3GPP TSG CN Plenary Meeting #12 **Tdoc NP-010330**
Stockholm, Sweden, 13th - 15th June 2001

Source: CN5

Title: Rel4 CRs to Open Service Architecture (OSA); Application Programming Interface (API) (29.198-1, 2, -3, -5, -6, -7, -8)

Agenda item: **8.5 OSA enhancements [OSA1]**

Document for: APPROVAL

Doc-1st-Level	Doc-2nd-Level	Spec	CR	Rev	Phase	Subject	Cat	Version-Current	Version-New	Meeting-2nd-Level	Workitem
NP-010330	N5-010267	29.198-1	001		Rel4	**Corrections to OSA API Rel4**	F	4.0.0	4.1.0	N5-11	OSA1
NP-010330	N5-010261	29.198-2	001		Rel4	**Corrections to OSA API Rel4**	F	4.0.0	4.1.0	N5-11	OSA1
NP-010330	N5-010262	29.198-3	001		Rel4	**Corrections to OSA API Rel4**	F	4.0.0	4.1.0	N5-11	OSA1
NP-010330	N5-010263	29.198-5	001		Rel4	**Corrections to OSA API Rel4**	F	4.0.0	4.1.0	N5-11	OSA1
NP-010330	N5-010264	29.198-6	001		Rel4	**Corrections to OSA API Rel4**	F	4.0.0	4.1.0	N5-11	OSA1
NP-010330	N5-010265	29.198-7	001		Rel4	**Corrections to OSA API Rel4**	F	4.0.0	4.1.0	N5-11	OSA1
NP-010330	N5-010266	29.198-8	001		Rel4	**Corrections to OSA API Rel4**	F	4.0.0	4.1.0	N5-11	OSA1

3GPP TSG_CN5 (Open Service Access – OSA) **Tdoc N5-010267**
Meeting #11, San Diego, CA, USA, 21 – 24 May 2001

CR-Form-v3

CHANGE REQUEST

⌘ **29.198-1** **CR** **001** ⌘ rev **-** ⌘ Current version: **4.0.0** ⌘

*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:	⌘	Corrections to OSA API Rel4
Source:	⌘	CN5
Work item code:	⌘	OSA1
Date:	⌘	07/06/2001
Category:	⌘	**F**
Release:	⌘	Rel4

Use one of the following categories:
F *(essential correction)*
A *(corresponds to a correction in an earlier release)*
B *(Addition of feature),*
C *(Functional modification of feature)*
D *(Editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
REL-4	*(Release 4)*
REL-5	*(Release 5)*

Reason for change:	⌘	Correction to IDL namespace to align with that of ETSI and Parlay equivalent APIs
Summary of change:	⌘	Change org.open_service_access root namespace to org.csapi
Consequences if not approved:	⌘	If not agreed, 3GPP OSA will use a different namespace than ETSI and Parlay. This will create inter-working difficulties between applications written using IDL in 29.198, and gateway servers written using Parlay or ETSI IDL, and vice versa.

Clauses affected:	⌘			
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘			

6.10 Prefixes

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

6.11 Naming space across CORBA modules

The following shows the naming space used in this specification.

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

CR-Form-v3

CHANGE REQUEST

⌘ **29.198-2** **CR** **001** ⌘ rev **-** ⌘ Current version: **4.0.0** ⌘

*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:	⌘	Corrections to OSA API Rel4
Source:	⌘	CN5
Work item code:	⌘	OSA1
Date:	⌘	07/06/2001
Category:	⌘	**F**
Release:	⌘	Rel4

Use one of the following categories:
F *(essential correction)*
A *(corresponds to a correction in an earlier release)*
B *(Addition of feature),*
C *(Functional modification of feature)*
D *(Editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
REL-4	*(Release 4)*
REL-5	*(Release 5)*

Reason for change:	⌘	Exception handling mechanism in 29.198 requires correction to enable it to be correctly used, without ambiguity
Summary of change:	⌘	Replace TpGeneralException and TpResultInfo with detailed exception classes which can be thrown for each method
Consequences if not approved:	⌘	29.198 will be ambiguous and difficult to implement correctly - inter-working might be jeopardised.

Clauses affected:	⌘				
Other specs affected:	⌘	**X**	Other core specifications	⌘	All other parts of 29.198 except part 1 have similar changes
		☐	Test specifications		
		☐	O&M Specifications		
Other comments:	⌘				

5.4 Method Result Data definitions

5.4.1 TpResult

Defines the **Error! Reference source not found.** 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 shall 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

Name	Value	Description
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.
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.~~

Name	Value	Description
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 [4]. 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: 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 [4]. 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 Co-ordinated (UTC). Otherwise, local time is assumed.

EXAMPLE: 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 [4]. 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 Co-ordinated (UTC). Otherwise, local time is assumed.

EXAMPLE: 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 **Error! Reference source not found.** 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.

5.5.6 TpTimeInterval

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

Sequence Element Name	Sequence Element Type
StartTime	TpDateAndTime
StopTime	TpDateAndTime

5.8 Exception Classes

5.8.1 TpCommonExceptions

Defines the structure of the exception class which is applicable to all methods.

Structure Element Name	Structure Element Type	Structure Element Description
exceptionType	TpInt32	Carries a constant from the list in the table below
extraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name

5.8.2 Constants associated with TpCommonExceptions

Name	Value	Description
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_NO_CALLBACK_ADDRESS_SET	0011h	The requested method is refused because no callback address is set
P_METHOD_NOT_SUPPORTED	0016h	The method is not allowed or supported within the context of the current service agreement.
P_INVALID_STATE	0306h	Unexpected sequence of methods, i.e., the sequence does not match the specified state diagrams.

5.8.3 Exceptions available to all methods on all interfaces

The following are the list of exception classes which are available to all interfaces of the API.

Name	Description
P_APPLICATION_NOT_ACTIVATED	An application is unauthorised to access information and request services with regards to users that have deactivated that particular application.
P_INFORMATION_NOT_AVAILABLE	An application is unauthorised to access information and request services with regards to users that have set their privacy flag regarding that particular service.
P_INVALID_ADDDRESS	Invalid address specified

Name	Description
P_INVALID_AMOUNT	Invalid amount specified.
P_INVALID_ASSIGNMENT_ID	The assignment ID is invalid
P_INVALID_CRITERIA	Invalid criteria specified
P_INVALID_CURRENCY	Invalid currency specified.
P_INVALID_EVENT_TYPE	Invalid event type
P_INVALID_INTERFACE_NAME	Invalid interface name
P_INVALID_INTERFACE_TYPE	The interface reference supplied by the client is the wrong type.
P_INVALID_NETWORK_STATE	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_INVALID_SESSION_ID	Invalid session ID.
P_INVALID_TIME_AND_DATE_FORMAT	Invalid date and time format provided
P_SET_LENGTH_EXCEEDED	The maximum set size is exceeded in a method parameter value.
P_UNKNOWN_SUBSCRIBER	An application is unauthorised to access information and request services with regards to users that are not subscribed to the application.

CR-Form-v3

CHANGE REQUEST

⌘ 29.198-3 CR 001 ⌘ rev - ⌘ Current version: 4.0.0 ⌘

*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 ☒

Title:	⌘	Corrections to OSA API Rel4
Source:	⌘	CN5
Work item code:	⌘	OSA1
Date:	⌘	07/06/2001
Category:	⌘	F
Release:	⌘	Rel4

Use one of the following categories:
- ***F*** *(essential correction)*
- ***A*** *(corresponds to a correction in an earlier release)*
- ***B*** *(Addition of feature),*
- ***C*** *(Functional modification of feature)*
- ***D*** *(Editorial modification)*

Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
REL-4	*(Release 4)*
REL-5	*(Release 5)*

Reason for change:	⌘	- Exception handling mechanism in 29.198 requires correction to enable it to be correctly used, without ambiguity - Security Mechanism in Framework needed to be corrected to ensure contact with non-authenticated entities is ended on authentication failure - Correction to Notification of added/withdrawn services and their service properties
Summary of change:	⌘	Replace TpGeneralException, TpUIException with detailed exception classes which can be thrown for each method; Remove accessCheck() method, add authenticationSucceeded() method; Add unannounceService() method and correct types associated with service properties.
Consequences if not approved:	⌘	29.198-3 will be ambiguous and difficult to implement correctly - inter-working might be jeopardised

Clauses affected:	⌘			
Other specs affected:	⌘	☒ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	All other parts of 29.198 except part 1 have similar changes
Other comments:	⌘			

3GPP TS 29.198-3 V4.0.~~0~~ 1 (2001-~~03~~06)

Technical Specification

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

Keywords
UMTS, 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.

Introduction

The present document is part 3 of a multi-part TS covering the 3rd Generation Partnership Project: Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API), as identified below. The **API specification** (3GPP TS 29.198) is structured in the following Parts:

Part 1:	Overview	
Part 2:	Common Data Definitions	
Part 3:	**Framework**	
Part 4:	Call Control SCF	
Part 5:	User Interaction SCF	
Part 6:	Mobility SCF	
Part 7:	Terminal Capabilities SCF	
Part 8:	Data Session Control SCF	
Part 9:	Generic Messaging SCF	(not part of 3GPP Release 4)
Part 10:	Connectivity Manager SCF	(not part of 3GPP Release 4)
Part 11:	Account Management SCF	
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.

OSA API specifications 29.198-family		OSA API Mapping - 29.998-family	
29.198-1	**Part 1: Overview**	**29.998-1**	**Part 1: Overview**
29.198-2	**Part 2: Common Data Definitions**	29.998-2	Not Applicable
29.198-3	**Part 3: Framework**	29.998-3	Not Applicable
29.198-4	**Part 4: Call Control SCF**	**29.998-4-1**	**Subpart 1: Generic Call Control – CAP mapping**
		29.998-4-2	
29.198-5	**Part 5: User Interaction SCF**	**29.998-5-1**	**Subpart 1: User Interaction – CAP mapping**
		29.998-5-2	
		29.998-5-3	
		29.998-5-4	**Subpart 4: User Interaction – SMS mapping**
29.198-6	**Part 6: Mobility SCF**	**29.998-6**	**User Status and User Location – MAP mapping**
29.198-7	**Part 7: Terminal Capabilities SCF**	29.998-7	Not Applicable
29.198-8	**Part 8: Data Session Control SCF**	**29.998-8**	**Data Session Control – CAP mapping**
29.198-9	Part 9: Generic Messaging SCF	29.998-9	Not Applicable
29.198-10	Part 10: Connectivity Manager SCF	29.998-10	Not Applicable
29.198-11	**Part 11: Account Management SCF**	29.998-11	Not Applicable
29.198-12	**Part 12: Charging SCF**	29.998-12	Not Applicable

1 Scope

The present document is Part 3 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 APIs. The concepts and the functional architecture for the OSA are contained in 3GPP TS 23.127 [3]. The requirements for OSA are contained in 3GPP TS 22.127 [2].

The present document specifies the Framework aspects of the interface. All aspects of the Framework are defined in the present document, 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, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[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 terms and definitions given in TS 29.198-1 [1] apply.

3.2 Abbreviations

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

4 Overview of the Framework

This clause 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 figure below). The description of the Framework in the present document separates the interfaces into two distinct sets: Framework to Application interfaces and Framework to Service interfaces.

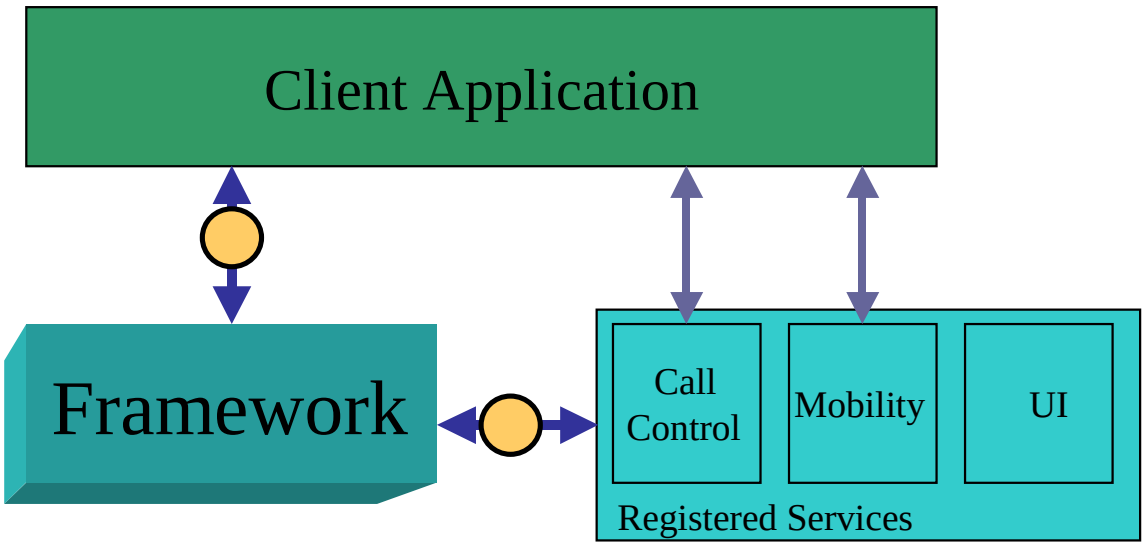


Figure:

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 shall authenticate the Framework and vice versa. The application shall 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 shall precede authorisation. Once authenticated, an application is authorised to access certain SCFs.

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

- **Establishment of service agreement:** Before any application can interact with a network SCF, a service agreement shall 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 SCF.

- **Access to network SCFs:** The Framework shall provide access control functions to authorise the access to SCFs 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 (SCS):

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

The following clauses 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* clause shows how each of the interfaces applicable to the Framework relate to one another.
- The *interface specification* clause describes in detail each of the interfaces shown within the class diagram part.
- The *State Transition Diagrams (STD)* show the transition between states in the Framework. The states and transitions are well-defined; either methods specified in the Interface specification or events occurring in the underlying networks cause state transitions.~~progression of internal processes, either in the application or in the gateway.~~
- The *data definitions* clause 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 the present document (29.198-2).

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.

<<Interface>> IpInterface

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.

<<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

TpCommonExceptions

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.

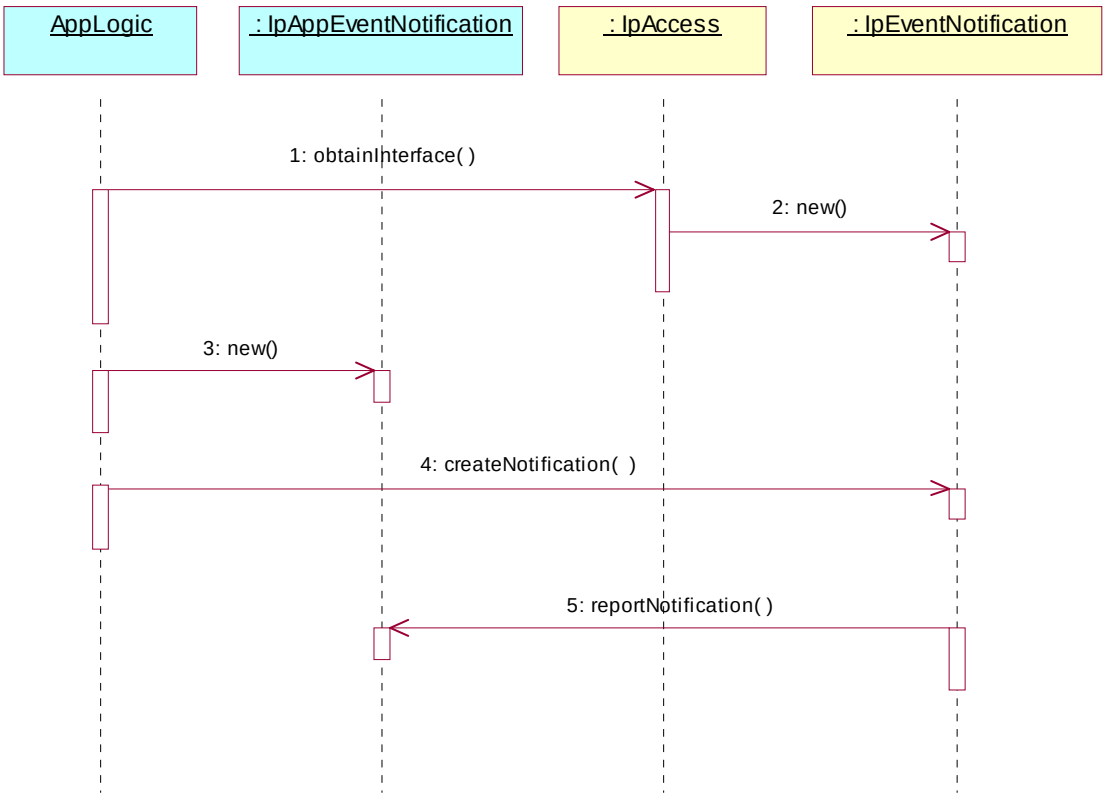
Raises

TpCommonExceptions

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: createNotification(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.

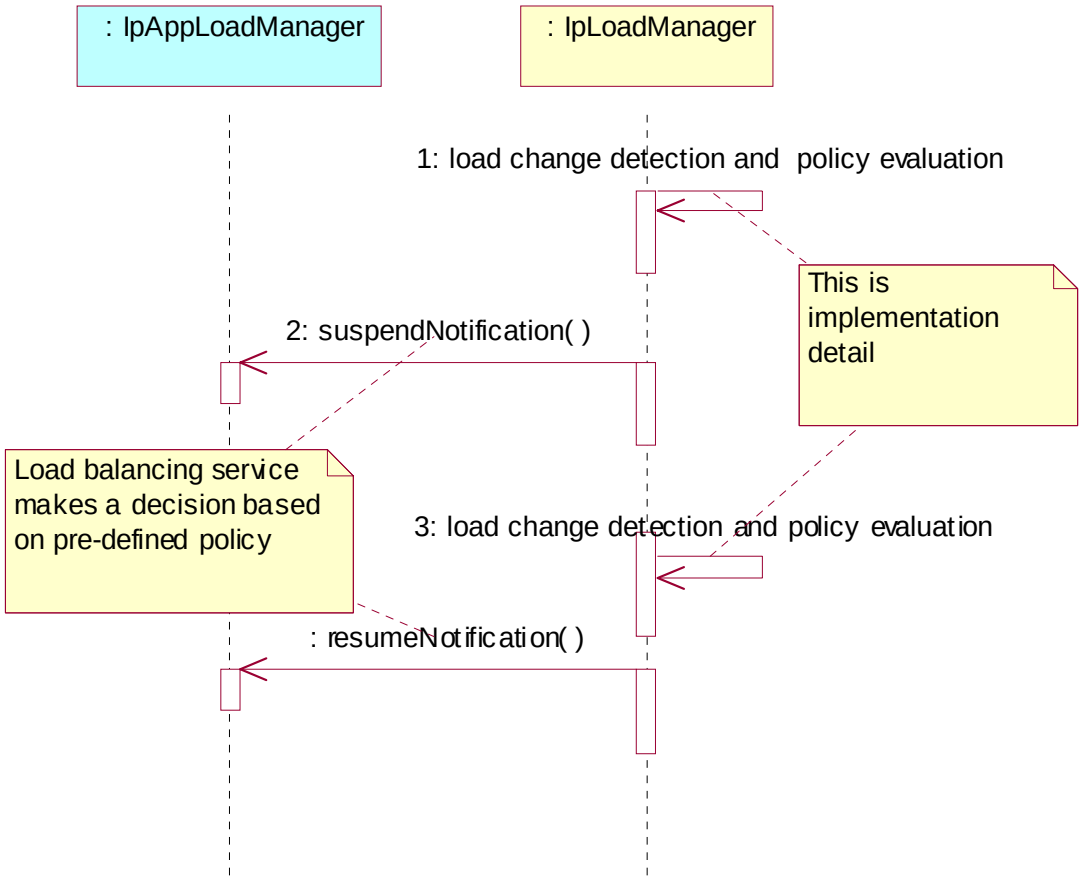
Alternatively, or additionally, the application can request to be informed of SCFs becoming unavailable.

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

6.2 Integrity Management Sequence Diagrams

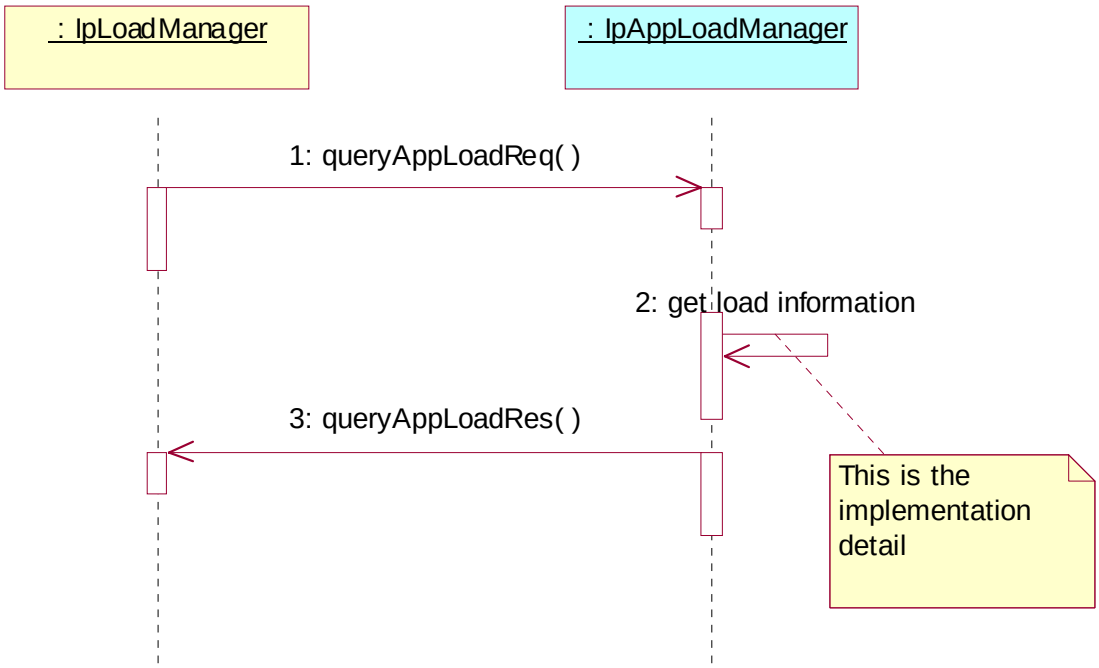
6.2.1 Load Management: Suspend/resume notification from application

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.



