge a user from the reserved amount, for instance to charge a user for a streamed video which lasts 10 minutes and costs a total of $2.00. The operations and interfaces that do not provide rating are employed throughout this sequence diagram.

We assume the application has already discovered the content based charging SCF. As a result, the application received an object reference pointing to an object that implements the IpChargingManager interface.

1. The application creates a local object implementing the IpAppChargingAmountSession interface.

2. The application opens a amount charging session, an reference to a new or existing object implementing IpChargingAmountSession is returned together with a unique session ID.

3. In this case a new object is used.

4. The application requests the reservation of $2.00.

5. Assuming the criteria for requesting a reservation are met (the application provider has permission to charge the requested amount, the charged user has agreed to pay the requested amount), the amount is reserved in the session. At this point, the application provider knows that the network operator will accept later debit requests up to the reserved amount. So, the application may start serving the user, for instance by sending the video stream.

6. The successful reservation is reported back to the application.

After half of the video has been sent to the user, the application may choose to capture half of the price already:

7. The application requests to debit $1.00 from the reservation.

8. The successful debit is reported back to the application.

9. The acknowledge is forwarded to the application.

10. The application checks if the remaining lifetime of the reservation will cover the remaining 5 minutes of video. Let us assume, it does not.

11. The application asks the IpChargingAmountReservation object to extend the lifetime of the reservation.

12. Assuming that the application provider is allowed to keep reservations open for longer thant 10 minutes, the extendLifeTimeReq() will be honored and confirmed properly.

13. The confirmation is forwarded to the application.

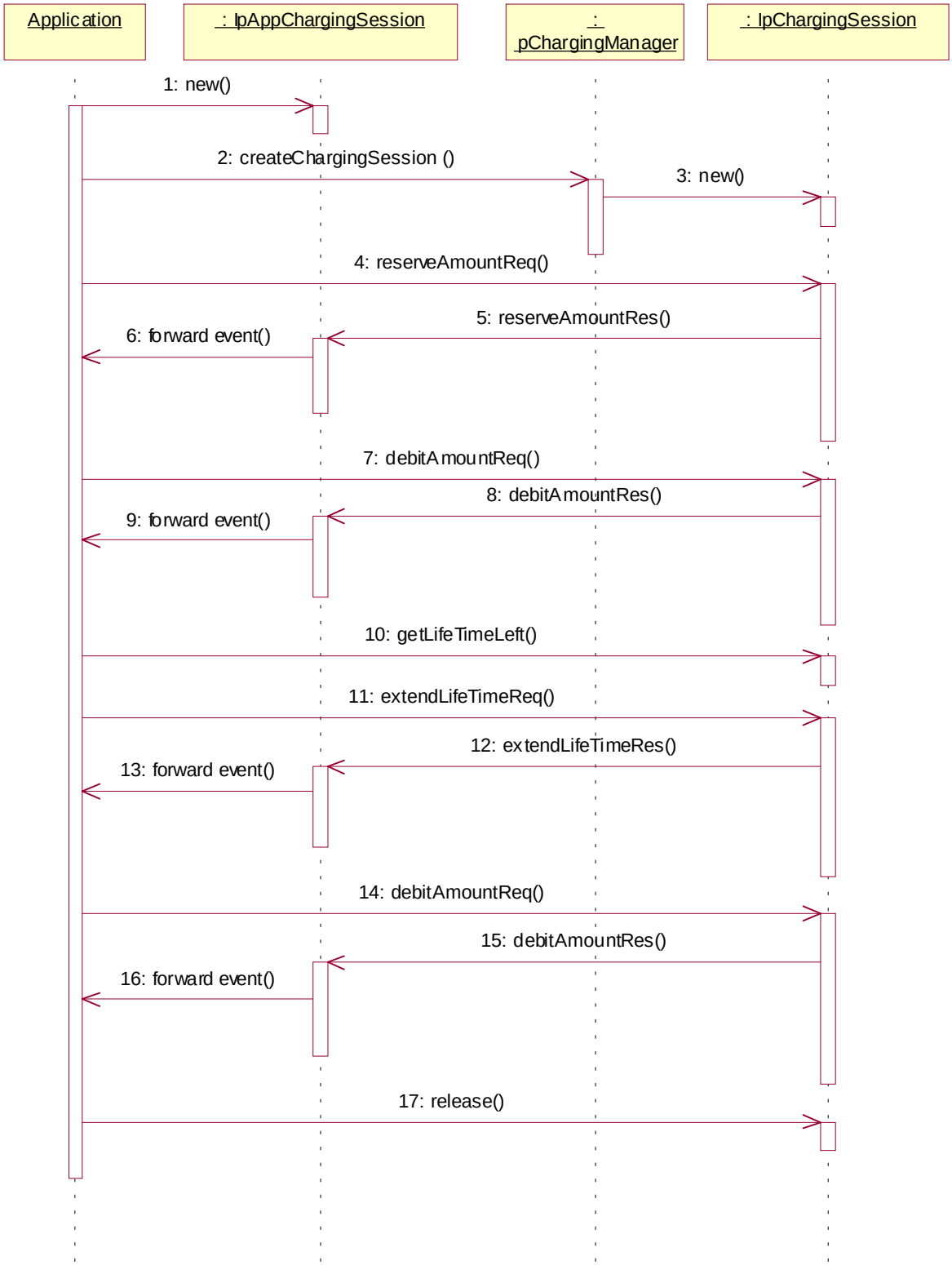
14. When the complete video has been transmitted to the user without errors, the application charges another $1.00.

15. The IpChargingAmountReservation object acknowledges the successful debit at the IpAppAmountReservation callback object.

16. The IpAppChargingAmountReservation object forwards the acknowledge to the application.

17. Since the service is complete, the application frees all resources associated with the reservation and session.

The operations which handle units are used exactly the same, except that the amount of application usage is indicated instead of a price.