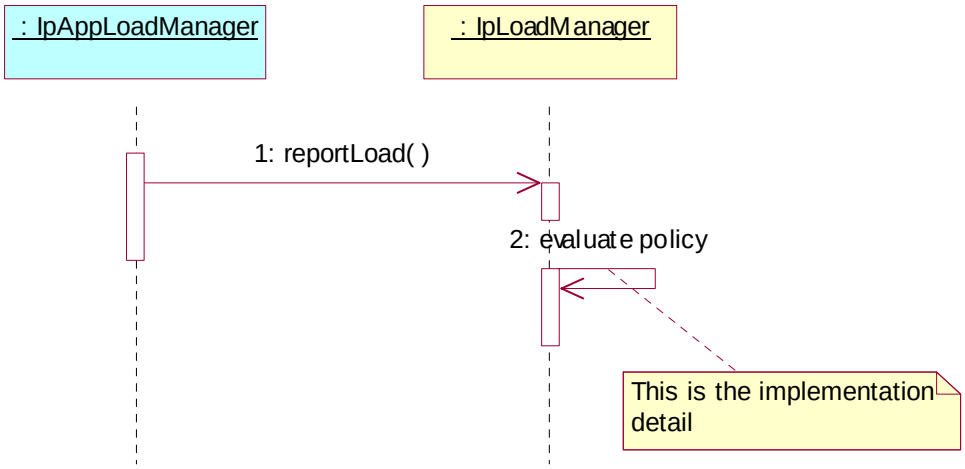
6.2.2 Load Management: Framework queries load statistics

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



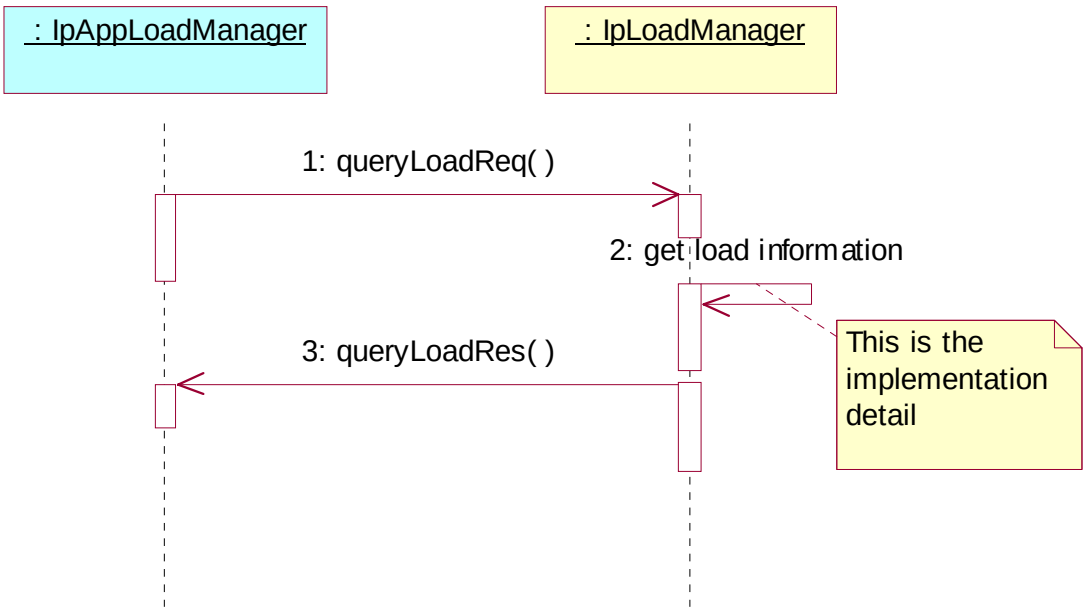
6.2.3 Load Management: Application reports current load condition

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



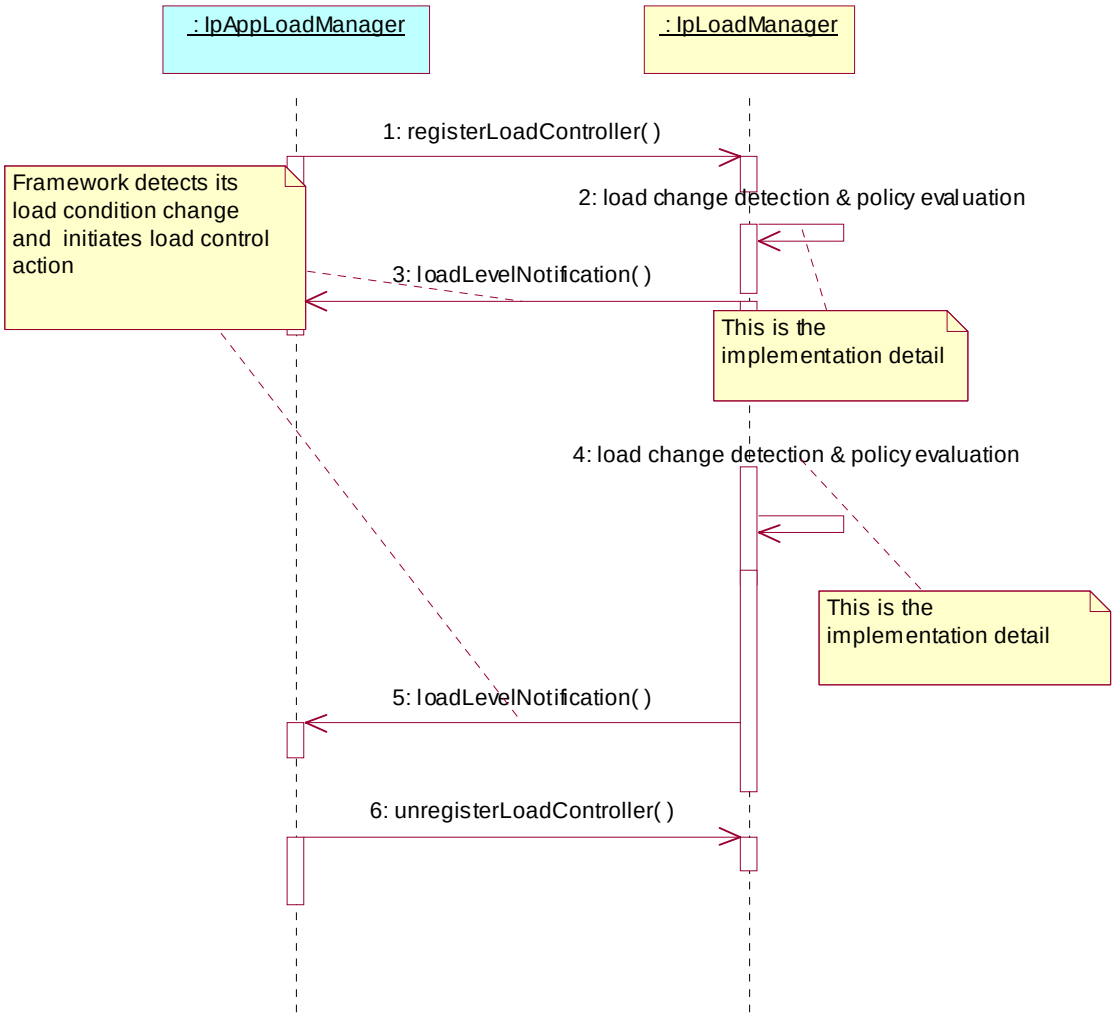
6.2.4 Load Management: Application queries load statistics

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

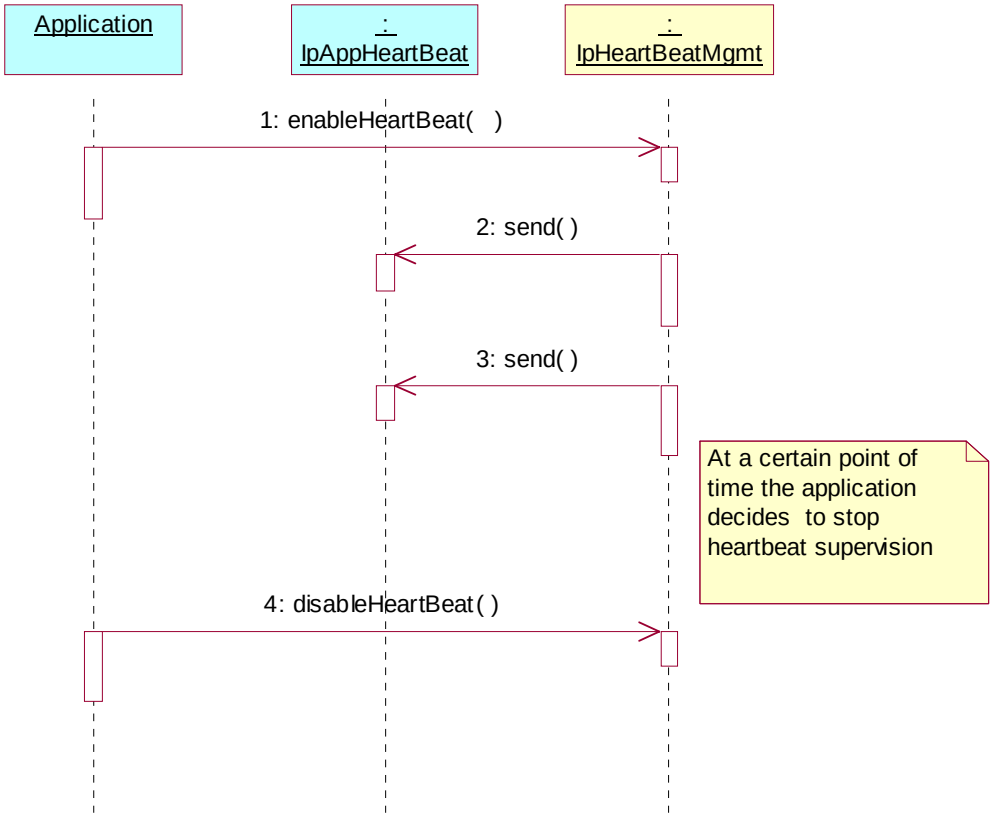


6.2.5 Load Management: Application callback registration and load control

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

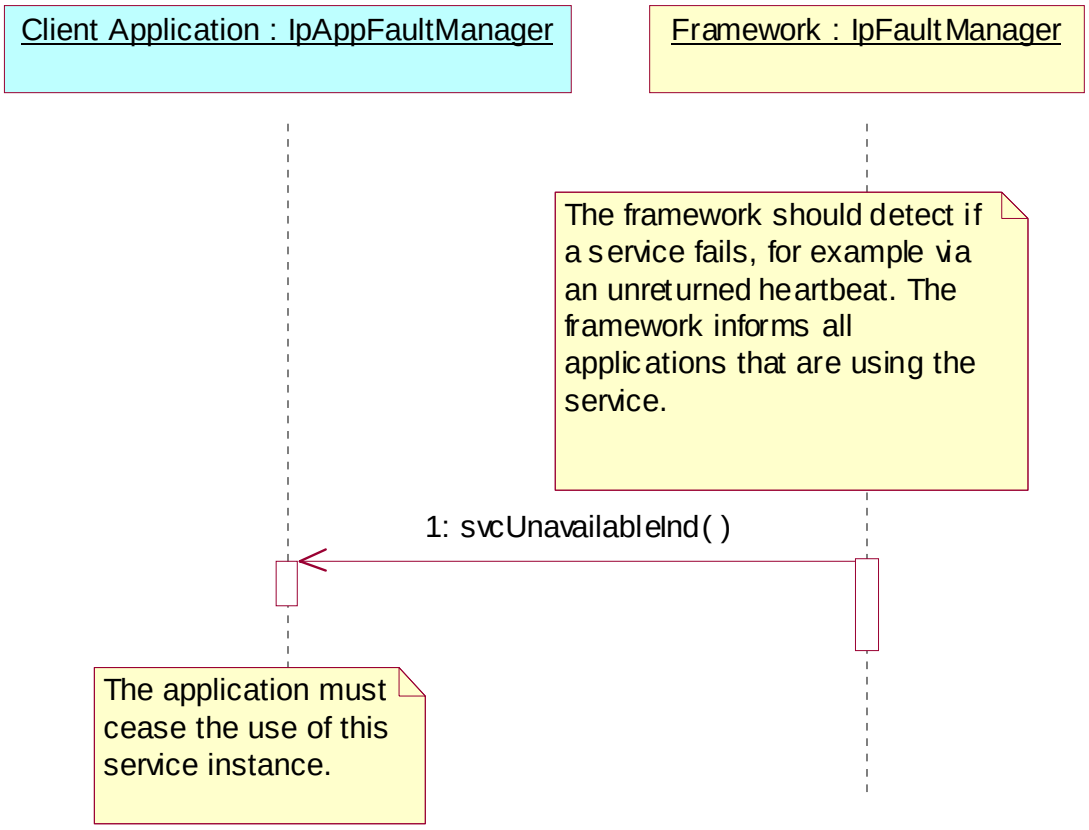


6.2.6 Heartbeat Management: Start/perform/end heartbeat supervision of application



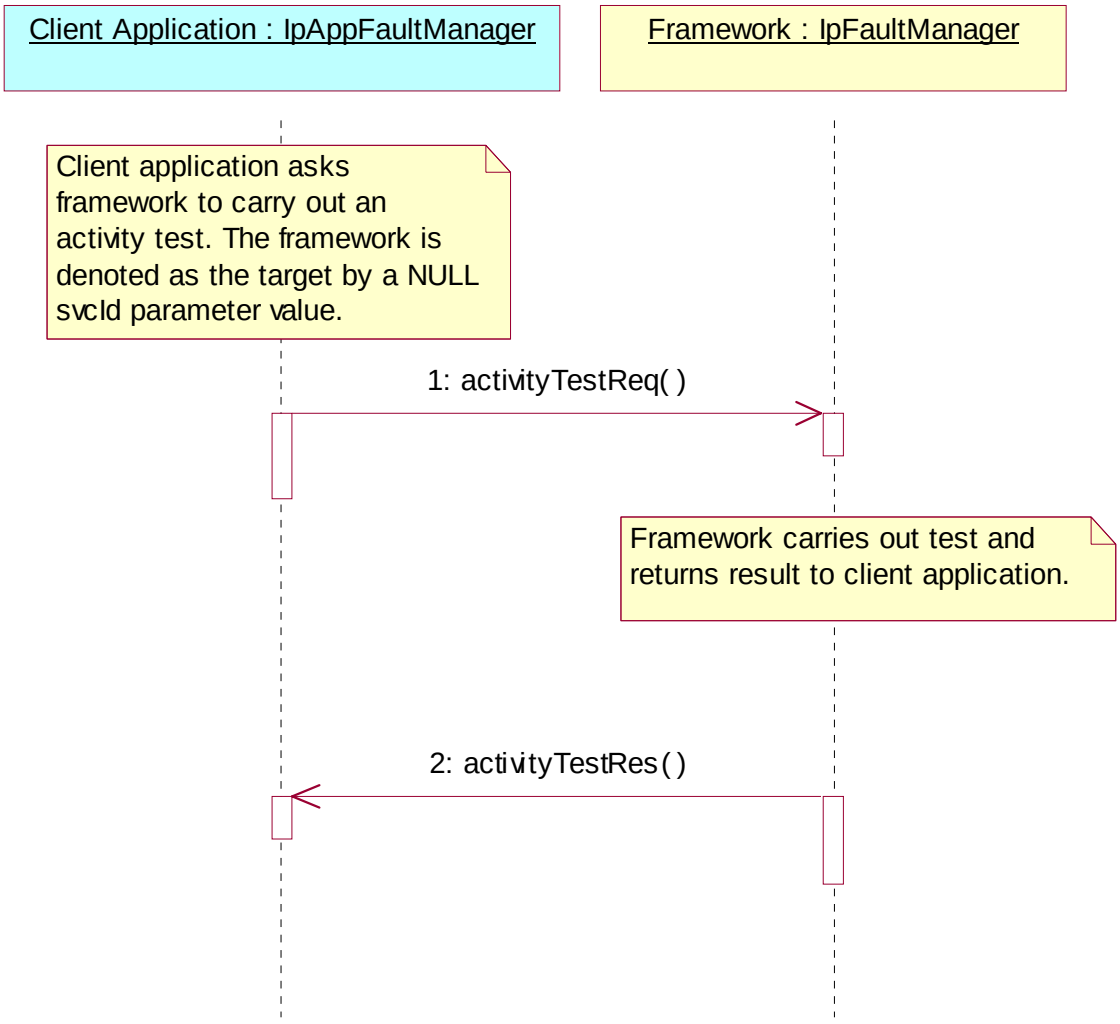
6.2.7 Fault Management: Framework detects a Service failure

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.

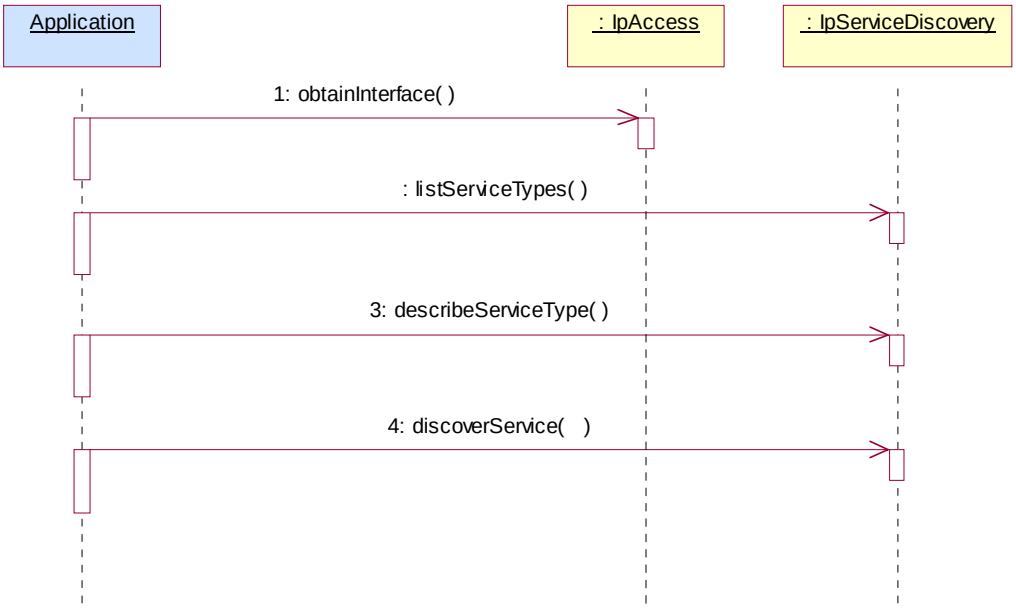
6.3 Service Discovery Sequence Diagrams

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 can be a three-step process. The first two steps have to be performed initially, but can subsequently be skipped (if the service type and its properties are already known, the application can invoke discoverService() without having to re-invoke the list/discoverServiceType methods):



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.

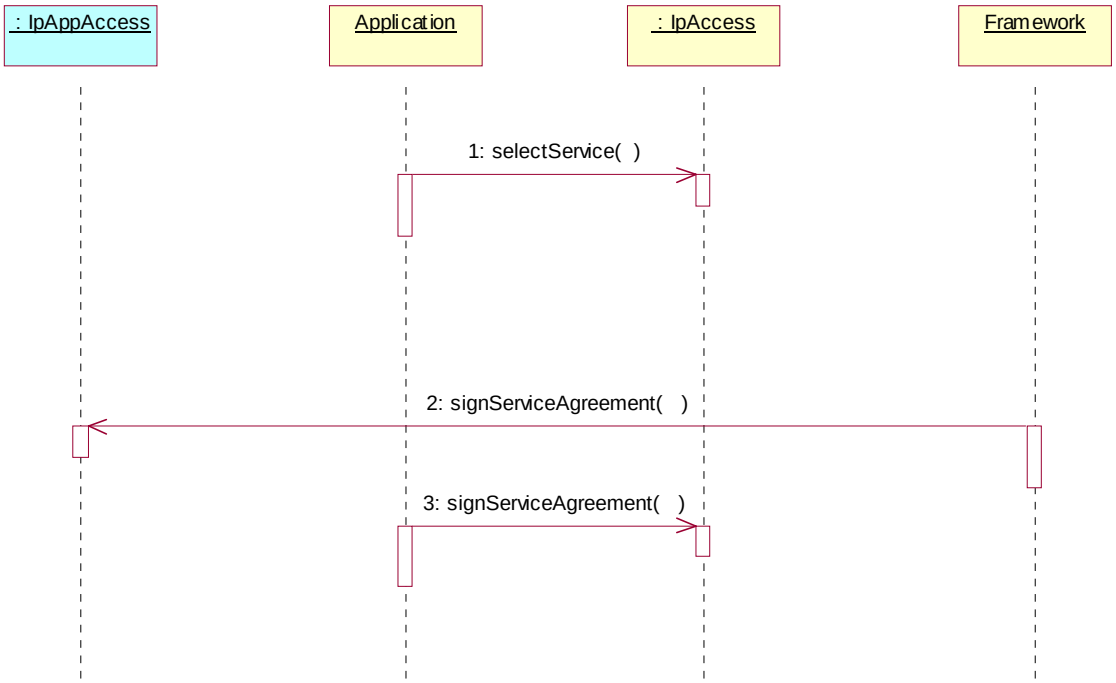
6.4 Trust and Security Management Sequence Diagrams

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.

2: 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 createServiceManager 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

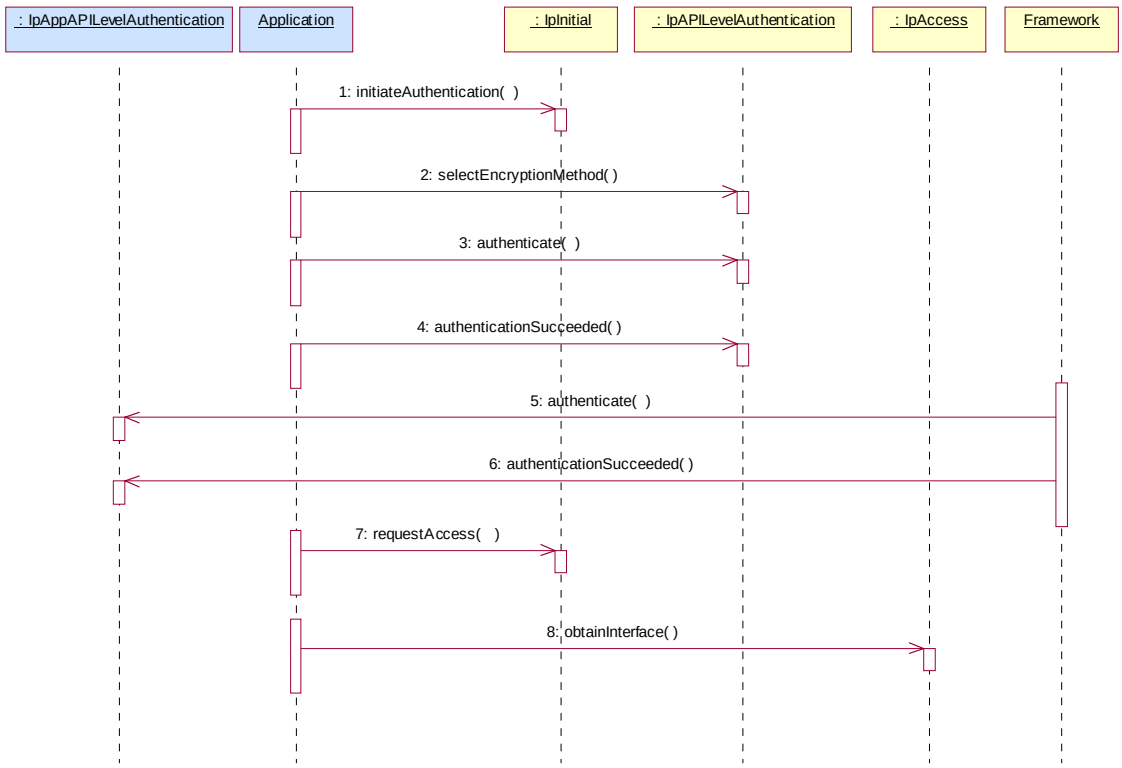
This is a reference to a structure containing the digital signature of the Framework for the service agreement, and a reference to the manager interface of the SCF.

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 selectEncryptionMethod 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 provides an indication if authentication succeeded.

5: 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.

6: The Framework provides an indication if authentication succeeded.

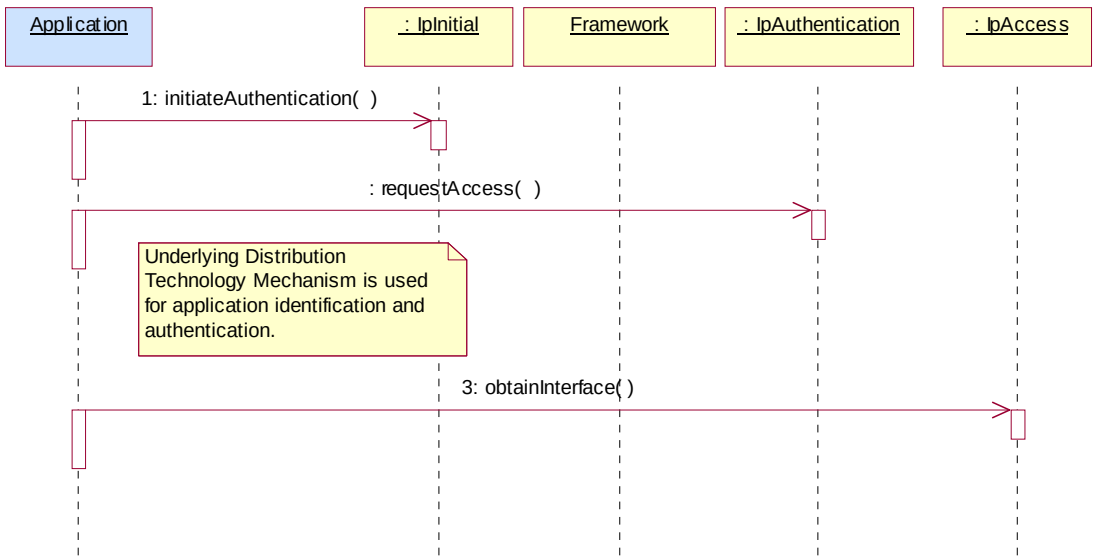
7: Request Access

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.

8: 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.

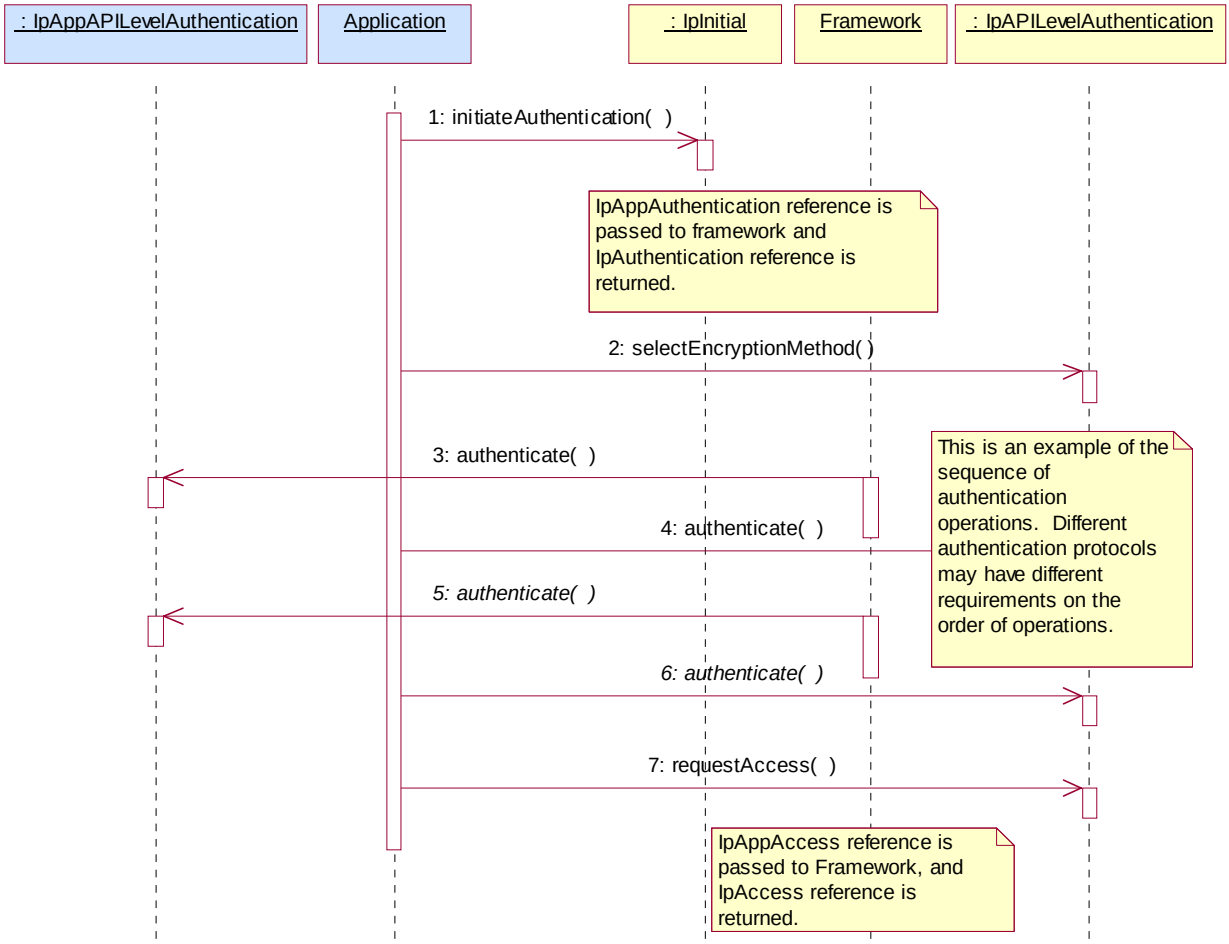
The OSA API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity. 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.



7 Framework-to-Application Class Diagrams

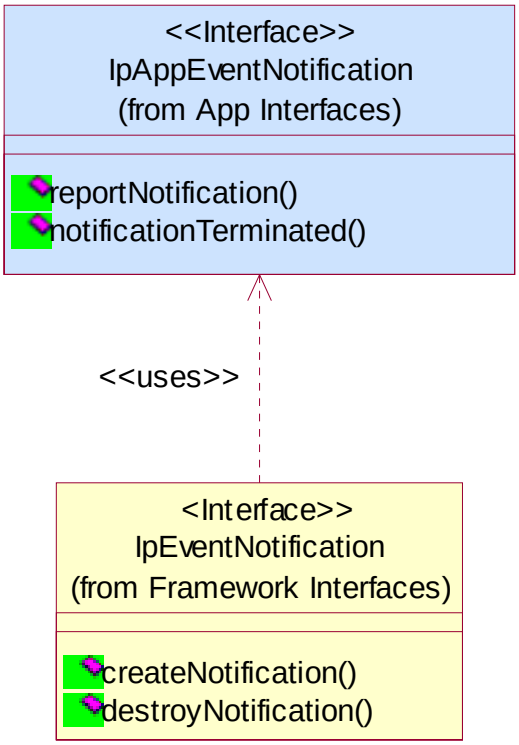


Figure: Event Notification Class Diagram

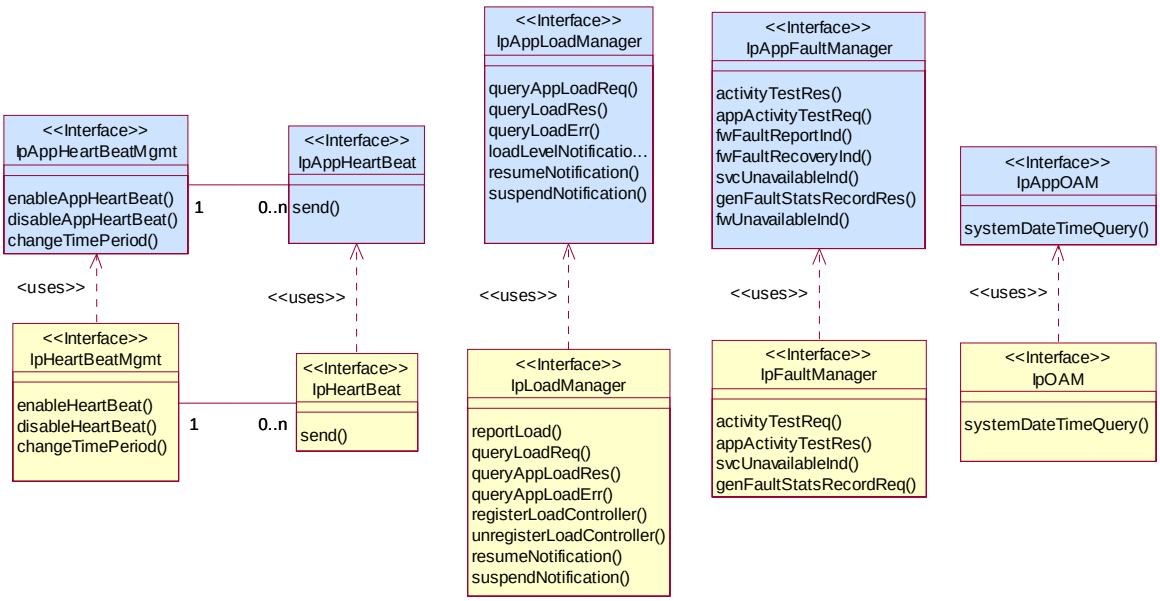


Figure: Integrity Management Package Overview

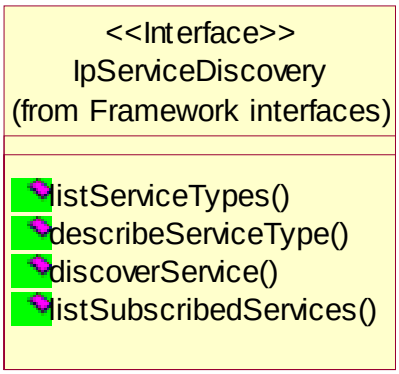


Figure: Service Discovery Package Overview

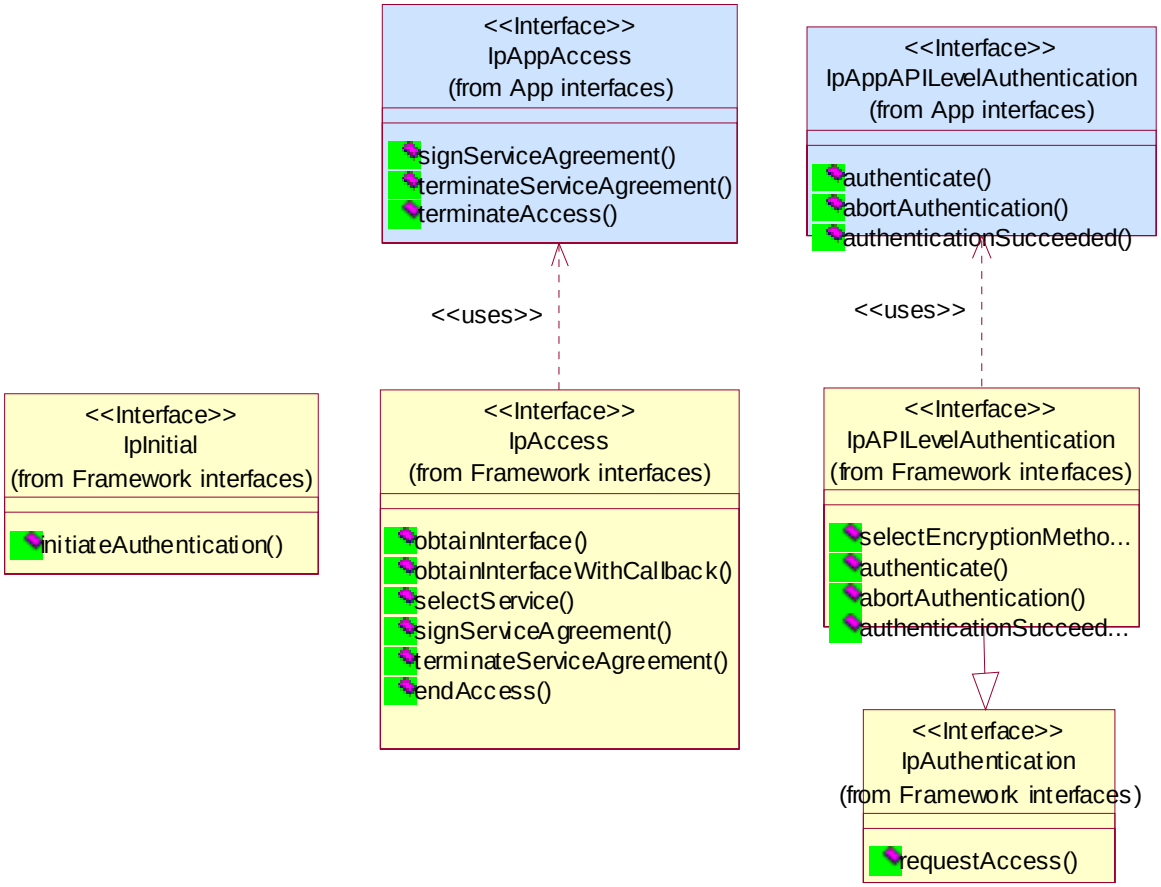


Figure: Trust and Security Management Package Overview

8 Framework-to-Application Interface Classes

8.1 Trust and Security Management Interface Classes

The Trust and Security Management Interfaces provide:

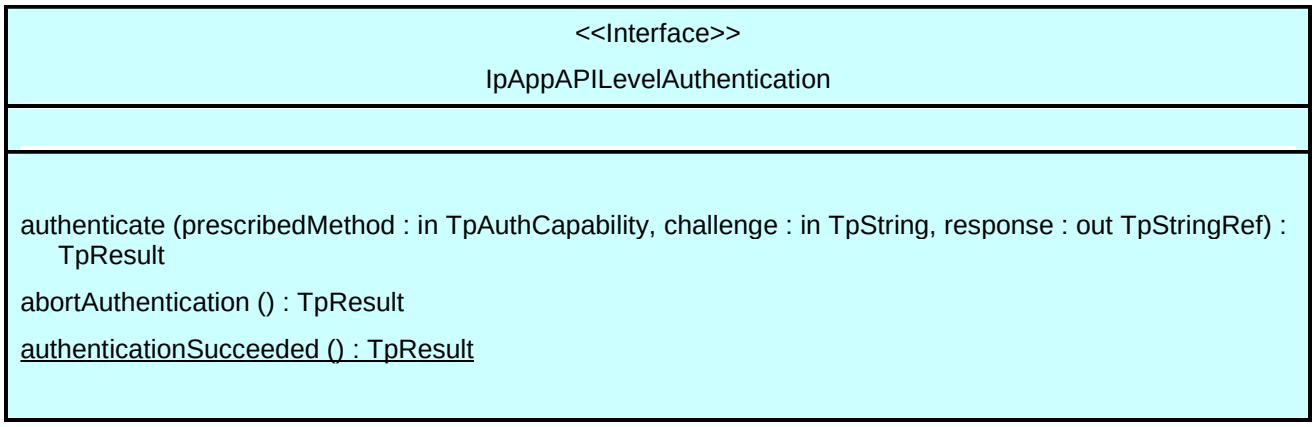
- the first point of contact for an application to access a Home Environment;
- the authentication methods for the application and Home Environment to perform an authentication protocol;
- the application with the ability to select a service capability feature to make use of;
- the application with a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

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, (unless the application responded incorrectly to a challenge in which case no further communication with the application should occur.) 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

Method

authenticationSucceeded()

The Framework uses this method to inform the client application of the success of the authentication attempt.

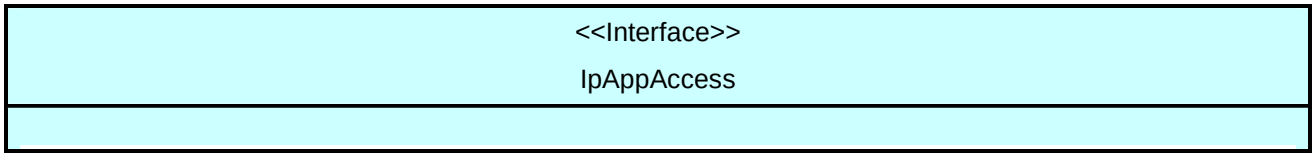
Parameters

No Parameters were identified for this method

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.



signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : out TpStringRef) : TpResult

terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpString) : TpResult

terminateAccess (terminationText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : in TpString) : TpResult

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. The service manager returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. 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.

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.

Method

terminateAccess()

The terminateAccess operation is used by the framework to end the client application's access session.

After terminateAccess() 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. If at any point the framework's level of confidence in the identity of the client becomes too low, perhaps due to re-authentication failing, the framework should terminate all outstanding service agreements for that client application, and should take steps to terminate the client application's access session WITHOUT invoking terminateAccess() on the client application. This follows a generally accepted security model where the framework has decided that it can no longer trust the application and will therefore sever ALL contact with it.

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.