Application
: IpAppChargingSession
: pChargingManager
: IpChargingSession
1: new()
2: createChargingSession ()
3: new()
4: reserveAmountReq()
5: reserveAmountRes()
6: forward event()
7: debitAmountReq()
8: debitAmountRes()
9: forward event()
10: getLifeTimeLeft()
11: extendLifeTimeReq()
12: extendLifeTimeRes()
13: forward event()
14: debitAmountReq()
15: debitAmountRes()
16: forward event()
17: release()

5.2 Immediate Charge

This sequence diagram illustrates how immediate charging is used. Assume a WAP gateway that charges the user $0.01 per requested URL. Since it is acceptable to loose one tick worth $0.01, no prior reservations are made. The WAP gateway sends an immediate debit for each requested URL, and should a payment have as result failure, the user is disconnected.

1. The application creates a local object implementing the IpAppChargingAmountSession interface. This object will receive response messages from the IpChargingAmountSession object.

2. The application orders the creation of a session.

3. The application requests to charge the user $0.01.

4. The payment is acknowledged.

5. The acknowledgement is forwarded in the application.

6. The application requests to charge the user $0.01.

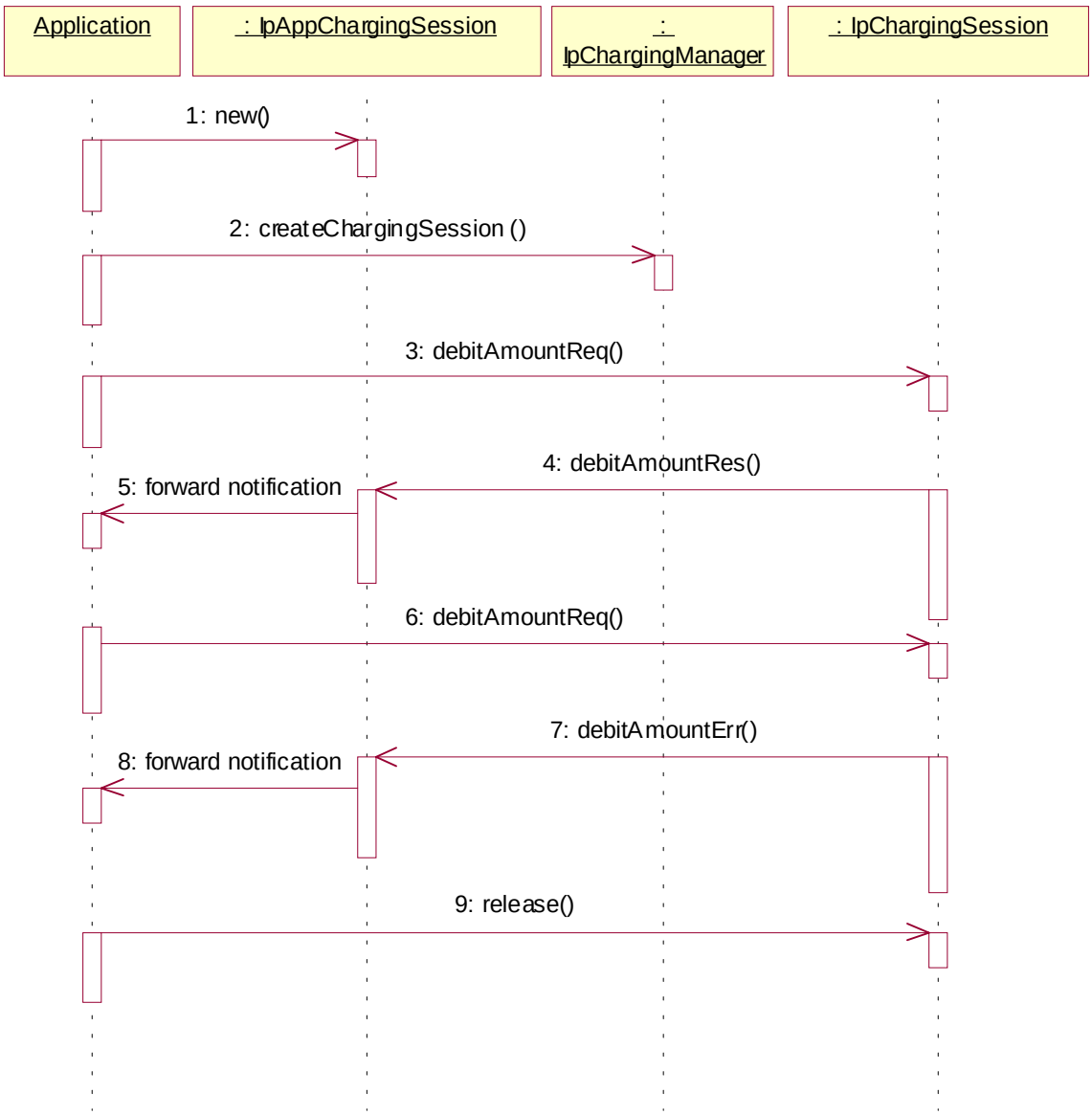
7. The payment is reported to fail.

8. The failure report is forwarded in the application.

(repeat steps 3 - 5 and 6 - 8 as long as you want to in any order you want to)

9. The application releases the session

The operations which handle units are used exactly the same, except that the amount of application usage is indicated instead of a price.


Application
: IpAppChargingSession
: IpChargingManager
: IpChargingSession
1: new()
2: createChargingSession ()
3: debitAmountReq()
4: debitAmountRes()
5: forward notification
6: debitAmountReq()
7: debitAmountErr()
8: forward notification
9: release()

6 Class Diagrams

This class diagram shows the application interfaces for charging and their relations to the service interfaces.