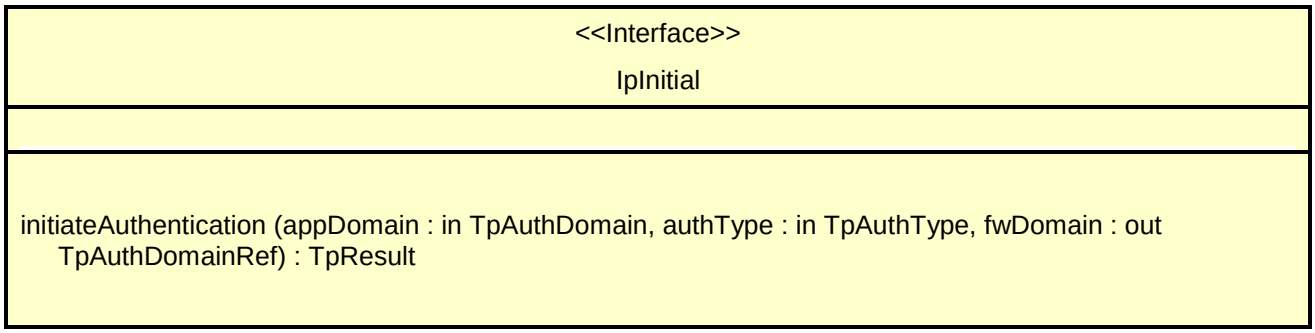
digitalSignature : in TpString

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.

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.



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.

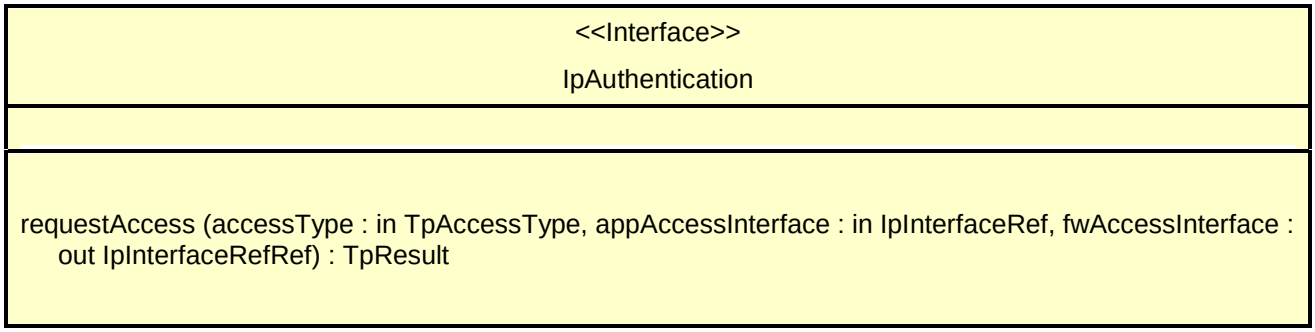
Raises

TpCommonExceptions,P_INVALID_DOMAIN_ID,P_INVALID_INTERFACE_TYPE,P_INVALID_AUTH_TYPE

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.

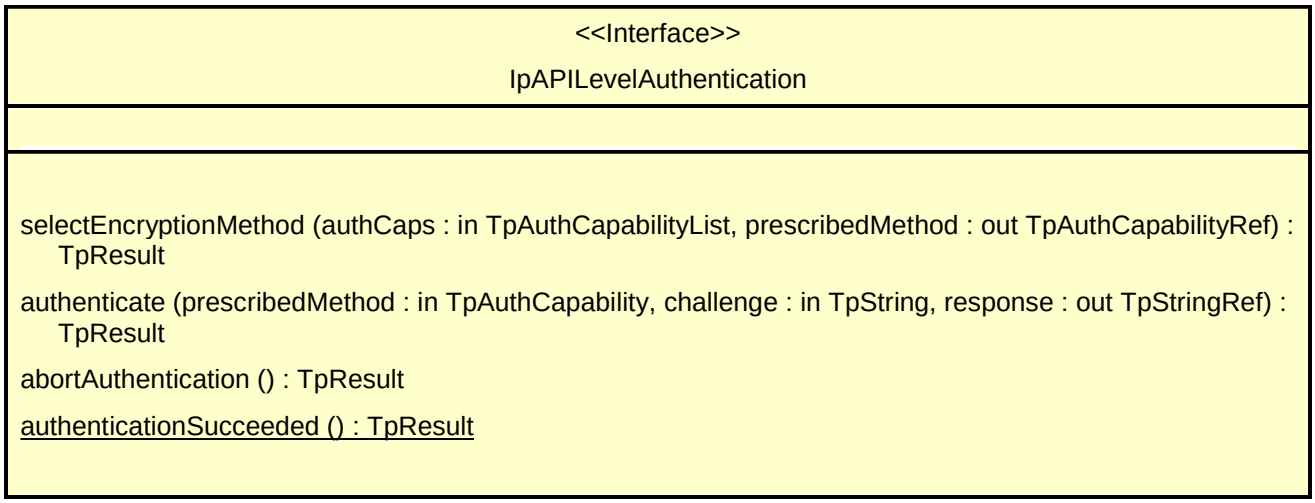
Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_ACCESS_TYPE,P_INVALID_INTERFACE_TYPE

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

TpCommonExceptions,P_ACCESS_DENIED,P_NO_ACCEPTABLE_AUTH_CAPABILITY

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

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AUTH_CAPABILITY

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, (unless the application responded incorrectly to a challenge in which case no further communication with the application should occur.) 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

TpCommonExceptions,P_ACCESS_DENIED

Method

authenticationSucceeded()

The client application uses this method to inform the framework of the success of the authentication attempt.

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions,P_ACCESS_DENIED

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

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_INTERFACE_NAME

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

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_INTERFACE_NAME,P_INVALID_INTERFACE_TYPE

~~*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

TpCommonExceptions,P_ACCESS_DENIED, P_INVALID_SERVICE_ID

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. The service manager

returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. 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

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AGREEMENT_TEXT,P_INVALID_SERVICE_TOKEN,P_INVALID_SIGNING_ALGORITHM,P_SERVICE_ACCESS_DENIED

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

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_SERVICE_TOKEN,P_INVALID_SIGNATURE

Method

endAccess()

The endAccess operation is used by the client to request that its access session with the framework 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

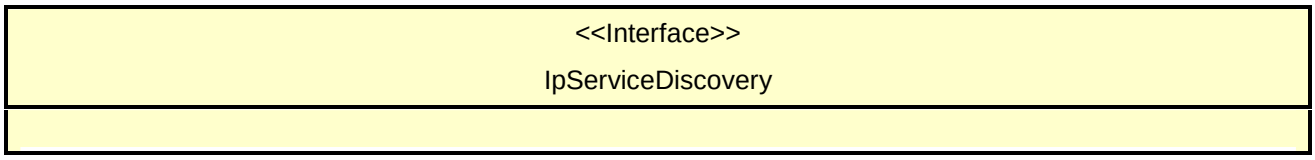
TpCommonExceptions,P_ACCESS_DENIED, P_INVALID_PROPERTY

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.



listServiceTypes (listTypes : out TpServiceTypeNameListRef) : TpResult

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

TpCommonExceptions,P_ACCESS_DENIED

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

TpCommonExceptions,P_ACCESS_DENIED,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE

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. The service properties returned will form a complete view of what the client application will be able to do with the service, as per the service level agreement. If the framework supports service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties.

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

TpCommonExceptions,P_ACCESS_DENIED,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE,P_INVALID_PROPERTY

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

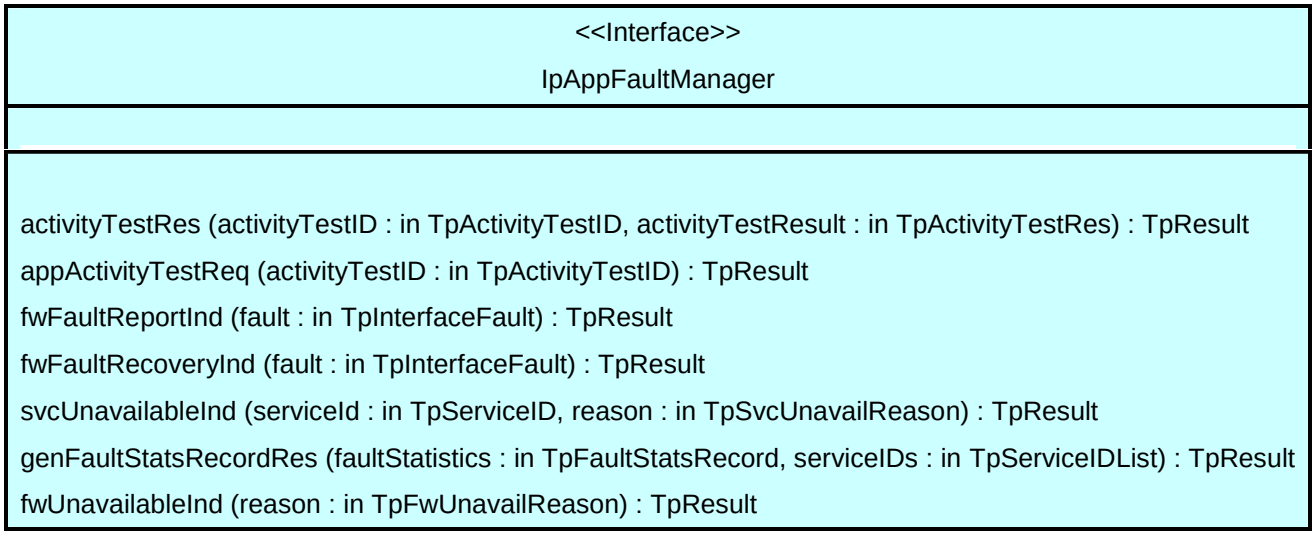
TpCommonExceptions,P_ACCESS_DENIED

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.

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.

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.

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.

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

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.

Method

fwUnavailableInd()

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

Parameters

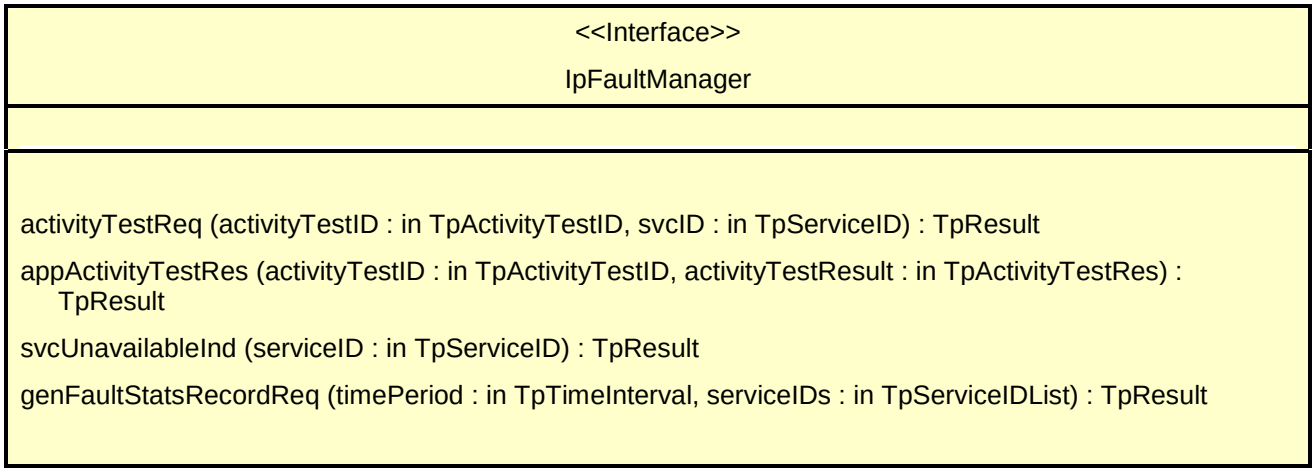
reason : in TpFwUnavailReason

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

TpCommonExceptions,P_INVALID_SERVICE_ID

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

TpCommonExceptions,P_INVALID_SERVICE_ID,P_INVALID_ACTIVITY_TEST_ID

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

TpCommonExceptions ,P_INVALID_SERVICE_ID

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.

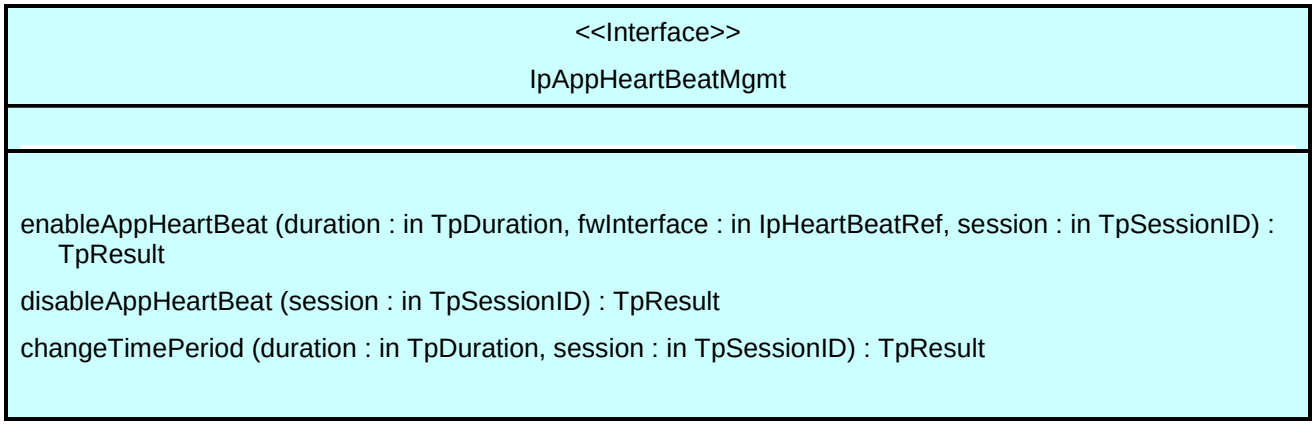
Raises

TpCommonExceptions ,P_INVALID_SERVICE_ID

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.

Method

disableAppHeartBeat()

Allows the stop of the heartbeat supervision of the application.

Parameters

session : in TpSessionID

Identifies the heartbeat session.

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.

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.

<<Interface>> IpAppHeartBeat
send (session : in TpSessionID) : TpResult

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.

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.

<<Interface>> IpHeartBeatMgmt
enableHeartBeat (duration : in TpDuration, appInterface : in IpAppHeartBeatRef, session : out TpSessionIDRef) : TpResult disableHeartBeat (session : in TpSessionID) : TpResult changeTimePeriod (duration : in TpDuration, session : in TpSessionID) : TpResult

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

TpCommonExceptions,P_INVALID_SESSION_ID

Method

disableHeartBeat()

Allows the stop of the heartbeat supervision of the application.

Parameters

session : in TpSessionID

Identifies the heartbeat session.

Raises

TpCommonExceptions,P_INVALID_SESSION_ID

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.

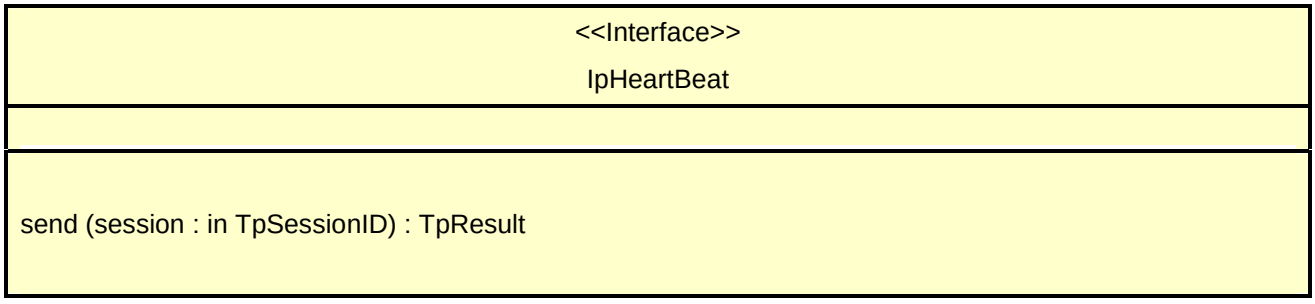
Raises

TpCommonExceptions,P_INVALID_SESSION_ID

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.

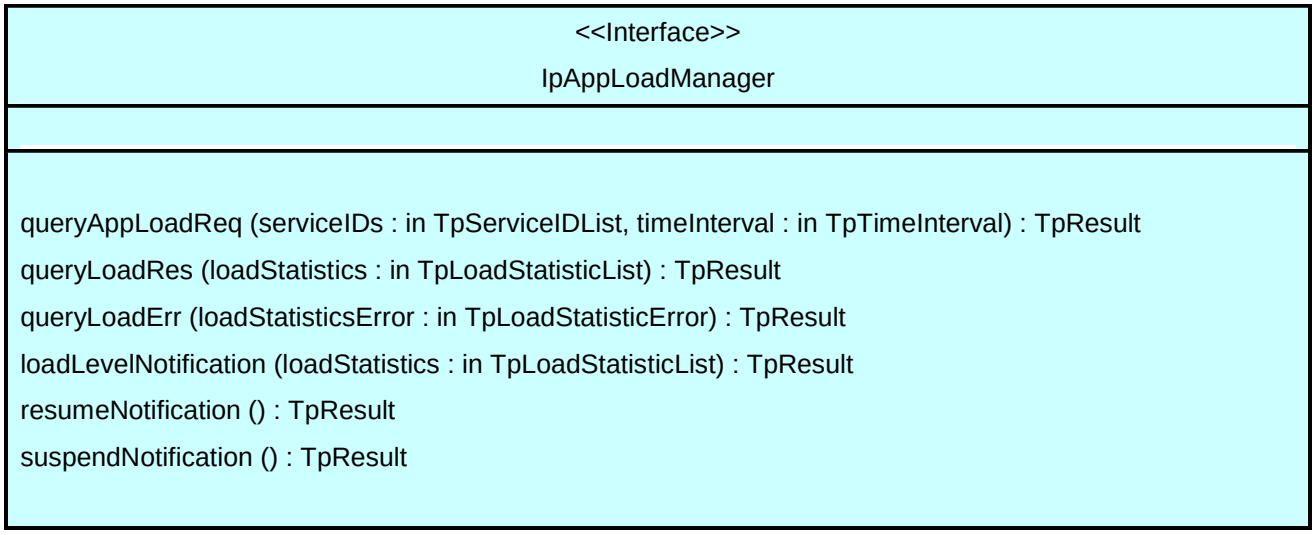
Raises

TpCommonExceptions,P_INVALID_SESSION_ID

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 statistics records for the application or for individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

Specifies the application or the services for which load statistic records should be reported. If this parameter is not an empty list, load statistics records of the specified services are returned, otherwise the load statistics record of the application is returned.

timeInterval : in TpTimeInterval

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

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

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.

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).

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

Method

suspendNotification()

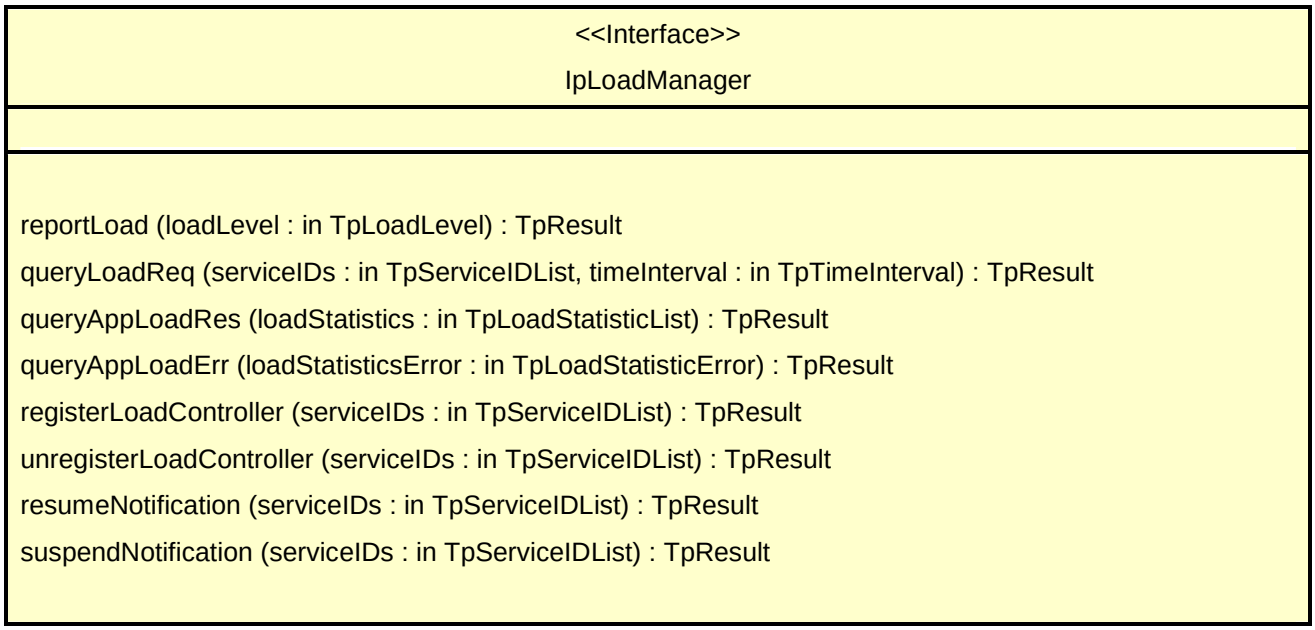
The framework uses this method to request the application to suspend sending it any notifications: e.g. while the framework handles a temporary overload condition.

Parameters
No Parameters were identified for this method

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.



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

TpCommonExceptions

Method

queryLoadReq()

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

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework or the services for which load statistics records should be reported. If this parameter is not an empty list, load statistics records of the specified services are returned, otherwise the load statistics record of the framework is returned.

timeInterval : in TpTimeInterval

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

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED

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

TpCommonExceptions

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

TpCommonExceptions

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

TpCommonExceptions, P_INVALID_SERVICE_ID

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

TpCommonExceptions,P_INVALID_SERVICE_ID

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

TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED

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

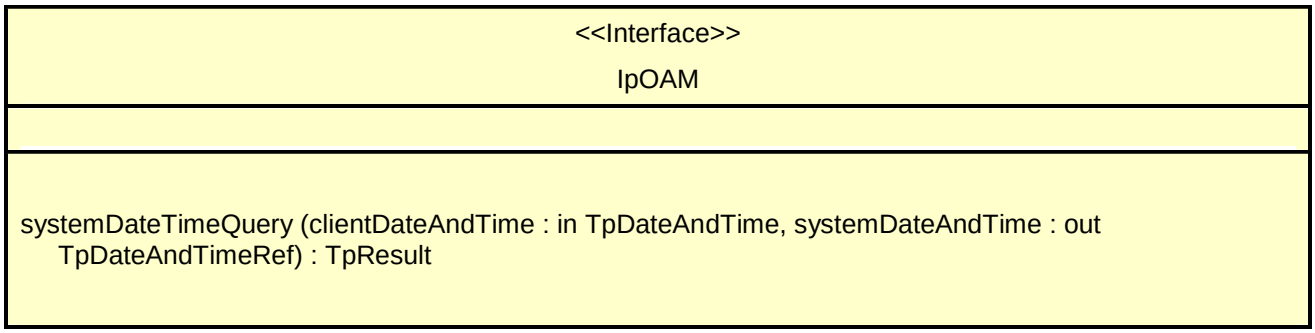
Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED

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.



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.

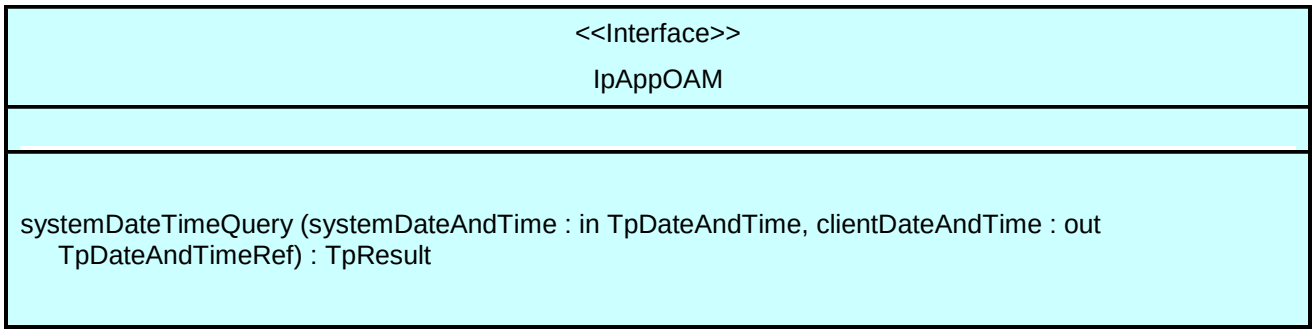
Raises

TpCommonExceptions,P_INVALID_TIME_AND_DATE_FORMAT

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.



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

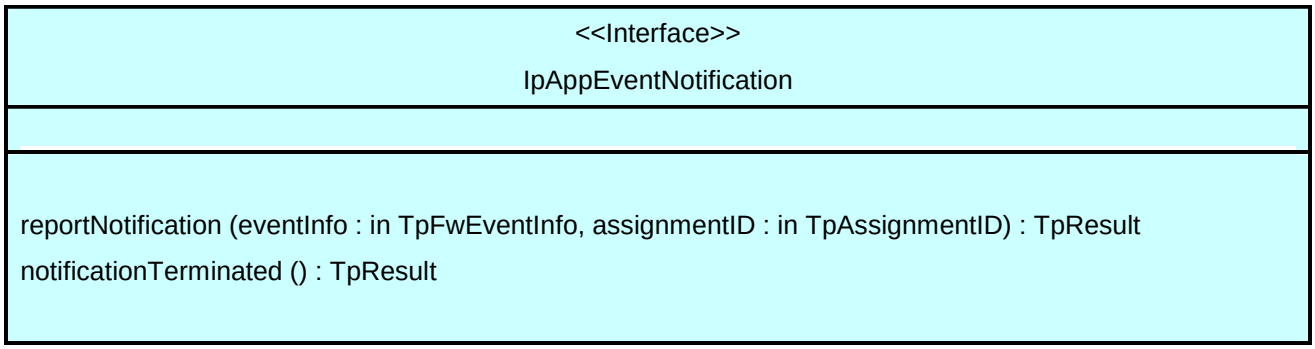
This is the date and time of the client (application). The error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

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.

Method
`notificationTerminated()`

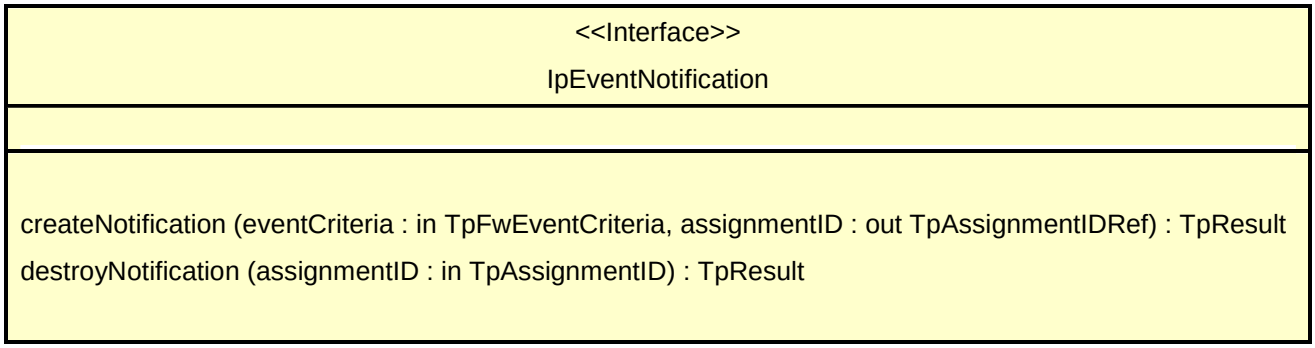
This method indicates to the application that all generic event notifications have been terminated (for example, due to faults detected).