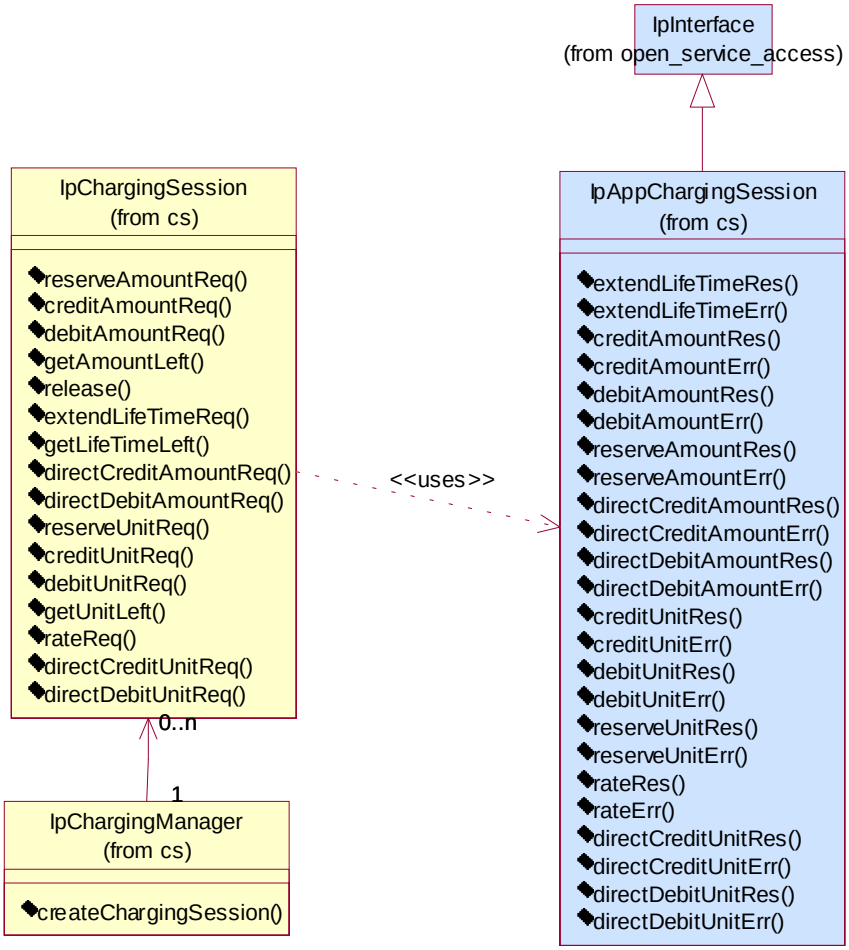


Figure: Application Interfaces

This class diagram shows the interfaces of the charging SCF.