Parameters
No Parameters were identified for this method

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

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE

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

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_ASSIGNMENT_ID

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

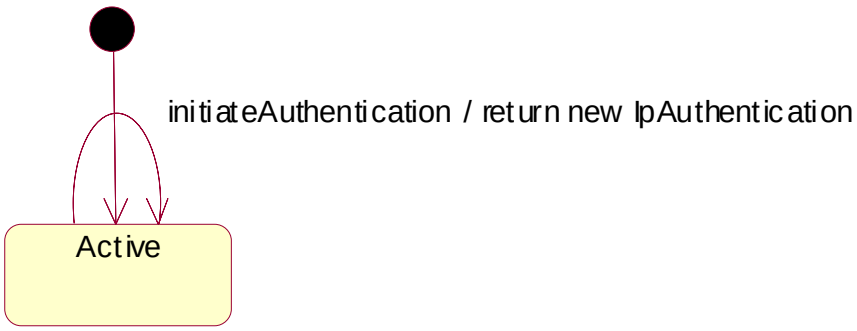


Figure : State Transition Diagram for IpInitial

9.1.1.1 Active State

9.1.2 State Transition Diagrams for IpAPILevelAuthentication

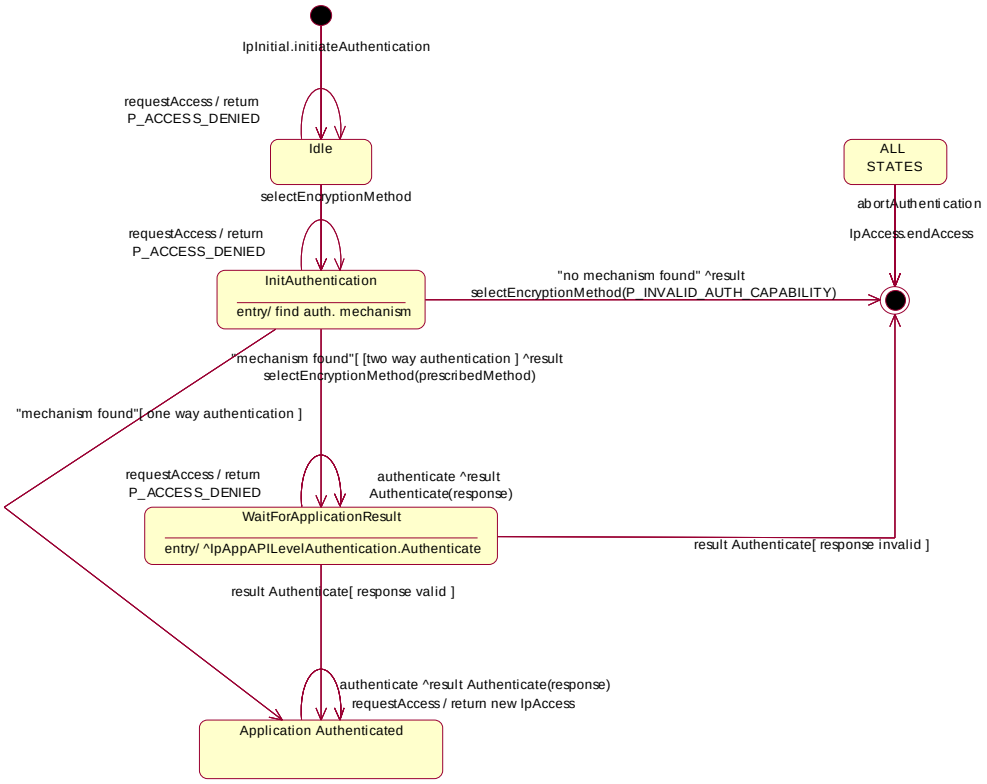


Figure : State Transition Diagram for IpAPILevelAuthentication

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

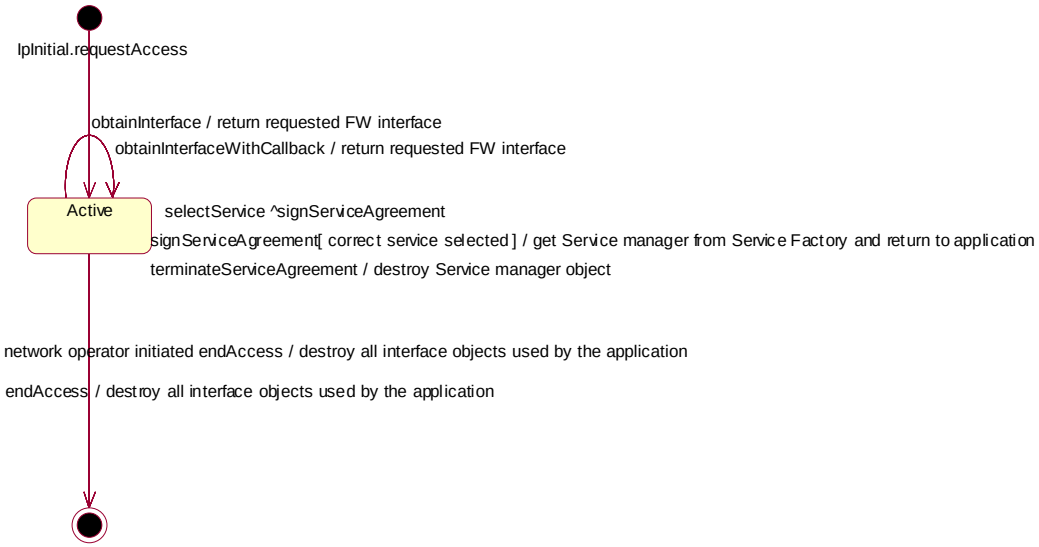


Figure : State Transition Diagram for IpAccess

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

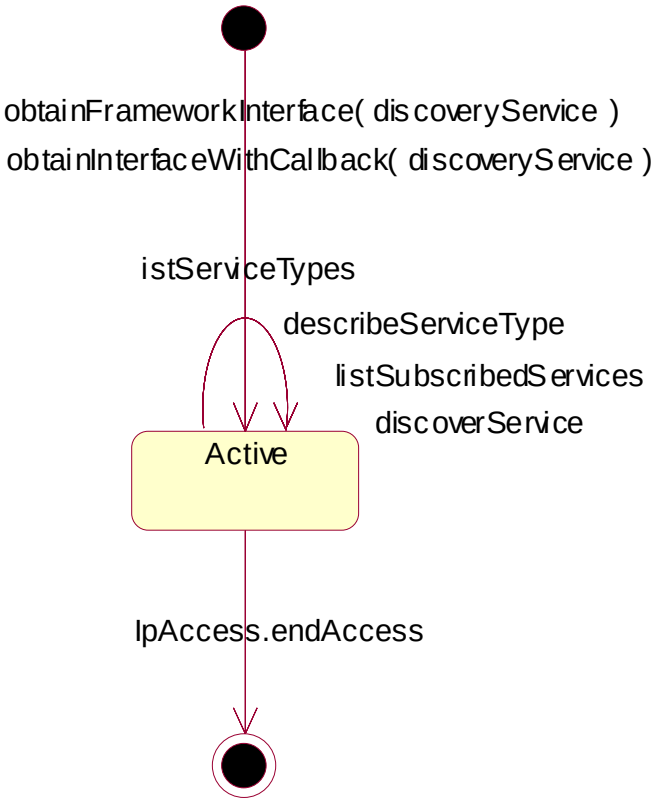


Figure : State Transition Diagram for IpServiceDiscovery

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

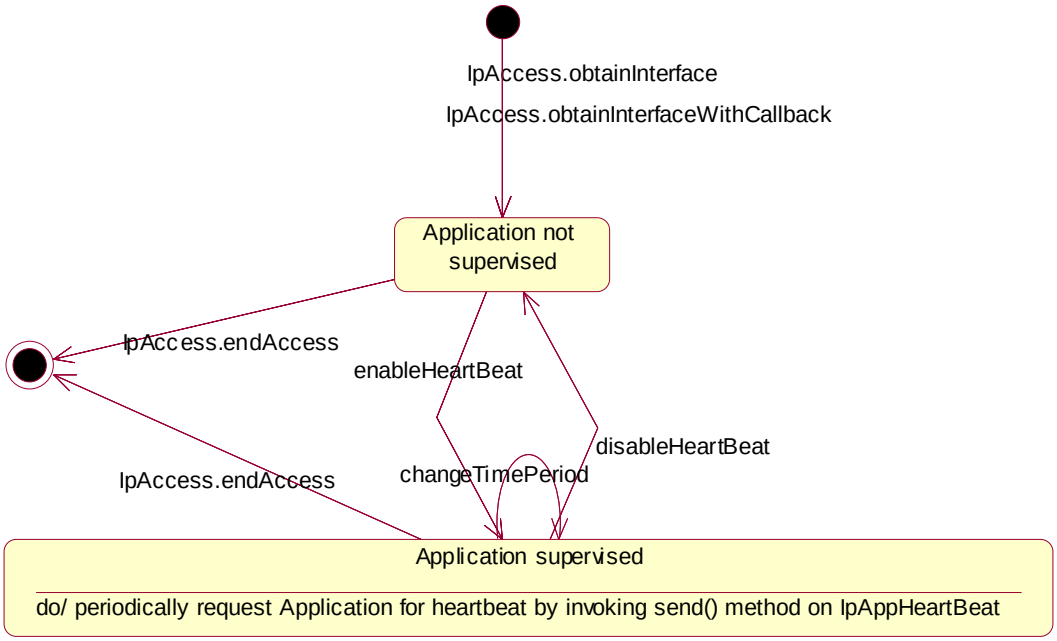


Figure : State Transition Diagram for IpHeartBeatMgmg

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

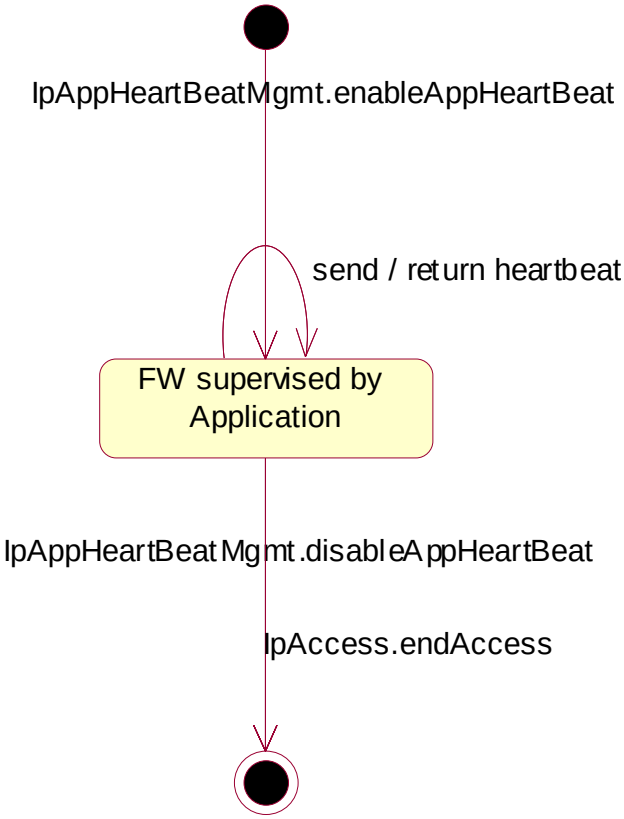


Figure : State Transition Diagram for IpHeatBeat

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

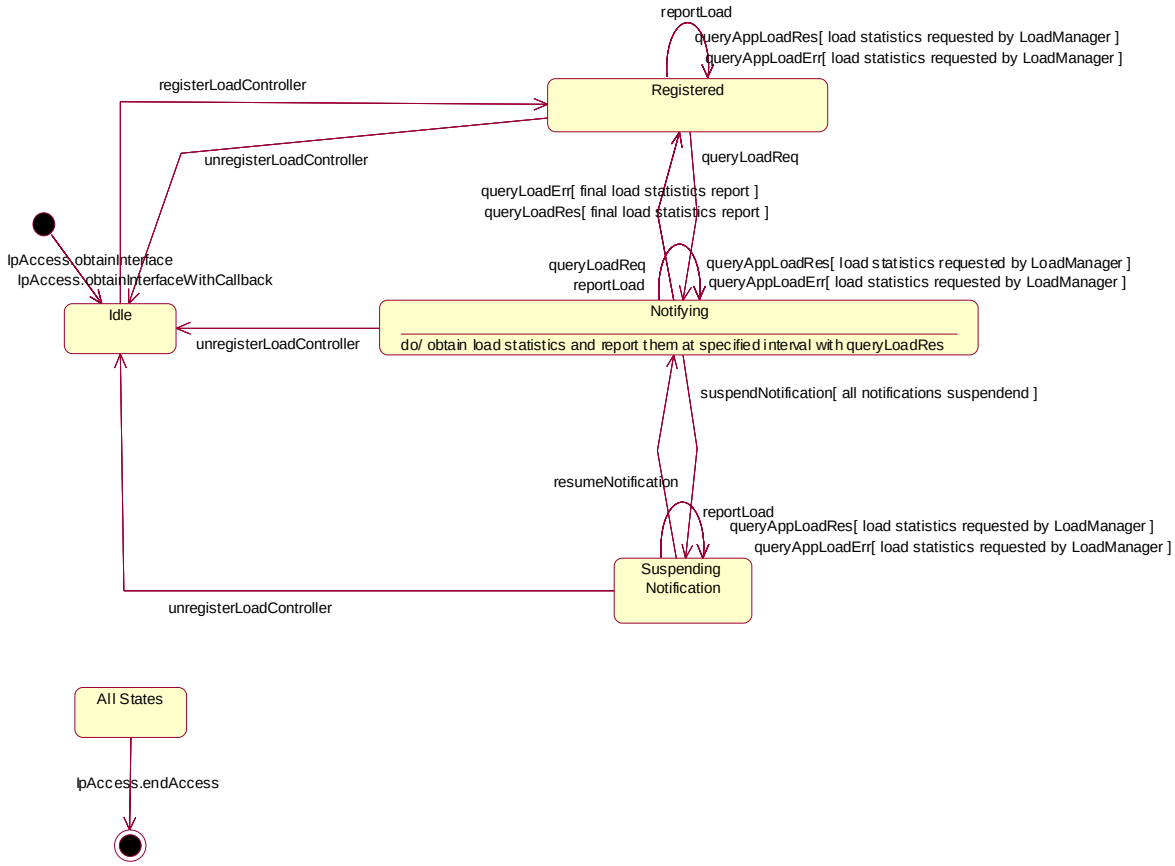


Figure : State Transition Diagram for IpLoadManager

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 LoadManagerInternal

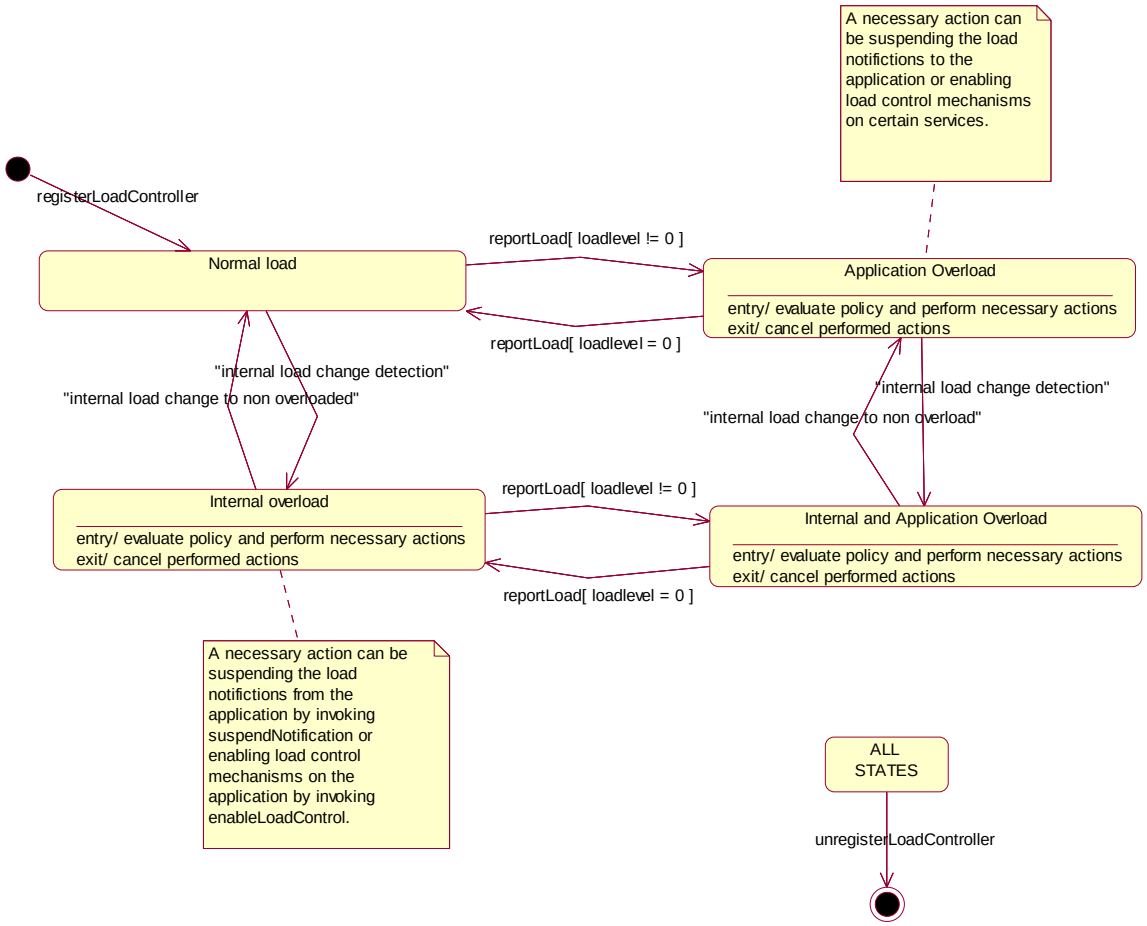


Figure : State Transition Diagram for LoadManagerInternal

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

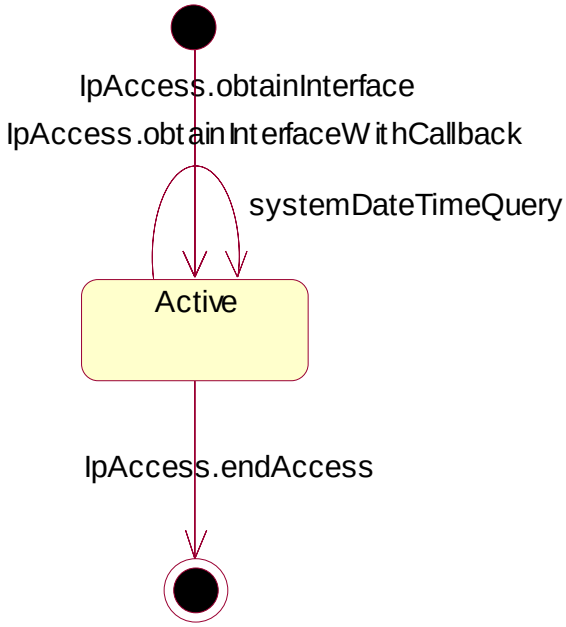


Figure : State Transition Diagram for IpOAM

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

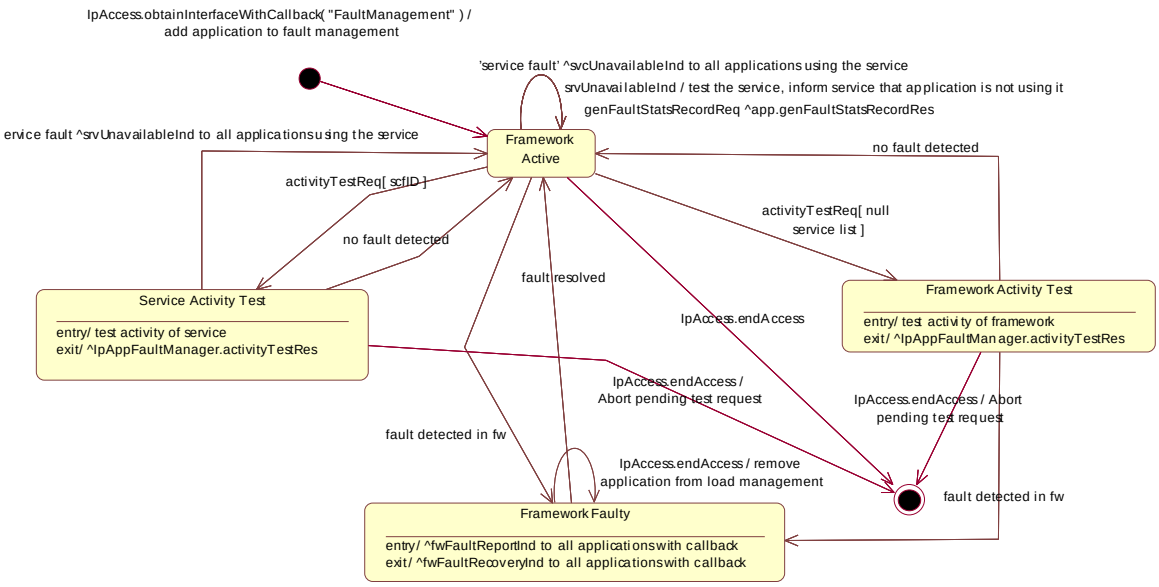


Figure : State Transition Diagram for IpFaultManager

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

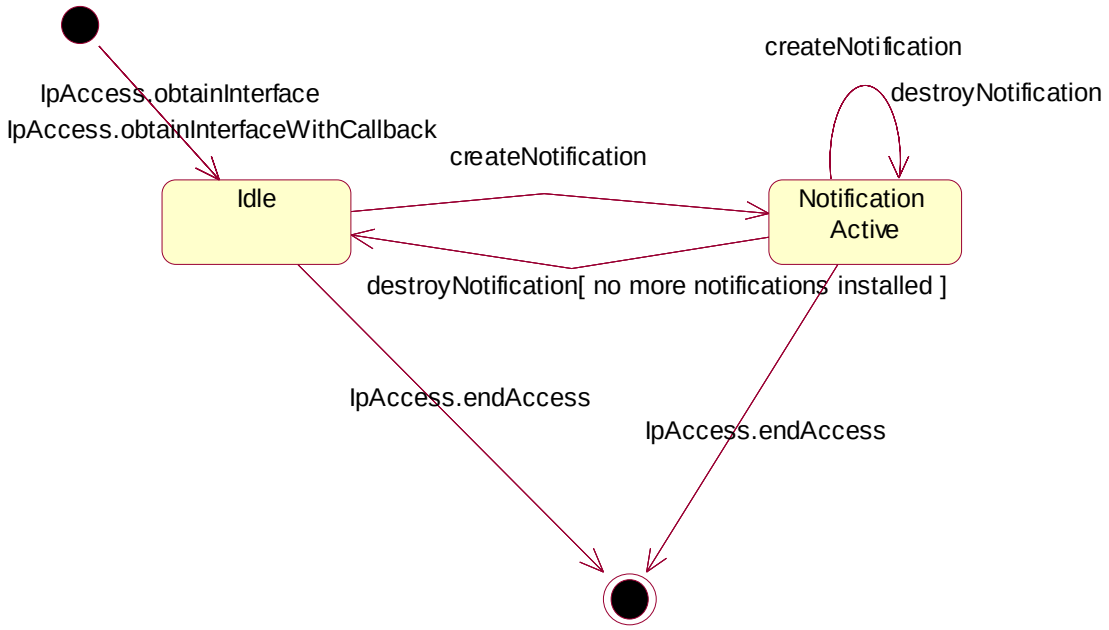


Figure : State Transition Diagram for IpEventNotification

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 order to make the SCF available an "entry point", called service factory, is used. The role of the service factory is to control the life cycle of an interface, or set of interfaces, and provide clients with the references that are necessary to invoke the methods offered by these interfaces. 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.

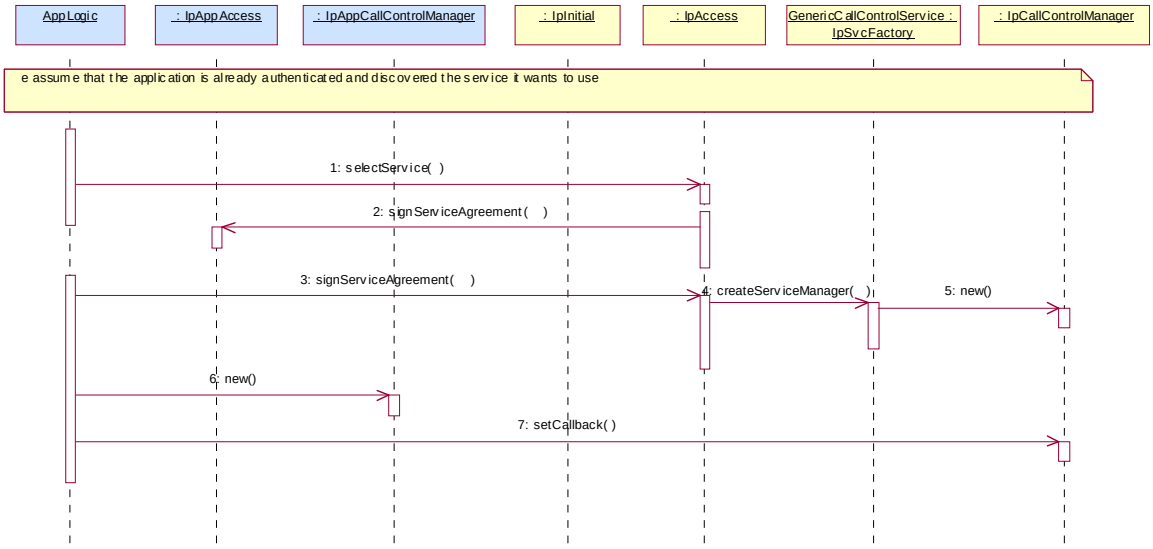
· in serviceFactoryRef

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 createServiceManager() 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

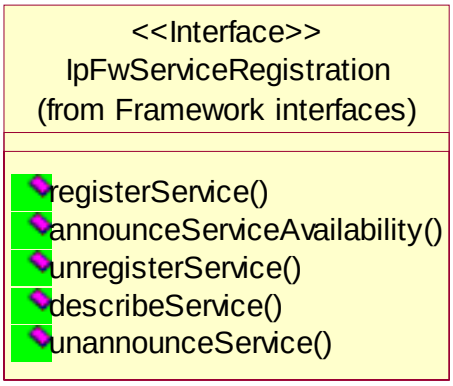


Figure: Service Registration Package Overview

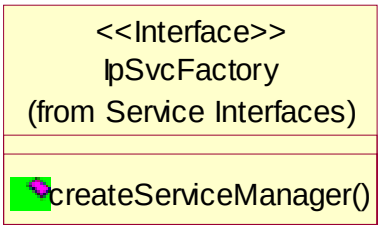


Figure: Service Factory Package Overview

12 Framework-to-Service Interface Classes

12.1 Service Registration Interface Classes

Before a service can be brokered (discovered, subscribed, accessed, etc.) by an enterprise, it has to be registered with the Framework. Services are registered against a particular service type. Therefore service types are created first, and then services corresponding to those types are accepted from the Service Suppliers for registration in the framework. The framework maintains a repository of service types and registered services.

In order to register a new service in the framework, the service supplier must select a service type and the "property values" for the service. The service discovery functionality described in the previous section enables the service

supplier to obtain a list of all the service types supported by the framework and their associated sets of service property values.

The Framework service registration-related interfaces are invoked by third party service supplier's administrative applications. They are described below. Note that these methods cannot be invoked until the authentication methods have been invoked successfully.

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 IpSvcFactoryRef) : TpResult unregisterService (serviceID : in TpServiceID) : TpResult describeService (serviceID : in TpServiceID, serviceDescription : out TpServiceDescriptionRef) : TpResult unannounceService (serviceID : in TpServiceID) : 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

TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID,P_PROPERTY_TYPE_MISMATCH,P_DUPLICATE_PROPERTY_NAME,
P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE,P_MISSING_MANDATORY_PROPERTY

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 createServiceManager(application: in TpClientAppID, serviceManager: out IpServiceRefRef). When the service agreement is signed for some serviceID (using signServiceAgreement()), the framework calls the createServiceManager() 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 IpSvcFactoryRef

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

Raises

TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID,P_INVALID_INTERFACE_TYPE

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. The service must be in the SCF Registered state. All instances of the service will be deleted.

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

TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID

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.

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.

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

TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID

Method

unannounceService()

This method results in the service no longer being discoverable by applications. It is, however, still registered and the service ID is still associated with it. Applications currently using the service can continue to use the service but no new applications should be able to start using the service. Also, all unused service tokens relating to the service will be

expired. This will prevent anyone who has already performed a selectService() but not yet performed the signServiceAgreement() from being able to obtain a new instance of the service.

Parameters

serviceID : in TpServiceID

The service ID of the service that is being unannounced. 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

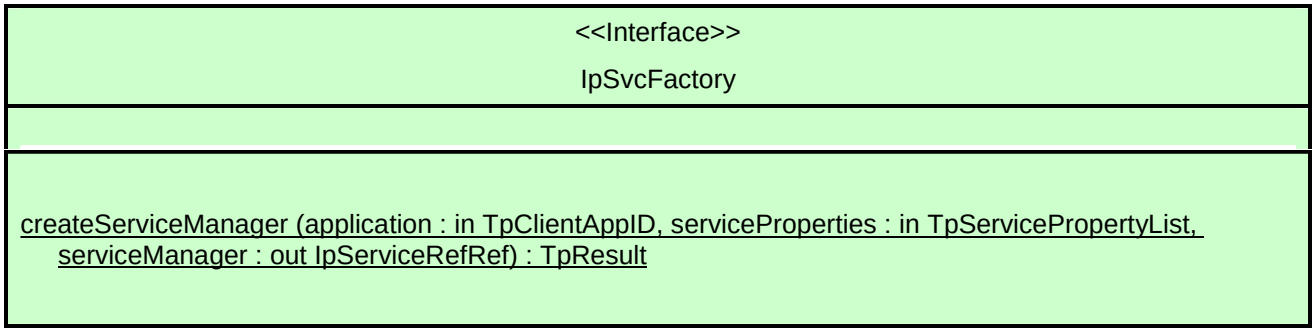
TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID

12.2 Service Factory Interface Classes

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.

12.2.1 Interface Class IpSvcFactory

Inherits from: IpInterface.



Method

createServiceManager()

This method returns a new service manager interface reference for the specified application. The service instance will be configured for the client application using the properties agreed in the service level agreement.

Parameters

application : in TpClientAppID

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

serviceProperties : in TpServicePropertyList

Specifies the service properties and their values that are to be used to configure the service instance. These properties form a part of the service level agreement. An example of these properties is a list of methods that the client application is allowed to invoke on the service interfaces.

serviceManager : out IpServiceRefRef

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

Raises

TpCommonExceptions,P_INVALID_PROPERTY

13 Framework-to-Service State Transition Diagrams

13.1 Service Registration State Transition Diagrams

13.1.1 State Transition Diagrams for IpFwServiceRegistration

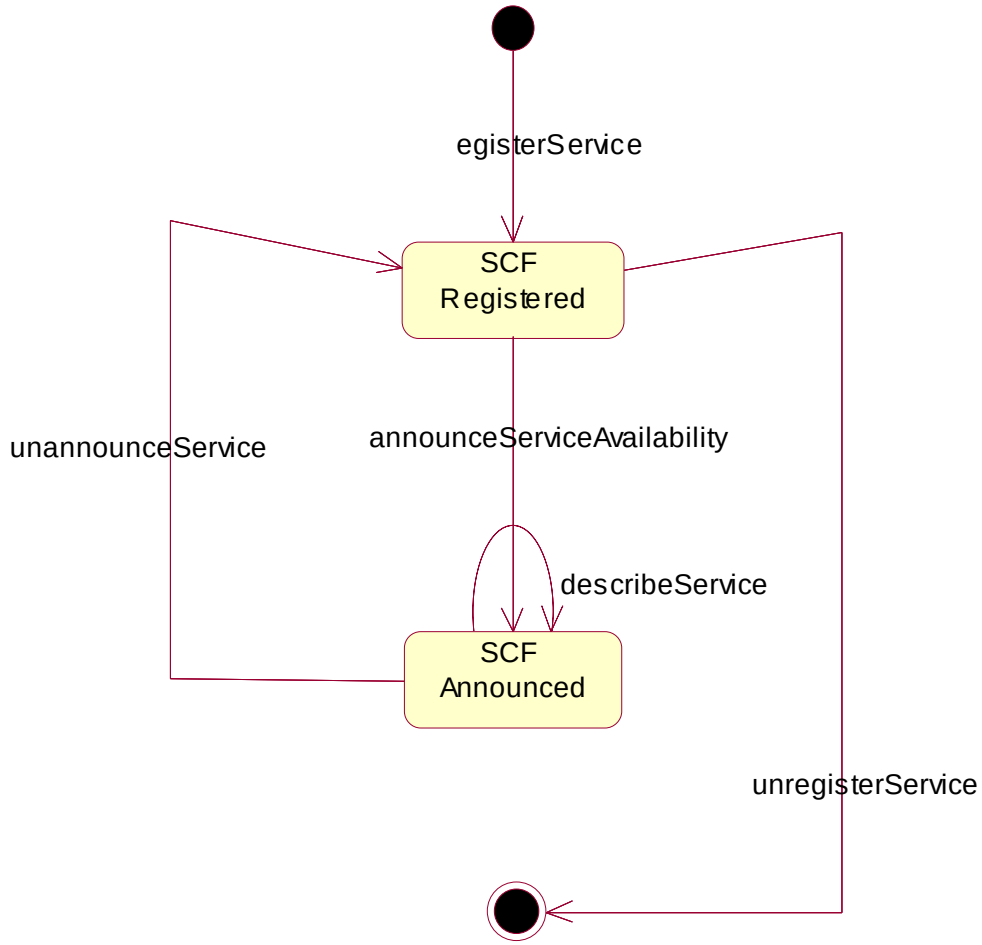


Figure : State Transition Diagram for IpFwServiceRegistration

13.1.1.1 SCF Registered State

This is the state entered when a Service Capability Server (SCS) registers 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.

An SCF may be unregistered, the service ID then being no longer associated with the SCF.

13.1.1.2 SCF Announced State

This is the state entered when the existence of the SCF has been announced, thus making it available for discovery by applications. The SCF can be unannounced at any time, taking it back into the SCF Registered state where it is no longer available for discovery.

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 supplier shall provide when he registers an SCF.

At Service Registration the properties of a type shall 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 property 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 shall be used for the property values instead 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 (see Note), the Service Properties shall thus be interpreted as the requested property values.

NOTE: This is achieved through the getServiceManager() operation in the Service Factory interface.

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
- 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 clause 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 (SCF).

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	
ServiceDescription~~ServicePropertyList~~	TpServiceDescription~~TpServicePropertyList~~	This field contains the description of the service

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.~~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

~~19.1.19TpUniqueServiceNumber~~

~~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).~~

~~19.1.20~~15.1.19 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	
Service<u>type</u>PropertyMode	TpService<u>Type</u>PropertyMode	
ServicePropertyTypeName	TpServicePropertyTypeName	

~~19.1.21~~15.1.20 TpServiceTypePropertyList

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

~~19.1.22~~15.1.21 TpService<u>Type</u>PropertyMode

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 shall be provided at service registration time

READONLY	2	The value of the corresponding SCF property type is optional, but once given a value it can ~~may~~ not be modified/restricted by a service level agreement
MANDATORY_READONLY	3	The value of the corresponding SCF property type shall be provided but can not ~~and~~ subsequently ~~it may not~~ be modified/restricted by a service level agreement.

~~19.1.23~~15.1.22 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.

~~19.1.24~~15.1.23 TpServicePropertyName

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

~~19.1.25~~15.1.24 TpServicePropertyNameList

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

~~19.1.26~~15.1.25 TpServicePropertyValue

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

~~19.1.27~~15.1.26 TpServicePropertyValueList

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

~~19.1.28~~15.1.27 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~~	

~~19.1.29~~15.1.28 TpServicePropertyList

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

~~19.1.30~~15.1.29 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.

~~19.1.31~~15.1.30 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)

~~19.1.32~~15.1.31 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

~~19.1.33~~15.1.32 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(s) available
P_EVENT_FW_SERVICE_UNAVAILABLE	2	Notification of SCS(s) becoming unavailable

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	ServiceType_Name_List
P_EVENT_FW_SERVICE_UNAVAILABLE	TpServiceTypeNameList	ServiceType Name List

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
P_EVENT_FW_ SERVICE_AVAILABLE	TpServiceIDList	ServiceID List
P_EVENT_FW_SERVICE_UNAVAILABLE	TpServiceIDList	ServiceID List

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 56-bit 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 128-bit 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 1024-bit 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.

~~19.3.8TpServiceAccessControl~~

~~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 shall also have a value.~~

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

~~19.3.9TpSecurityContext~~

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

~~19.3.10TpSecurityDomain~~

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

~~19.3.11TpSecurityGroup~~

~~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.~~

~~19.3.12TpServiceAccessType~~

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

~~19.3.13~~15.3.8 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.

~~19.3.14~~15.3.9 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.

~~19.3.15~~15.3.10 TpSigningAlgorithm

This data type is identical to a TpString, and is defined as a string of characters that identify the signing algorithm that shall 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

~~19.4.12TpTimeInterval~~

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

~~Sequence Element Name~~	~~Sequence Element Type~~
~~StartTime~~	~~TpDateAndTime~~
~~StopTime~~	~~TpDateAndTime~~

~~19.4.13~~15.4.12 TpLoadPolicy

Defines the load balancing policy.

Sequence Element Name	Sequence Element Type
LoadPolicy	TpString

~~19.4.14~~15.4.13 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

~~19.4.15~~15.4.14 TpLoadStatisticList

Defines a Numbered List of Data Elements of type TpLoadStatistic.

~~19.4.16~~15.4.15 TpLoadStatisticData

Defines the Sequence of Data Elements that represents load statistic information

Sequence Element Name	Sequence Element Type
LoadValue (see Note)	TpFloat
LoadLevel	TpLoadLevel
NOTE: LoadValue is expressed as a percentage.	

~~19.4.17~~15.4.16 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

~~19.4.18~~15.4.17 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

~~19.4.19~~15.4.18 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

~~19.4.20~~15.4.19 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

~~19.4.21~~15.4.20 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 Subscription Assignment Group (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 Subscription Assignment Group (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 TpServicePropertyList~~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 represents a service contract. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceContractID	TpServiceContractID
ServiceContractDescription	TpServiceContractDescription

15.5.21 TpServiceContractDescription

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
`ServiceRequestor`	TpServiceRequestor
`BillingContact`	TpBillingContact
`ServiceStartDate`	TpServiceStartDate
`ServiceEndDate`	TpServiceEndDate
`ServiceTypeName`	TpServiceTypeName
`ServiceID`	TpServiceID
`ServiceSubscriptionProperties`	TpServiceSubscriptionProperties

~~19.5.20TpServiceContract~~

~~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~~

~~19.5.21TpPassword~~

~~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 represents a Service Profile. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceProfileID	TpServiceProfileID
ServiceProfileDescription	TpServiceProfileDescription

15.5.31 TpServiceProfileDescription

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
ServiceContractID	TpServiceContractID
ServiceStartDate	TpServiceStartDate
ServiceEndDate	TpServiceEndDate
ServiceTypeName	TpServiceTypeName
ServiceSubscriptionProperties	TpServiceSubscriptionProperties

~~19.5.30TpServiceProfile~~

~~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~~

16 Exception Classes

The following are the list of exception classes which are used in this interface of the API.

Name	Description
P_ACCESS_DENIED	The client is not currently authenticated with the framework
P_APPLICATION_NOT_ACTIVATED	An application is unauthorised to access information and request services with regards to users that have deactivated that particular application.
P_DUPLICATE_PROPERTY_NAME	A dupilcate property name has been received
P_ILLEGAL_SERVICE_TYPE	Illegal Service Type
P_INVALID_ACCESS_TYPE	The framework does not support the type of access interface requested by the client.
P_INVALID_ACTIVITY_TEST_ID	ID does not correspond to a valid activity test request
P_INVALID_AGREEMENT_TEXT	Invalid agreement text
P_INVALID_AUTH_CAPABILITY	Invalid authentication capability
P_INVALID_AUTH_TYPE	Invalid type of authentication mechanism
P_INVALID_CLIENT_APP_ID	Invalid Client Application ID
P_INVALID_DOMAIN_ID	Invalid client ID
P_INVALID_ENT_OPP_ID	Invalid Enterprise Operator ID
P_INVALID_PROPERTY	The framework does not recognise the property supplied by the client
P_INVALID_SAG_ID	Invalid Subscription Assignment Group ID
P_INVALID_SERVICE_CONTRACT_ID	Invalid Service Contrac ID
P_INVALID_SERVICE_ID	Invalid service ID
P_INVALID_SERVICE_PROFILE_ID	Invalid service profile ID
P_INVALID_SERVICE_TOKEN	The service token has not been issued, or it has expired.

Name	Description
P_INVALID_SERVICE_TYPE	Invalid Service Type
P_INVALID_SIGNATURE	Invalid digital signature
P_INVALID_SIGNING_ALGORITHM	Invalid signing algorithm
P_MISSING_MANDATORY_PROPERTY	Mandatory Property Missing
P_NO_ACCEPTABLE_AUTH_CAPABILITY	An authentication mechanism, which is acceptable to the framework, is not supported by the client
P_PROPERTY_TYPE_MISMATCH	Property Type Mismatch
P_SERVICE_ACCESS_DENIED	The client application is not allowed to access this service.
P_SERVICE_ACCESS_TYPE	The framework does not support the type of access interface requested by the client.
P_SERVICE_NOT_ENABLED	The service ID does not correspond to a service that has been enabled
P_UNKNOWN_SERVICE_TYPE	Unknown Service Type

Each exception class contains the following structure:

Structure Element Name	Structure Element Type	Structure Element Description
extraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name

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 TS 29.198 R99

The following is a list of the differences between the present document and 3GPP TS 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

announceServiceAvailability (serviceID : in TpServiceID, serviceFactoryRef : in IpSvcFactory~~Service~~Ref) : TpResult

unannounceService(serviceID : in TpServiceID) : TpResult

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

Method accessCheck() has been removed

~~accessCheck(serviceToken: in TpServiceToken,securityContext: in TpStringTpSecurityContext, securityDomain: in TpStringTpSecurityDomain, group : in TpStringTpSecurityGroup, serviceAccessTypes: in TpStringTpServiceAccessType, 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

The following method is added to both interfaces:

authenticationSucceededResult (authenticated : in TpBoolean) : 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 IpSvcFactory

~~getServiceManager~~createServiceManager (application : in TpClientApp~~Domain~~ID, serviceProperties : in TpServicePropertyList, serviceManager : out IpServiceRefRef) : TpResult

B.9 All Interfaces

All methods on IpApp interfaces no longer throw exceptions.

All methods on the other interfaces throw TpCommonExceptions and individual, identified exceptions

B.10~~8~~ Data Type Changes

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~~ServiceDescription	~~TpServicePropertyList~~TpServiceDescription	This field contains the description of the service

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.~~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	
ServiceTypePropertyMode	TpServiceTypePropertyMode	
ServicePropertyTypeName	TpServicePropertyTypeName	

TpServiceTypePropertyList

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

TpServiceTypePropertyMode

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 shall be provided at service registration time
READONLY	2	The value of the corresponding SCF property type is optional, but once given a value it can ~~may~~ not be modified/restricted by a service level agreement
MANDATORY_READONLY	3	The value of the corresponding SCF property type shall be provided but can not ~~and~~ subsequently ~~it may not~~ be modified/restricted by a service level agreement.