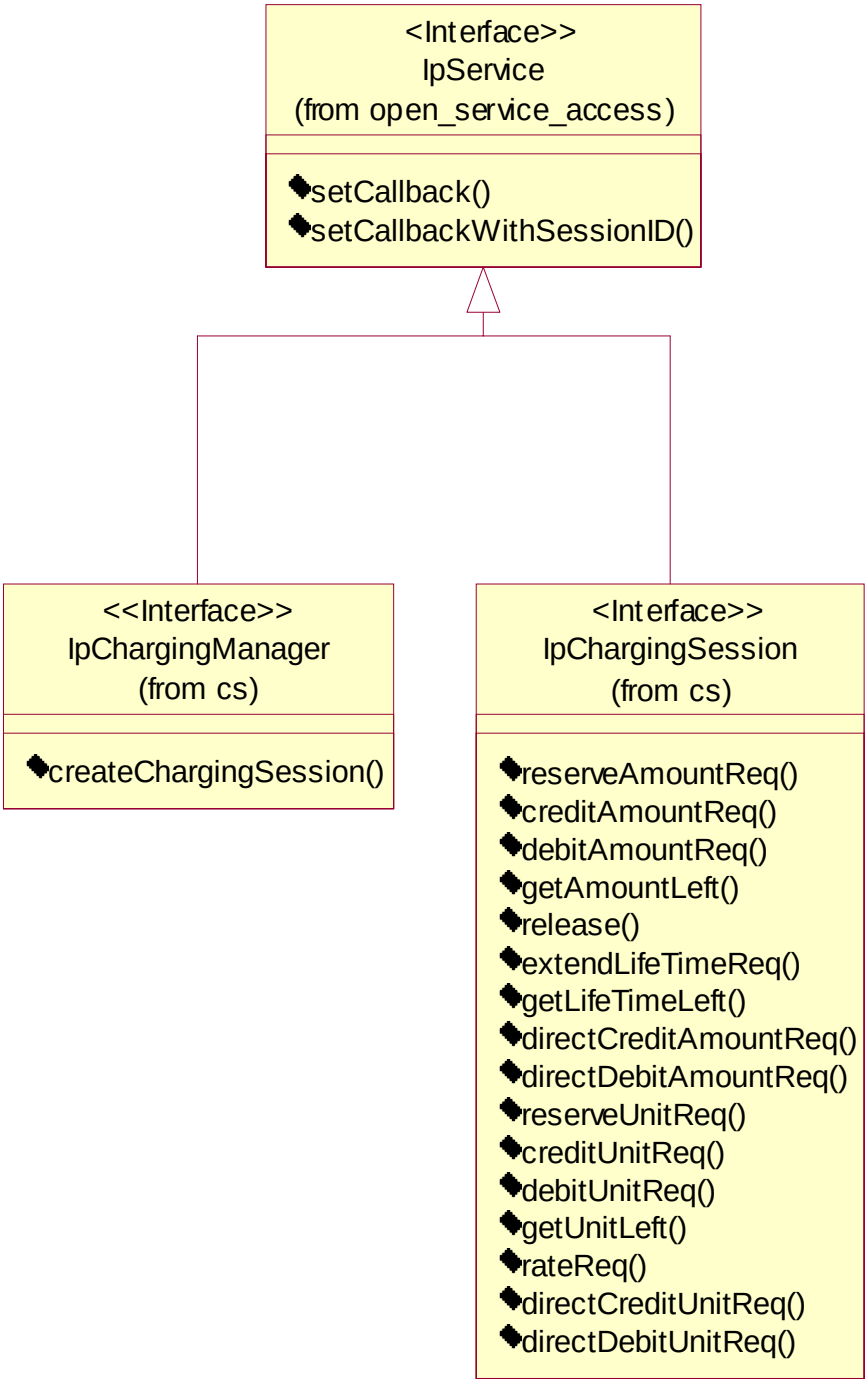


Figure: Service Interfaces

7 The Service Interface Specifications

7.1 Interface Specification Format

This section defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

7.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for enterprise-based client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>

7.1.2 Method descriptions

Each method (API method “call”) is described. All methods in the API return a value of type `TpResult`, indicating, amongst other things, if the method invocation was sucessfully executed or not.

Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a '`Req`' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a '`Res`' or '`Err`' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name> or IpSvc<name>` interfaces to provide the callback mechanism.

7.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.

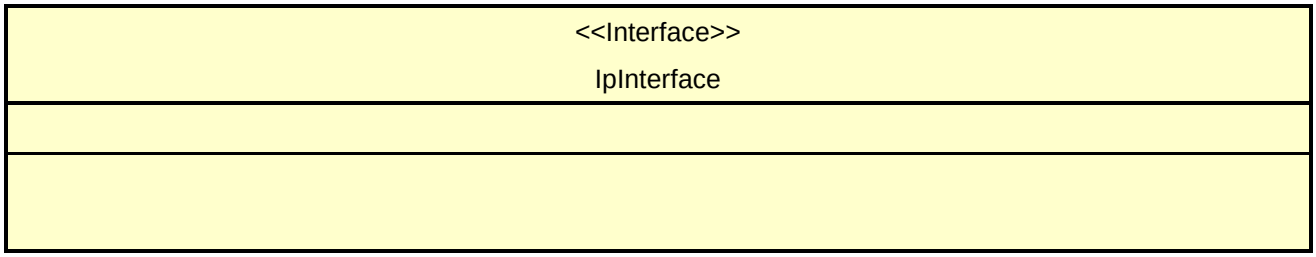
7.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

7.2 Base Interface

7.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.



7.3 Service Interfaces

7.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

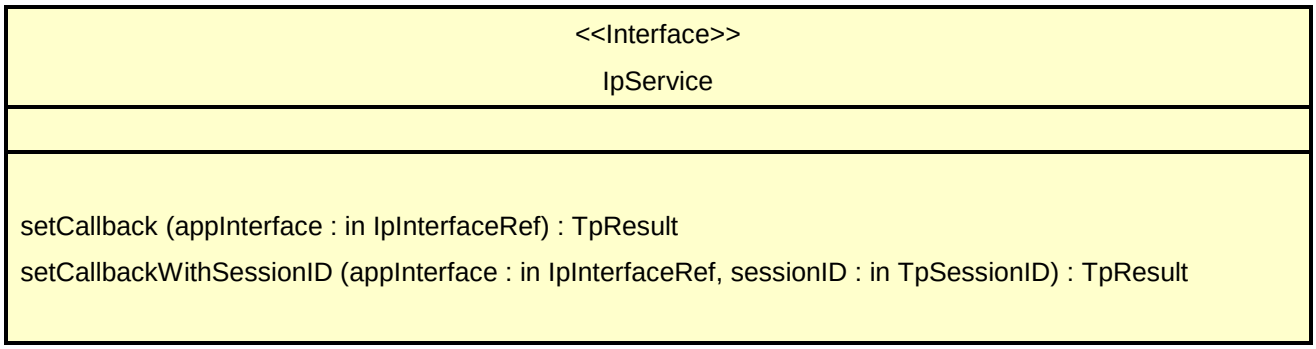
The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

7.4 Generic Service Interface

7.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.



Method
setCallback()

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

Raises

TpGeneralException

Method
setCallbackWithSessionID()

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

sessionID : in TpSessionID

Specifies the session for which the service can invoke the application's callback interface.

Raises

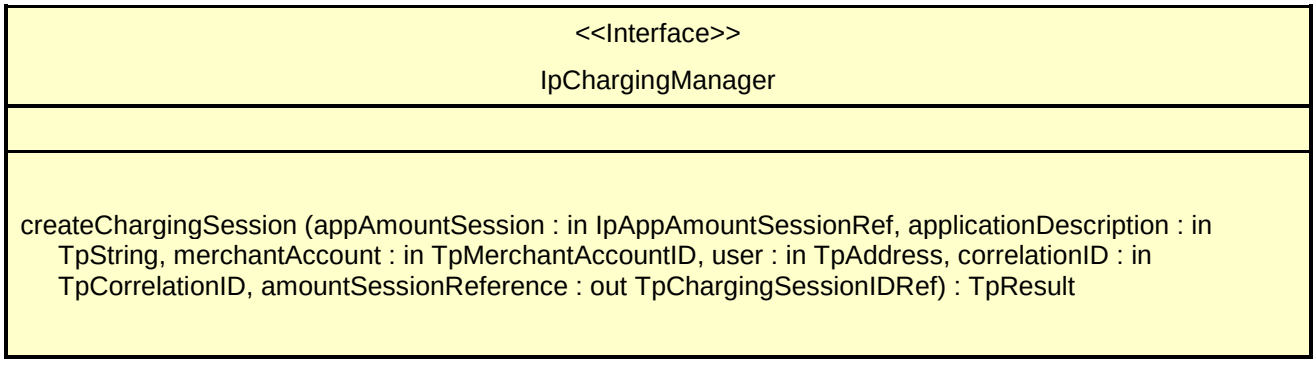
TpGeneralException

8 Charging Interface Classes

8.1 Interface Class IpChargingManager

Inherits from: IpService.

This interface is the 'service manager' interface for the Charging Service. The Charging manager interface provides management functions to the charging service. The application programmer can use this interface to start charging sessions.



Method

createChargingSession()

This method creates a container for further transactions between an end user and a merchant (either direct or via a reservation).

Parameters

appAmountSession : in IpAppAmountSessionRef

Callback interface for the session in the application

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user).

merchantAccount : in TpMerchantAccountID

Identifies the account of the party providing the application to be used.

`user : in TpAddress`

Specifies the user that is using the application. This may or may not be the user that will be charged. The Charging service will determine the charged user.

`correlationID : in TpCorrelationID`

This value can be used to correlate the charging to network activity.

`amountSessionReference : out TpChargingSessionIDRef`

Defines the session.

8.2 Interface Class IpAppChargingSession

Inherits from: IpInterface.

This application interface must be implemented by the client application to handle callbacks from the IpChargingSession.

<<Interface>> IpAppChargingSession
extendLifeTimeRes (sessionID : in TpSessionID, SessionTimeLeft : in TpInt32) : TpResult extendLifeTimeErr (sessionID : in TpSessionID, error : in TpChargingError) : TpResult creditAmountRes (sessionID : in TpSessionID, creditedAmount : in TpChargingPrice, reservedAmountLeft : in TpChargingPrice, requestNumberNextRequest : in TpInt32) : TpResult creditAmountErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult debitAmountRes (sessionID : in TpSessionID, debitedAmount : in TpChargingPrice, reservedAmountLeft : in TpChargingPrice, requestNumberNextRequest : in TpInt32) : TpResult debitAmountErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult reserveAmountRes (sessionID : in TpSessionID, reservedAmount : in TpChargingPrice, sessionTimeLeft : in TpInt32, requestNumberNextRequest : in TpInt32) : TpResult reserveAmountErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult directCreditAmountRes (sessionID : in TpSessionID, creditedAmount : in TpChargingPrice, requestNumberNextRequest : in TpInt32) : TpResult directCreditAmountErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult directDebitAmountRes (sessionID : in TpSessionID, debitedAmount : in TpChargingPrice, requestNumberNextRequest : in TpInt32) : TpResult directDebitAmountErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult creditUnitRes (sessionID : in TpSessionID, creditedVolumes : in TpVolumeSet, reservedUnitsLeft : in TpVolumeSet, requestNumberNextRequest : in TpInt32) : TpResult creditUnitErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult debitUnitRes (sessionID : in TpSessionID, debitedVolumes : in TpVolumeSet, reservedUnitsLeft : in

TpVolumeSet, requestNumberNextRequest : in TpInt32) : TpResult

debitUnitErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult

reserveUnitRes (sessionID : in TpSessionID, reservedUnits : in TpVolumeSet, sessionTimeLeft : in TpInt32, requestNumberNextRequest : in TpInt32) : TpResult

reserveUnitErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult

rateRes (sessionID : in TpSessionID, rates : in TpPriceVolumeSet, validityTimeLeft : in TpInt32) : TpResult

rateErr (sessionID : in TpSessionID, error : in TpChargingError) : TpResult

directCreditUnitRes (sessionID : in TpSessionID, creditedVolumes : in TpVolumeSet, requestNumberNextRequest : in TpInt32) : TpResult

directCreditUnitErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult

directDebitUnitRes (sessionID : in TpSessionID, debitedVolumes : in TpVolumeSet, requestNumberNextRequest : in TpInt32) : TpResult

directDebitUnitErr (sessionID : in TpSessionID, error : in TpChargingError, requestNumberNextRequest : in TpInt32) : TpResult

Method

extendLifeTimeRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

SessionTimeLeft : in TpInt32

Indicates the number of seconds that the session remains valid.

Method

extendLifeTimeErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

Method
creditAmountRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

creditedAmount : in TpChargingPrice

Indicates the credited amount.

reservedAmountLeft : in TpChargingPrice

The amount left of the reservation.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method
creditAmountErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method
debitAmountRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

debitedAmount : in TpChargingPrice

Indicates the debited amount.

reservedAmountLeft : in TpChargingPrice

The amount left of the reservation.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

debitAmountErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

reserveAmountRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the same as the session ID returned in the request.

reservedAmount : in TpChargingPrice

The amount reserved.

sessionTimeLeft : in TpInt32

Indicates the number of seconds that the session and the reservation therein remains valid.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

reserveAmountErr()

This method indicates that the corresponding request failed. The reservation cannot be used.

Parameters

sessionID : in TpSessionID

This is the same as the session ID returned in the request.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

directCreditAmountRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

creditedAmount : in TpChargingPrice

Indicates the credited amount.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

directCreditAmountErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method
directDebitAmountRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

debitedAmount : in TpChargingPrice

Indicates the debited amount.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method
directDebitAmountErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method
creditUnitRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

creditedVolumes : in TpVolumeSet

Indicates the credited volumes of application useage.

reservedUnitsLeft : in TpVolumeSet

The volume of application usage left in the reservation.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

creditUnitErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

debitUnitRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

debitedVolumes : in TpVolumeSet

Indicates the debited volumes of application useage.

reservedUnitsLeft : in TpVolumeSet

The volume of application usage left in the reservation.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

debitUnitErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

reserveUnitRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the same as the session ID returned in the request.

reservedUnits : in TpVolumeSet

The volume of application usage reserved.

sessionTimeLeft : in TpInt32

Indicates the number of seconds that the session and the reservation therein remains valid.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method

reserveUnitErr()

This method indicates that the corresponding request failed. The reservation cannot be used.

Parameters

sessionID : in TpSessionID

This is the same as the session ID returned in the request.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method
rateRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

rates : in TpPriceVolumeSet

The applicable rates.

validityTimeLeft : in TpInt32

Indicates the number of seconds that this information remains valid.

Method
rateErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

Method
directCreditUnitRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

creditedVolumes : in TpVolumeSet

Indicates the credited volumes of application useage.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method
directCreditUnitErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method
directDebitUnitRes()

This method indicates that the corresponding request was successfull.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

debitedVolumes : in TpVolumeSet

Indicates the debited volumes of application useage.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

Method
directDebitUnitErr()

This method indicates that the corresponding request failed.

Parameters

sessionID : in TpSessionID

This is the ID of the session for which the operation was called.

error : in TpChargingError

Indicates the reason for failure.

requestNumberNextRequest : in TpInt32

This request number must be used in the next request for this session.

8.3 Interface Class IpChargingSession

Inherits from: IpService.

The Charging Amount Session interface provides operations to facilitate amount transactions between a merchant and an user. The application programmer can use this interface to debit or credit amounts towards a user, to create and extend the lifetime of a reservation and to get information about what is left of the reservation.

<<Interface>> IpChargingSession
reserveAmountReq (sessionID : in TpSessionID, preferredAmount : in TpChargingPrice, minimumAmount : in TpChargingPrice, applicationDescription : in TpString, requestNumber : in TpInt32, chargingParameters : in TpChargingParameterSet) : TpResult creditAmountReq (sessionID : in TpSessionID, applicationDescription : in TpString, amount : in TpChargingPrice, closeReservation : in TpBoolean, requestNumber : in TpInt32) : TpResult debitAmountReq (sessionID : in TpSessionID, applicationDescription : in TpString, amount : in TpChargingPrice, closeReservation : in TpBoolean, requestNumber : in TpInt32) : TpResult getAmountLeft (sessionID : in TpSessionID, amountLeft : out TpChargingPrice) : TpResult release (sessionID : in TpSessionID, requestNumber : in TpInt32) : TpResult extendLifeTimeReq (sessionID : in TpSessionID) : TpResult getLifeTimeLeft (sessionID : in TpSessionID, reservationTimeLeft : out TpInt32) : TpResult directCreditAmountReq (sessionID : in TpSessionID, applicationDescription : in TpString, chargingParameters : in TpChargingParameterSet, amount : in TpChargingPrice, requestNumber : in TpInt32) : TpResult directDebitAmountReq (sessionID : in TpSessionID, applicationDescription : in TpString, amount : in TpChargingPrice, requestNumber : in TpInt32, chargingParameters : in TpChargingParameterSet) : TpResult reserveUnitReq (sessionID : in TpSessionID, chargingParameters : in TpChargingParameterSet, volumes : in TpVolumeSet, applicationDescription : in TpString, requestNumber : in TpInt32) : TpResult creditUnitReq (sessionID : in TpSessionID, applicationDescription : in TpString, volumes : in TpVolumeSet, closeReservation : in TpBoolean, requestNumber : in TpInt32) : TpResult debitUnitReq (sessionID : in TpSessionID, applicationDescription : in TpString, volumes : in TpVolumeSet, closeReservation : in TpBoolean, requestNumber : in TpInt32) : TpResult getUnitLeft (sessionID : in TpSessionID, volumesLeft : out TpVolumeSet) : TpResult rateReq (sessionID : in TpSessionID, chargingParameters : in TpChargingParameterSet) : TpResult directCreditUnitReq (sessionID : in TpSessionID, applicationDescription : in TpString, chargingParameters : in TpChargingParameterSet, volumes : in TpVolumeSet, requestNumber : in TpInt32) : TpResult directDebitUnitReq (sessionID : in TpSessionID, applicationDescription : in TpString, chargingParameters : in TpChargingParameterSet, volumes : in TpVolumeSet, requestNumber : in TpInt32) : TpResult

Method

`reserveAmountReq()`

This method is used whan an application wants to reserve an amount for services to be delivered to an user.

Parameters

sessionID : in TpSessionID

The ID of the session.

preferredAmount : in TpChargingPrice

The amount that the application wants to be reserved.

minimumAmount : in TpChargingPrice

The minimum amount that can be used by the application.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

chargingParameters : in TpChargingParameterSet

Method

creditAmountReq()

This method credits an amount towards the reservation associated with the session.

The amount left in the reservation will be increased by this amount.

Each request to debit / credit an amount towards a reservation is handled separately. For example, two requests for a payment of EUR 1,- will give a total payment of EUR 2,-.

A credit of EUR 1,- and a debit of EUR 1 will give a total payment of EUR 0,-.

Parameters

sessionID : in TpSessionID

The ID of the session.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

amount : in TpChargingPrice

The amount to be credited towards the user.

closeReservation : in TpBoolean

If set to true, this parameter indicates that the rest of the reservation can be freed.

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

Method
debitAmountReq()

This method debits an amount from the reservation.

The amount left in the reservation will be decreased by this amount.

Each request to debit / credit an amount towards a reservation is handled separately. For example, two requests for a payment of EUR 1,- will give a total payment of EUR 2,-.

A credit of EUR 1,- and a debit of EUR 1 will give a total payment of EUR 0,-.

When a debit operation would exceed the limit of the reservation, the debit operation fails.

Parameters

sessionID : in TpSessionID

The ID of the session.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

amount : in TpChargingPrice

The amount to be debited from the user.

closeReservation : in TpBoolean

If set to true, this parameter indicates that the reservation can be freed.

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

Method
getAmountLeft()

With this method an application can request the remaining amount of the reservation.

Parameters

sessionID : in TpSessionID

The ID of the session.

amountLeft : out TpChargingPrice

Gives the amount left in the reservation.

Method
release()

This method releases the session, no operations can be done towards this session anymore (not even retries). Unused parts of a reservation are freed.

Parameters

sessionID : in TpSessionID

The ID of the session.

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

Method

extendLifeTimeReq()

With this method an application can request the lifetime of the session and possible reservation therein to be extended.

Parameters

sessionID : in TpSessionID

The ID of the session.

Method

getLifeTimeLeft()

With this method an application can request the remaining lifetime of the session.

Parameters

sessionID : in TpSessionID

The ID of the session.

reservationTimeLeft : out TpInt32

Indicates the number of seconds that the session remains valid.

Method

directCreditAmountReq()

This method directly credits an amount towards the user.

A possible reservation associated with this session is not influenced.

Parameters

sessionID : in TpSessionID

The ID of the session.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

chargingParameters : in TpChargingParameterSet

These parameters and their values specify to the charging service what was provided to the end user so that the charging service can determine the applicable tariff..

amount : in TpChargingPrice

The amount to be credited towards the user.

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

Method

directDebitAmountReq()

This method directly debits an amount towards the user.

A possible reservation associated with this session is not influenced.

Parameters

sessionID : in TpSessionID

The ID of the session.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

amount : in TpChargingPrice

The amount to be debited from the user.

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

chargingParameters : in TpChargingParameterSet

These parameters and their values specify to the charging service what was provided to the end user so that the charging service can determine the applicable tariff..

Method

reserveUnitReq()

This method is used whan an application wants to reserve volumes of application usage to be delivered to an user in the session.

Parameters

sessionID : in TpSessionID

The ID of the session.

chargingParameters : in TpChargingParameterSet

These parameters and their values specify to the charging service what was provided to the end user so that the charging service can determine the applicable tariff..

volumes : in TpVolumeSet

Specifies the units and number of units for each unit that will be reserved.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

Method

creditUnitReq()

This method credits a volume of application usage towards the reservation.

The volumes left in the reservation of this will be increased by this amount.

Each request to debit / credit a volume towards a reservation is handled separately. For example, two requests for a payment for 10 kilobytes will give a total payment for 20 kilobytes

Parameters

sessionID : in TpSessionID

The ID of the session.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

volumes : in TpVolumeSet

Specifies the units and number of units for each unit which will be credited towards the user.

closeReservation : in TpBoolean

If set to true, this parameter indicates that the reservation can be freed.

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

Method

debitUnitReq()

This method debits a volume of application usage from the reservation.

The volumes left in the reservation will be decreased by this amount.

Each request to debit / credit a volume towards a reservation is handled separately. For example, two requests for a payment for 10 kilobytes will give a total payment for 20 kilobytes.

When a debit operation would exceed the limit of the reservation, the debit operation succeeds, and the debited volumes will be the rest of the volumes in the reservation.

Parameters

sessionID : in TpSessionID

The ID of the session.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

volumes : in TpVolumeSet

Specifies the units and number of units for each unit which will be debited from the user.

closeReservation : in TpBoolean

If set to true, this parameter indicates that the reservation can be freed.

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

Method

getUnitLeft()

With this method an application can request the remaining amount of the reservation.

Parameters

sessionID : in TpSessionID

The ID of the session.

volumesLeft : out TpVolumeSet

Specifies the units and number of units for each unit which can still be debited from the user.

Method

rateReq()

This method is used when the application wants to have an item rated by the charging service. The result can be used to present pricing information to the end user before the end user actually want to start using the service.

Parameters

sessionID : in TpSessionID

The ID of the session.

chargingParameters : in TpChargingParameterSet

These parameters and their values specify to the charging service what was provided to the end user so that the charging service can determine the applicable tariff..

Method

directCreditUnitReq()

This method directly credits a volume of application usage towards the user.

The volumes in a possible reservation associated with this session are not influenced.

Parameters

sessionID : in TpSessionID

The ID of the reservation.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

chargingParameters : in TpChargingParameterSet

These parameters and their values specify to the charging service what was provided to the end user so that the charging service can determine the applicable tariff..

volumes : in TpVolumeSet

Specifies the units and number of units for each unit which will be credited towards the user.

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

Method

directDebitUnitReq()

This method directly credits a volume of application usage towards the user.

The volumes in a possible reservation associated with this session are not influence.

Parameters

sessionID : in TpSessionID

The ID of the reservation.

applicationDescription : in TpString

Descriptive text for informational puposes (e.g. text presented on the bill and used in communication towards the user)

chargingParameters : in TpChargingParameterSet

These parameters and their values specify to the charging service what was provided to the end user so that the charging service can determine the applicable tariff..

volumes : in TpVolumeSet

Specifies the units and number of units for each unit which will be debited from the user.

requestNumber : in TpInt32

Specifies the number given in the result of the previous operation on this session, or when creating the session. When no answer is received the same operation with the same parameters must be retried with the same requestNumber.

9 State Transition Diagrams

There are no State Transition Diagrams for the Charging SCF.

10 Data Definitions

10.1 Charging Data Definitions

This section provides the Charging specific data definitions necessary to support the OSA interface specification.

This document is written using Hypertext link, to aid navigation through the data structures. Underlined text represents Hypertext links.

The general format of a data definition specification is the following:

- Data type, that shows the name of the data type.
- Description, that describes the data type.
- Tabular specification, that specifies the data types and values of the data type.
- Example, if relevant, shown to illustrate the data type.

TpMerchantAccountID

Defines a Sequence of Data Elements that defines the used service.

Sequence Element Name	Sequence Element Type
MerchantID	TpString
AccountID	TpInt32

TpCorrelationID

Defines the Sequence of Data Elements that identify a correlation.

Sequence Element Name	Sequence Element Type
CorrelationID	TpSessionID
CorrelationType	TpCorrelationType

TpCorrelationType

Defines the type of correlation.

Name	Value	Description
P_CHS_CORRELATION_UNDEFINED	0	Unknown correlation type.
P_CHS_CORRELATION_VOICE	1	Voice Call
P_CHS_CORRELATION_DATA	2	Data Session
P_CHS_CORRELATION_MM	3	Multi Media Session
P_CHS_CORRELATION_RUNTIME	0x10000000-0xFFFFFFFF	Parameters with values greater than P_CHS_CORRELATION_RUNTIME may be added run-time.

TpChargingPrice

Defines the Sequence of Data Elements that identify a price.

Sequence Element Name	Sequence Element Type
Currency	TpString
Amount	TpAmount

Currencies as defined by ISO 4217:1995.

TpAmount

Defines the Sequence of Data Elements that define an amount in integers as “Number * 10 ^ Exponent” (i.e. if Number = 6543 and Exponent = -2 then the amount is 65,43). This representation avoids unwanted rounding off.

Sequence Element Name	Sequence Element Type
Number	TpInt32
Exponent	TpInt32

TpChargingParameterSet

Defines a Numbered Set of Data Elements of TpChargingParameter

TpChargingParameter

Defines a Sequence of Data Elements that defines the used service.

Sequence Element Name	Sequence Element Type
ParameterID	TpChargingParameterID
ParameterValue	TpChargingParameterValue

TpChargingParameterID

Defines the type of charging parameter.

Name	Value	Description
P_CHS_PARAM_UNDEFINED	0	Unknown parameter
P_CHS_PARAM_ITEM	1	Parameter represents kind of service delivered to the end user
P_CHS_PARAM_SUBTYPE	2	Parameter represents subtype / operation of service delivered to the end user
P_CHS_PARAM_RESULT	3	Parameter represents the result of the service

P_CHS_PARAM_RUNTIME	0x10000000-0xFFFFFFFF	Parameters with values greater than P_CHS_PARAM_RUNTIME may be added run-time.

TpChargingParameterValue

Defines the Tagged Choice of Data Elements that identify a charging parameter.

	Tag Element Type	
	TpChargingParameterValueType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CHS_PARAMETER_INT32	TpInt32	IntValue
P_CHS_PARAMETER_FLOAT	TpFloat	FloatValue
P_CHS_PARAMETER_STRING	TpString	StringValue
P_CHS_PARAMETER_BOOLEAN	TpBoolean	BooleanValue

TpChargingParameterValueType

Defines the type of charging parameter.

Name	Value	Description
P_CHS_PARAMETER_INT32	0	Parameter represented by a TpInt32
P_CHS_PARAMETER_FLOAT	1	Parameter represented by a TpFloat
P_CHS_PARAMETER_STRING	2	Parameter represented by a TpString
P_CHS_PARAMETER_BOOLEAN	3	Parameter represented by a TpBoolean

TpVolumeSet

Defines the Numbered Set of Data Elements that describes list TpVolume.

TpVolume

Defines a volume.

Sequence Element Name	Sequence Element Type
Amount	TpAmount
Unit	TpUnitID

TpUnitID

Defines the unit that is used in a TpVolume.

Name	Value	Description
P_CHS_UNIT_UNDEFINED	0	Undefined
P_CHS_UNIT_NUMBER	1	number of times / events
P_CHS_UNIT_OCTETS	2	unit is octets
P_CHS_UNIT_SECONDS	3	unit is seconds
P_CHS_UNIT_MINUTES	4	unit is minutes
P_CHS_UNIT_HOURS	5	unit is hours

P_CHS_UNIT_DAYS	6	unit is days
P_CHS_UNIT_RUNTIME	0x10000000-0xFFFFFFFF	unit is defined runtime

TpChargingSessionID

Defines the Sequence of Data Elements that unambiguously specify the Charging Session object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
ChargingSessionReference	IpChargingSessionRef	This element specifies the interface reference for the charging session object.
ChargingSessionID	TpSessionID	This element specifies the session ID for the charging session.
RequestNumberNextRequest	TpInt32	This element specifies the request number to use for the next request.

TpPriceVolumeSet

Defines a Numbered Set of Data Elements of TpPriceVolume.

TpPriceVolume

Defines the Sequence of Data Elements that identify a price for a volume.

Sequence Element Name	Sequence Element Type
Price	TpChargingPrice
Volume	TpVolume

TpChargingError

Indicates the error that occured.

Name	Value	Description
P_CHS_ERR_UNDEFINED	0	Generic error
P_CHS_ERR_ACCOUNT	1	Merchant account unknown
P_CHS_ERR_USER	2	Unknown user
P_CHS_ERR_PARAMETER	3	The set of charging parameters contains an unknown parameter, or a required parameter is missing.
P_CHS_ERR_NO_DEBIT	4	For some reason the application is not allowed to get money from this user.
P_CHS_ERR_NO_CREDIT	5	For some reason the application is not alllowed to pay this user.
P_CHS_ERR_VOLUMES	6	Required volumes are missing.
P_CHS_ERR_CURRENCY	7	This currency is not supported for this transaction.
P_CHS_ERR_NO_EXTEND	8	Request to extend the liftetime of a reservation is rejected.
P_CHS_RESERVATION_LIMIT	9	This amount or volume violates the bounds of the reservation

Annex A (normative): OMG IDL Description of Charging SCF

The OMG IDL representation of this interface specification is contained in a text file (cs.idl contained in archive 2919812IDL.ZIP) which accompanies the present document.

History

Document history		
1.0.0	10 March 2001	Submitted by CN5 to CN#11 for Information