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

Annex C (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
16 Mar 2001	CN_11	NP-010134	047	-	CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158)	3.2.0	4.0.0
1 Jun 2001	CN_12		001		CR 29.198: adding detailed exceptions for each method	4.0.0	4.0.1

```
//Source file: fw_data.idl
//Date:  8 June 2001


#ifndef __FW_DATA_DEFINED
#define __FW_DATA_DEFINED



#include "osa.idl"

module org {

      module csapi {



            typedef TpString TpAccessType;



            typedef TpInt32 TpActivityTestID;



            typedef TpString TpActivityTestRes;



            enum TpAPIUnavailReason {
                  API_UNAVAILABLE_UNDEFINED,
                  API_UNAVAILABLE_LOCAL_FAILURE,
                  API_UNAVAILABLE_GATEWAY_FAILURE,
                  API_UNAVAILABLE_OVERLOADED,
                  API_UNAVAILABLE_CLOSED,
                  API_UNAVAILABLE_PROTOCOL_FAILURE
            };



            typedef TpString TpAuthCapability;



            typedef TpString TpAuthCapabilityList;



            typedef TpString TpAuthType;



            typedef TpString TpClientAppID;

            typedef sequence <TpClientAppID> TpClientAppIDList;



            enum TpDomainIDType {
                  P_FW,
                  P_CLIENT_APPLICATION,
```

```
      P_ENT_OP,
      P_REGISTERED_SERVICE,
      P_SERVICE_SUPPLIER
};



typedef TpString TpEmail;

typedef TpString TpEntOpID;

typedef sequence <TpEntOpID> TpEntOpIDList;



enum TpFwEventName {
      P_EVENT_FW_NAME_UNDEFINED,
      P_EVENT_FW_SERVICE_AVAILABLE,
      P_EVENT_FW_SERVICE_UNAVAILABLE
};



enum TpFWExceptionType {
      P_FW_DUMMY
};



exception TpFWException {
      TpFWExceptionType exceptionType;
};



typedef TpString TpFwID;



enum TpFwUnavailReason {
      FW_UNAVAILABLE_UNDEFINED,
      FW_UNAVAILABLE_LOCAL_FAILURE,
      FW_UNAVAILABLE_GATEWAY_FAILURE,
      FW_UNAVAILABLE_OVERLOADED,
      FW_UNAVAILABLE_CLOSED,
      FW_UNAVAILABLE_PROTOCOL_FAILURE
};



typedef TpString TpHomePage;



enum TpInterfaceFault {
      INTERFACE_FAULT_UNDEFINED,
      INTERFACE_FAULT_LOCAL_FAILURE,
      INTERFACE_FAULT_GATEWAY_FAILURE,
      INTERFACE_FAULT_PROTOCOL_ERROR
};
```

```
struct TpFaultStats {
      TpInterfaceFault Fault;
      TpInt32 Occurrences;
      TpInt32 MaxDuration;
      TpInt32 TotalDuration;
      TpInt32 NumberOfClientsAffected;
};


typedef sequence <TpFaultStats> TpFaultStatsSet;


struct TpFaultStatsRecord {
      TpTimeInterval Period;
      TpFaultStatsSet FaultStatsSet;
};


typedef TpString TpInterfaceName;


enum TpLoadLevel {
      LOAD_LEVEL_NORMAL,
      LOAD_LEVEL_OVERLOAD,
      LOAD_LEVEL_SEVERE_OVERLOAD
};


struct TpLoadPolicy {
      TpString LoadPolicy;
};

struct TpLoadStatisticData {
      TpFloat LoadValue;
      TpLoadLevel LoadLevel;
};


enum TpLoadStatisticEntityType {
      P_LOAD_STATISTICS_FW_TYPE,
      P_LOAD_STATISTICS_SVC_TYPE,
      P_LOAD_STATISTICS_APP_TYPE
};


enum TpLoadStatisticInfoType {
      P_LOAD_STATISTICS_VALID,
      P_LOAD_STATISTICS_INVALID
};
```

```
enum TpLoadStatusError {
      LOAD_STATUS_ERROR_UNDEFINED,
      LOAD_STATUS_ERROR_UNAVAILABLE
};



struct TpLoadThreshold {
      TpFloat LoadThreshold;
};



struct TpLoadInitVal {
      TpLoadLevel LoadLevel;
      TpLoadThreshold LoadThreshold;
};



typedef TpString TpPersonName;



typedef TpString TpPostalAddress;



typedef TpString TpPropertyName;



typedef TpString TpPropertyValue;

struct TpProperty {
      TpPropertyName PropertyName;
      TpPropertyValue PropertyValue;
};



typedef sequence <TpProperty> TpPropertyList;



typedef TpPropertyList TpClientAppProperties;



struct TpClientAppDescription {
      TpClientAppID ClientAppID;
      TpClientAppProperties ClientAppProperties;
};

typedef TpPropertyList TpEndAccessProperties;



typedef TpPropertyList TpEntOpProperties;
```

```
struct TpEntOp {
      TpEntOpID EntOpID;
      TpEntOpProperties EntOpProperties;
};

typedef TpPropertyList TpPersonProperties;

typedef TpString TpSagDescription;



typedef TpString TpSagID;

struct TpSag {
      TpSagID SagID;
      TpSagDescription SagDescription;
};



typedef sequence <TpSagID> TpSagIDList;

typedef TpString TpServiceContractID;



typedef TpDateAndTime TpServiceEndDate;



typedef TpString TpServiceID;

typedef sequence <TpServiceID> TpServiceIDList;



typedef TpString TpServiceProfileID;

typedef sequence <TpServiceProfileID> TpServiceProfileIDList;



enum TpServiceTypePropertyMode {
      NORMAL,
      MANDATORY,
      _READONLY,
      MANDATORY_READONLY
};

typedef TpString TpServicePropertyName;

typedef sequence <TpServicePropertyName> TpServicePropertyNameList;

typedef TpString TpServicePropertyTypeName;

typedef TpString TpServicePropertyValue;

typedef sequence <TpServicePropertyValue>
TpServicePropertyValueList;
```

```
            struct TpServiceProperty {
                  TpServicePropertyName ServicePropertyName;
                  TpServicePropertyValueList ServicePropertyValueList;
            };

            typedef sequence <TpServiceProperty> TpServicePropertyList;



            typedef TpString TpServiceSpecString;



            typedef TpDateAndTime TpServiceStartDate;



            typedef TpServicePropertyList TpServiceSubscriptionProperties;

            typedef TpString TpServiceSupplierID;



            union TpDomainID switch(TpDomainIDType) {
                  case P_FW: TpFwID FwID;
                  case P_CLIENT_APPLICATION: TpClientAppID ClientAppID;
                  case P_ENT_OP: TpEntOpID EntOpID;
                  case P_REGISTERED_SERVICE: TpServiceID ServiceID;
                  case P_SERVICE_SUPPLIER: TpServiceSupplierID
ServiceSupplierID;
            };

            struct TpAuthDomain {
                  TpDomainID DomainID;
                  IpInterface AuthInterface;
            };

            typedef TpString TpServiceToken;

            typedef TpString TpServiceTypeName;



            struct TpServiceDescription {
                  TpServiceTypeName ServiceTypeName;
                  TpServicePropertyList ServicePropertyList;
            };



            struct TpService {
                  TpServiceID ServiceID;
                  TpServiceDescription ServiceDescription;
            };

            typedef sequence <TpService> TpServiceList;
```

```
            struct TpServiceProfileDescription {
                  TpServiceContractID ServiceContractID;
                  TpServiceStartDate ServiceStartDate;
                  TpServiceEndDate ServiceEndDate;
                  TpServiceTypeName ServiceTypeName;
                  TpServiceSubscriptionProperties ServiceSubscriptionProperties;
            };

            typedef sequence <TpServiceTypeName> TpServiceTypeNameList;



            union TpFwEventCriteria switch(TpFwEventName) {
                  case P_EVENT_FW_NAME_UNDEFINED: TpString EventNameUndefined;
                  case P_EVENT_FW_SERVICE_AVAILABLE: TpServiceTypeNameList
ServiceTypeNameList;
                  case P_EVENT_FW_SERVICE_UNAVAILABLE: TpServiceTypeNameList
UnavailableServiceTypeNameList;
            };

            union TpFwEventInfo switch(TpFwEventName) {
                  case P_EVENT_FW_NAME_UNDEFINED: TpString EventNameUndefined;
                  case P_EVENT_FW_SERVICE_AVAILABLE: TpServiceTypeNameList
ServiceIDList;
                  case P_EVENT_FW_SERVICE_UNAVAILABLE: TpServiceTypeNameList
UnavailableServiceIDList;
            };



            struct TpServiceTypeProperty {
                  TpServicePropertyName ServicePropertyName;
                  TpServiceTypePropertyMode ServiceTypePropertyMode;
                  TpServicePropertyTypeName ServicePropertyTypeName;
            };



            typedef sequence <TpServiceTypeProperty> TpServiceTypePropertyList;



            struct TpServiceTypeDescription {
                  TpServiceTypePropertyList ServiceTypePropertyList;
                  TpServiceTypeNameList ServiceTypeNameList;
                  TpBoolean EnabledOrDisabled;
            };



            struct TpSignatureAndServiceMgr {
                  TpString DigitalSignature;
                  IpService ServiceMgrInterface;
            };



            typedef TpString TpSigningAlgorithm;
```

```
enum TpSvcUnavailReason {
      SERVICE_UNAVAILABLE_UNDEFINED,
      SERVICE_UNAVAILABLE_LOCAL_FAILURE,
      SERVICE_UNAVAILABLE_GATEWAY_FAILURE,
      SERVICE_UNAVAILABLE_OVERLOADED,
      SERVICE_UNAVAILABLE_CLOSED
};



typedef TpString TpTelephoneNumber;



struct TpPerson {
      TpPersonName PersonName;
      TpPostalAddress PostalAddress;
      TpTelephoneNumber TelephoneNumber;
      TpEmail Email;
      TpHomePage HomePage;
      TpPersonProperties PersonProperties;
};

typedef TpPerson TpBillingContact;



typedef TpPerson TpServiceRequestor;



struct TpServiceContractDescription {
      TpServiceRequestor ServiceRequestor;
      TpBillingContact BillingContact;
      TpServiceStartDate ServiceStartDate;
      TpServiceEndDate ServiceEndDate;
      TpServiceTypeName ServiceTypeName;
      TpServiceID ServiceID;
      TpServiceSubscriptionProperties ServiceSubscriptionProperties;
};

union TpLoadStatisticEntityID switch(TpLoadStatisticEntityType) {
      case P_LOAD_STATISTICS_FW_TYPE: TpFwID FrameworkID;
      case P_LOAD_STATISTICS_SVC_TYPE: TpServiceID ServiceID;
      case P_LOAD_STATISTICS_APP_TYPE: TpClientAppID ClientAppID;
};



enum TpLoadStatisticError {
      P_LOAD_INFO_ERROR_UNDEFINED,
      P_LOAD_INFO_UNAVAILABLE
};

typedef sequence <TpLoadStatisticError> TpLoadStatisticErrorList;

union TpLoadStatisticInfo switch(TpLoadStatisticInfoType) {
      case P_LOAD_STATISTICS_VALID: TpLoadStatisticData
LoadStatisticData;
      case P_LOAD_STATISTICS_INVALID: TpLoadStatisticError
LoadStatisticError;
```

```
};


struct TpLoadStatistic {
      TpLoadStatisticEntityID LoadStatisticEntityID;
      TpDateAndTime TimeStamp;
      TpLoadStatisticInfo LoadStatisticInfo;
};

typedef sequence <TpLoadStatistic> TpLoadStatisticList;


exception P_INVALID_SERVICE_ID {
      TpString extraInformation;
};


exception P_SERVICE_ACCESS_DENIED {
      TpString extraInformation;
};


exception P_ACCESS_DENIED {
      TpString extraInformation;
};


exception P_SERVICE_NOT_ENABLED {
      TpString extraInformation;
};


exception P_SERVICE_ACCESS_TYPE {
      TpString extraInformation;
};


exception P_INVALID_AUTH_CAPABILITY {
      TpString extraInformation;
};


exception P_NO_ACCEPTABLE_AUTH_CAPABILITY {
      TpString extraInformation;
};


exception P_INVALID_AGREEMENT_TEXT {
      TpString extraInformation;
};
```

```
exception P_INVALID_SERVICE_TOKEN {
      TpString extraInformation;
};



exception P_INVALID_SIGNATURE {
      TpString extraInformation;
};



exception P_INVALID_SIGNING_ALGORITHM {
      TpString extraInformation;
};



exception P_INVALID_DOMAIN_ID {
      TpString extraInformation;
};



exception P_APPLICATION_NOT_ACTIVATED {
      TpString extraInformation;
};



exception P_INVALID_PROPERTY {
      TpString extraInformation;
};

struct TpServiceContract {
      TpServiceContractID ServiceContractID;
      TpServiceContractDescription ServiceContractDescription;
};

struct TpServiceProfile {
      TpServiceProfileID ServiceProfileID;
      TpServiceProfileDescription ServiceProfileDescription;
};

exception P_INVALID_ACCESS_TYPE {
      TpString extraInformation;
};

exception P_ILLEGAL_SERVICE_TYPE {
      TpString extraInformation;
};

exception P_UNKNOWN_SERVICE_TYPE {
      TpString extraInformation;
};

exception P_MISSING_MANDATORY_PROPERTY {
      TpString extraInformation;
};
```

```
            exception P_DUPLICATE_PROPERTY_NAME {
                  TpString extraInformation;
            };

            exception P_PROPERTY_TYPE_MISMATCH {
                  TpString extraInformation;
            };

            exception P_INVALID_SERVICE_TYPE {
                  TpString extraInformation;
            };

            exception P_INVALID_CLIENT_APP_ID {
                  TpString extraInformation;
            };

            exception P_INVALID_AUTH_TYPE {
                  TpString extraInformation;
            };

            exception P_INVALID_SAG_ID {
                  TpString extraInformation;
            };

            exception P_INVALID_SERVICE_PROFILE_ID {
                  TpString extraInformation;
            };

            exception P_INVALID_SERVICE_CONTRACT_ID {
                  TpString extraInformation;
            };

            exception P_INVALID_ACTIVITY_TEST_ID {
                  TpString extraInformation;
            };

            exception P_INVALID_ENT_OP_ID {
                  TpString extraInformation;
            };

            exception P_ILLEGAL_SERVICE_ID {
                  TpString extraInformation;
            };

            exception P_UNKNOWN_SERVICE_ID {
                  TpString extraInformation;
            };

      };

};

#endif
```

```
//Source file: fw_if_3gpp.idl
//Date:  8 June 2001

#ifndef __FW_IF_3GPP_DEFINED
#define __FW_IF_3GPP_DEFINED



#include "osa.idl"
#include "fw_data.idl"


module org {

      module csapi {

            module fw_client {

                  module notification {



                        interface IpAppEventNotification : IpInterface {

                              void reportNotification (
                                    in TpFwEventInfo eventInfo,
                                    in TpAssignmentID assignmentID
                                    );


                              void notificationTerminated ();

                        };



                        interface IpEventNotification : IpInterface {

                              void createNotification (
                                    in TpFwEventCriteria eventCriteria,
                                    out TpAssignmentID assignmentID
                                    )
                                    raises (TpCommonExceptions, P_ACCESS_DENIED,
P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE);


                              void destroyNotification (
                                    in TpAssignmentID assignmentID
                                    )
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_ASSIGNMENT_ID);

                        };

                  };

                  module integrity {



                        interface IpAppFaultManager : IpInterface {
```

```
                        void activityTestRes (
                              in TpActivityTestID activityTestID,
                              in TpActivityTestRes activityTestResult
                              );


                        void appActivityTestReq (
                              in TpActivityTestID activityTestID
                              );


                        void fwFaultReportInd (
                              in TpInterfaceFault fault
                              );


                        void fwFaultRecoveryInd (
                              in TpInterfaceFault fault
                              );


                        void svcUnavailableInd (
                              in TpServiceID serviceId,
                              in TpSvcUnavailReason reason
                              );


                        void genFaultStatsRecordRes (
                              in TpFaultStatsRecord faultStatistics,
                              in TpServiceIDList serviceIDs
                              );


                        void fwUnavailableInd (
                              in TpFwUnavailReason reason
                              );

                  };



                  interface IpAppHeartBeat : IpInterface {


                        void send (
                              in TpSessionID session
                              );

                  };



                  interface IpHeartBeat : IpInterface {


                        void send (
                              in TpSessionID session
                              )
                              raises
(TpCommonExceptions,P_INVALID_SESSION_ID);
```

```
                        };


                        interface IpAppHeartBeatMgmt : IpInterface {

                              void enableAppHeartBeat (
                                    in TpDuration duration,
                                    in IpHeartBeat fwInterface,
                                    in TpSessionID session
                                    );

                              void disableAppHeartBeat (
                                    in TpSessionID session
                                    );

                              void changeTimePeriod (
                                    in TpDuration duration,
                                    in TpSessionID session
                                    );

                        };


                        interface IpHeartBeatMgmt : IpInterface {

                              void enableHeartBeat (
                                    in TpDuration duration,
                                    in IpAppHeartBeat appInterface,
                                    out TpSessionID session
                                    )
                                    raises
(TpCommonExceptions,P_INVALID_SESSION_ID);

                              void disableHeartBeat (
                                    in TpSessionID session
                                    )
                                    raises
(TpCommonExceptions,P_INVALID_SESSION_ID);

                              void changeTimePeriod (
                                    in TpDuration duration,
                                    in TpSessionID session
                                    )
                                    raises
(TpCommonExceptions,P_INVALID_SESSION_ID);

                        };


                        interface IpAppLoadManager : IpInterface {
```

```
                        void queryAppLoadReq (
                              in TpServiceIDList serviceIDs,
                              in TpTimeInterval timeInterval
                              );


                        void queryLoadRes (
                              in TpLoadStatisticList loadStatistics
                              );


                        void queryLoadErr (
                              in TpLoadStatisticError loadStatisticsError

                              );


                        void loadLevelNotification (
                              in TpLoadStatisticList loadStatistics
                              );


                        void resumeNotification ();


                        void suspendNotification ();

                  };



                  interface IpLoadManager : IpInterface {

                        void reportLoad (
                              in TpLoadLevel loadLevel
                              )
                              raises (TpCommonExceptions);


                        void queryLoadReq (
                              in TpServiceIDList serviceIDs,
                              in TpTimeInterval timeInterval
                              )
                              raises
(TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED);


                        void queryAppLoadRes (
                              in TpLoadStatisticList loadStatistics
                              )
                              raises (TpCommonExceptions);


                        void queryAppLoadErr (
                              in TpLoadStatisticError loadStatisticsError

                              )
                              raises (TpCommonExceptions);


                        void registerLoadController (
```

```
                                    in TpServiceIDList serviceIDs
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SERVICE_ID);


                              void unregisterLoadController (
                                    in TpServiceIDList serviceIDs
                                    )
                                    raises
(TpCommonExceptions,P_INVALID_SERVICE_ID);


                              void resumeNotification (
                                    in TpServiceIDList serviceIDs
                                    )
                                    raises
(TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED);


                              void suspendNotification (
                                    in TpServiceIDList serviceIDs
                                    )
                                    raises
(TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED);

                        };



                        interface IpAppOAM : IpInterface {

                              void systemDateTimeQuery (
                                    in TpDateAndTime systemDateAndTime,
                                    out TpDateAndTime clientDateAndTime
                                    );

                        };



                        interface IpOAM : IpInterface {

                              void systemDateTimeQuery (
                                    in TpDateAndTime clientDateAndTime,
                                    out TpDateAndTime systemDateAndTime
                                    )
                                    raises
(TpCommonExceptions,P_INVALID_TIME_AND_DATE_FORMAT);

                        };



                        interface IpFaultManager : IpInterface {

                              void activityTestReq (
                                    in TpActivityTestID activityTestID,
                                    in TpServiceID svcID
                                    )
```

```
                                    raises
(TpCommonExceptions,P_INVALID_SERVICE_ID);


                              void appActivityTestRes (
                                    in TpActivityTestID activityTestID,
                                    in TpActivityTestRes activityTestResult
                                    )
                                    raises
(TpCommonExceptions,P_INVALID_SERVICE_ID,P_INVALID_ACTIVITY_TEST_ID);


                              void svcUnavailableInd (
                                    in TpServiceID serviceID
                                    )
                                    raises (TpCommonExceptions
,P_INVALID_SERVICE_ID);


                              void genFaultStatsRecordReq (
                                    in TpTimeInterval timePeriod,
                                    in TpServiceIDList serviceIDs
                                    )
                                    raises (TpCommonExceptions
,P_INVALID_SERVICE_ID);

                        };

                  };


                  module discovery {


                        interface IpServiceDiscovery : IpInterface {

                              void listServiceTypes (
                                    out TpServiceTypeNameList listTypes
                                    )
                                    raises (TpCommonExceptions,P_ACCESS_DENIED);


                              void describeServiceType (
                                    in TpServiceTypeName name,
                                    out TpServiceTypeDescription
serviceTypeDescription
                                    )
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYP
E);


                              void discoverService (
                                    in TpServiceTypeName serviceTypeName,
                                    in TpServicePropertyList
desiredPropertyList,
                                    in TpInt32 max,
                                    out TpServiceList serviceList
                                    )
```

```
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYP
E,P_INVALID_PROPERTY);


                              void listSubscribedServices (
                                    out TpServiceList serviceList
                                    )
                                    raises (TpCommonExceptions,P_ACCESS_DENIED);

                        };

                  };



                  module trust_and_security {



                        interface IpInitial : IpInterface {

                              void initiateAuthentication (
                                    in TpAuthDomain appDomain,
                                    in TpAuthType authType,
                                    out TpAuthDomain fwDomain
                                    )
                                    raises
(TpCommonExceptions,P_INVALID_DOMAIN_ID,P_INVALID_INTERFACE_TYPE,P_INVALID_AUTH_
TYPE);

                        };



                        interface IpAuthentication : IpInterface {

                              void requestAccess (
                                    in TpAccessType accessType,
                                    in IpInterface appAccessInterface,
                                    out IpInterface fwAccessInterface
                                    )
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_ACCESS_TYPE,P_INVALID_INTERFACE_TY
PE);

                        };



                        interface IpAppAccess : IpInterface {

                              void signServiceAgreement (
                                    in TpServiceToken serviceToken,
                                    in TpString agreementText,
                                    in TpSigningAlgorithm signingAlgorithm,
                                    out TpString digitalSignature
                                    );


                              void terminateServiceAgreement (
```

```
                                    in TpServiceToken serviceToken,
                                    in TpString terminationText,
                                    in TpString digitalSignature
                                    );


                              void terminateAccess (
                                    in TpString terminationText,
                                    in TpSigningAlgorithm signingAlgorithm,
                                    in TpString digitalSignature
                                    );

                        };

                        interface IpAccess : IpInterface {

                              void obtainInterface (
                                    in TpInterfaceName interfaceName,
                                    out IpInterface fwInterface
                                    )
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_INTERFACE_NAME);


                              void obtainInterfaceWithCallback (
                                    in TpInterfaceName interfaceName,
                                    in IpInterface appInterface,
                                    out IpInterface fwInterface
                                    )
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_INTERFACE_NAME,P_INVALID_INTERFACE
_TYPE);


                              void selectService (
                                    in TpServiceID serviceID,
                                    out TpServiceToken serviceToken
                                    )
                                    raises (TpCommonExceptions,P_ACCESS_DENIED,
P_INVALID_SERVICE_ID);


                              void signServiceAgreement (
                                    in TpServiceToken serviceToken,
                                    in TpString agreementText,
                                    in TpSigningAlgorithm signingAlgorithm,
                                    out TpSignatureAndServiceMgr
signatureAndServiceMgr
                                    )
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AGREEMENT_TEXT,P_INVALID_SERVICE_T
OKEN,P_INVALID_SIGNING_ALGORITHM,P_SERVICE_ACCESS_DENIED);


                              void terminateServiceAgreement (
                                    in TpServiceToken serviceToken,
                                    in TpString terminationText,
                                    in TpString digitalSignature
                                    )
```

```
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_SERVICE_TOKEN,P_INVALID_SIGNATURE)
;


                              void endAccess (
                                    in TpEndAccessProperties endAccessProperties

                                    )
                                    raises (TpCommonExceptions,P_ACCESS_DENIED,
P_INVALID_PROPERTY);

                        };

                        interface IpAppAPILevelAuthentication : IpInterface {

                              void authenticate (
                                    in TpAuthCapability prescribedMethod,
                                    in TpString challenge,
                                    out TpString response
                                    );


                              void abortAuthentication ();


                              void authenticationSucceeded ();

                        };



                        interface IpAPILevelAuthentication : IpAuthentication {

                              void selectEncryptionMethod (
                                    in TpAuthCapabilityList authCaps,
                                    out TpAuthCapability prescribedMethod
                                    )
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_NO_ACCEPTABLE_AUTH_CAPABILITY);


                              void authenticate (
                                    in TpAuthCapability prescribedMethod,
                                    in TpString challenge,
                                    out TpString response
                                    )
                                    raises
(TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AUTH_CAPABILITY);


                              void abortAuthentication ()

                                    raises (TpCommonExceptions,P_ACCESS_DENIED);


                              void authenticationSucceeded ()

                                    raises (TpCommonExceptions,P_ACCESS_DENIED);

                        };
```

```
		};


	};

	module fw_service {

		module service_factory {

			interface IpSvcFactory : IpInterface {

				void createServiceManager (
					in TpClientAppID application,
					in TpServicePropertyList serviceProperties,

					out IpService serviceManager
					)
					raises
(TpCommonExceptions,P_INVALID_PROPERTY);

			};

		};


		module service_registration {



			interface IpFwServiceRegistration : IpInterface {

				void registerService (
					in TpServiceTypeName serviceTypeName,
					in TpServicePropertyList
servicePropertyList,
					out TpServiceID serviceID
					)
					raises
(TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID,P_PROPERTY_TYPE_MI
SMATCH,P_DUPLICATE_PROPERTY_NAME,
	P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE,P_MISSING_MANDATORY_PROPERTY
);


				void announceServiceAvailability (
					in TpServiceID serviceID,
					in service_factory::IpSvcFactory
serviceFactoryRef
					)
					raises
(TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID,P_INVALID_INTERFAC
E_TYPE);


				void unregisterService (
					in TpServiceID serviceID
					)
					raises
(TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID);
```

```
                                    void describeService (
                                          in TpServiceID serviceID,
                                          out TpServiceDescription serviceDescription

                                          )
                                          raises
(TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID);


                                    void unannounceService (
                                          in TpServiceID serviceID
                                          )
                                          raises
(TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID);

                              };

                        };

                  };

            };

};

#endif
```

3GPP TSG_CN5 (Open Service Access – OSA) **Tdoc N5-010263**
Meeting #11, San Diego, CA, USA, 21 – 24 May 2001

CR-Form-v3

CHANGE REQUEST

⌘ **29.198-5** **CR** **001** ⌘ rev **-** ⌘ Current version: **4.0.0** ⌘

*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:	⌘	Corrections to OSA API Rel4
Source:	⌘	CN5
Work item code:	⌘	OSA1
Date:	⌘	07/06/2001
Category:	⌘	**F**
Release:	⌘	Rel4

Use one of the following categories:
F *(essential correction)*
A *(corresponds to a correction in an earlier release)*
B *(Addition of feature),*
C *(Functional modification of feature)*
D *(Editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
REL-4	*(Release 4)*
REL-5	*(Release 5)*

Reason for change:	⌘	Exception handling mechanism in 29.198 requires correction to enable it to be correctly used, without ambiguity
Summary of change:	⌘	Replace TpGeneralException, TpUIException with detailed exception classes which can be thrown for each method
Consequences if not approved:	⌘	29.198-5 will be ambiguous and difficult to implement correctly - inter-working might be jeopardised.

Clauses affected:	⌘				
Other specs affected:	⌘	**X**	Other core specifications	⌘	All other parts of 29.198 except part 1 have similar changes
		☐	Test specifications		
		☐	O&M Specifications		
Other comments:	⌘				

3GPP TS 29.198-5 V4.0.1~~0~~ (2001-~~03~~06)

Technical Specification

3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API); Part 5: Generic User Interaction (Release 4)

Keywords
UMTS, 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.

Introduction

The present document is part 5 of a multi-part TS covering the 3rd Generation Partnership Project: Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API), as identified below. The **API specification** (3GPP TS 29.198) is structured in the following Parts:

Part 1:	Overview	
Part 2:	Common Data Definitions	
Part 3:	Framework	
Part 4:	Call Control SCF	
Part 5:	**User Interaction SCF**	
Part 6:	Mobility SCF	
Part 7:	Terminal Capabilities SCF	
Part 8:	Data Session Control SCF	
Part 9:	Generic Messaging SCF	(not part of 3GPP Release 4)
Part 10:	Connectivity Manager SCF	(not part of 3GPP Release 4)
Part 11:	Account Management SCF	
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.

OSA API specifications 29.198-family		OSA API Mapping - 29.998-family	
29.198-1	**Part 1: Overview**	**29.998-1**	**Part 1: Overview**
29.198-2	**Part 2: Common Data Definitions**	29.998-2	Not Applicable
29.198-3	**Part 3: Framework**	29.998-3	Not Applicable
29.198-4	**Part 4: Call Control SCF**	**29.998-4-1**	**Subpart 1: Generic Call Control – CAP mapping**
		29.998-4-2	
29.198-5	**Part 5: User Interaction SCF**	**29.998-5-1**	**Subpart 1: User Interaction – CAP mapping**
		29.998-5-2	
		29.998-5-3	
		29.998-5-4	**Subpart 4: User Interaction – SMS mapping**
29.198-6	**Part 6: Mobility SCF**	**29.998-6**	**User Status and User Location – MAP mapping**
29.198-7	**Part 7: Terminal Capabilities SCF**	29.998-7	Not Applicable
29.198-8	**Part 8: Data Session Control SCF**	**29.998-8**	**Data Session Control – CAP mapping**
29.198-9	Part 9: Generic Messaging SCF	29.998-9	Not Applicable
29.198-10	Part 10: Connectivity Manager SCF	29.998-10	Not Applicable
29.198-11	**Part 11: Account Management SCF**	29.998-11	Not Applicable
29.198-12	**Part 12: Charging SCF**	29.998-12	Not Applicable

1 Scope

This document is Part 5 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 APIs. The concepts and the functional architecture for the OSA are contained in 3GPP TS 23.127 [3]. The requirements for OSA are contained in 3GPP TS 22.127 [2].

The present document specifies the User Interaction (UI) 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, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[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 and abbreviations

3.1 Definitions

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

3.2 Abbreviations

For the purposes of the present document, the abbreviations given 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).

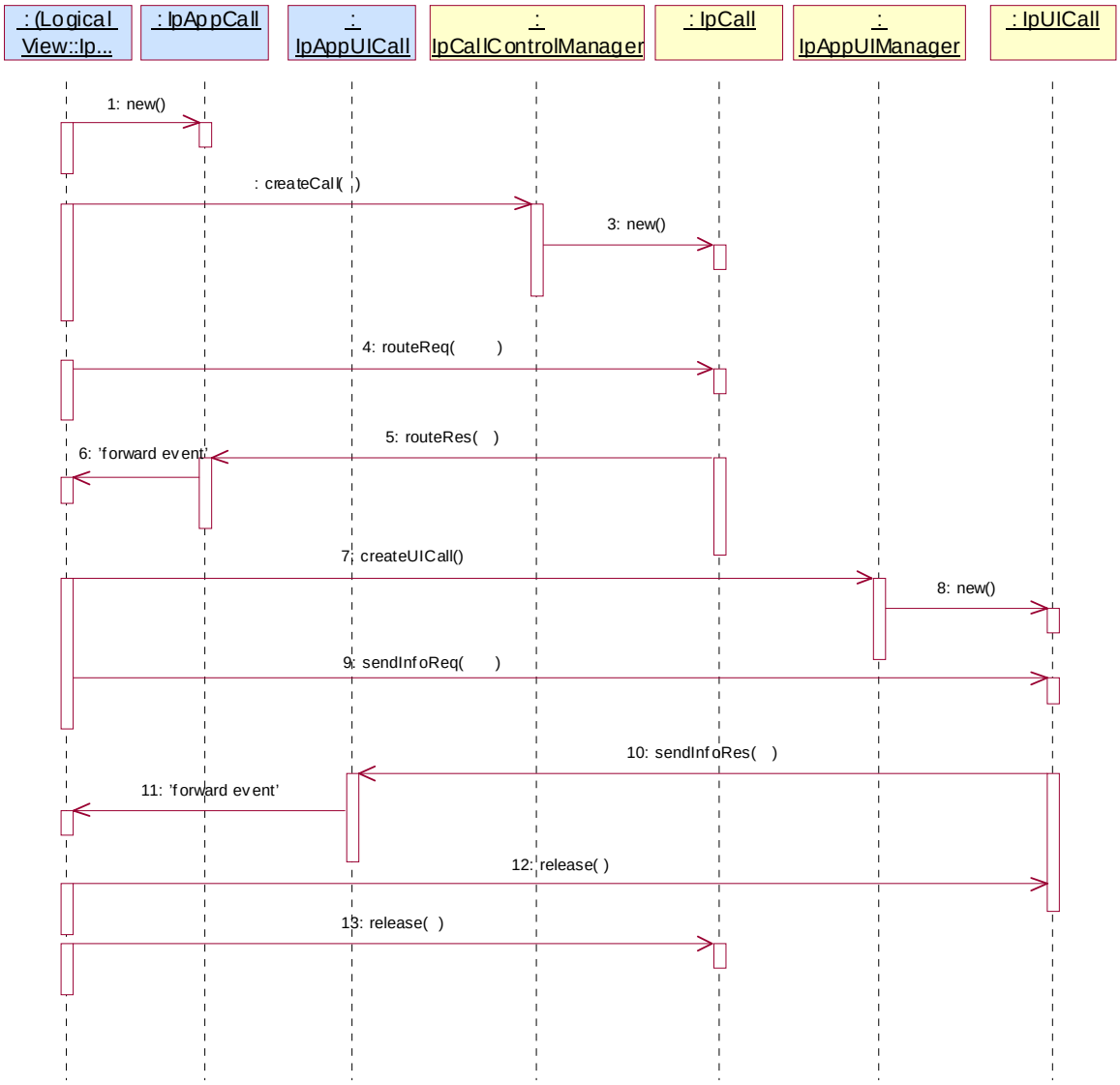
The order is as follows:

- 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.

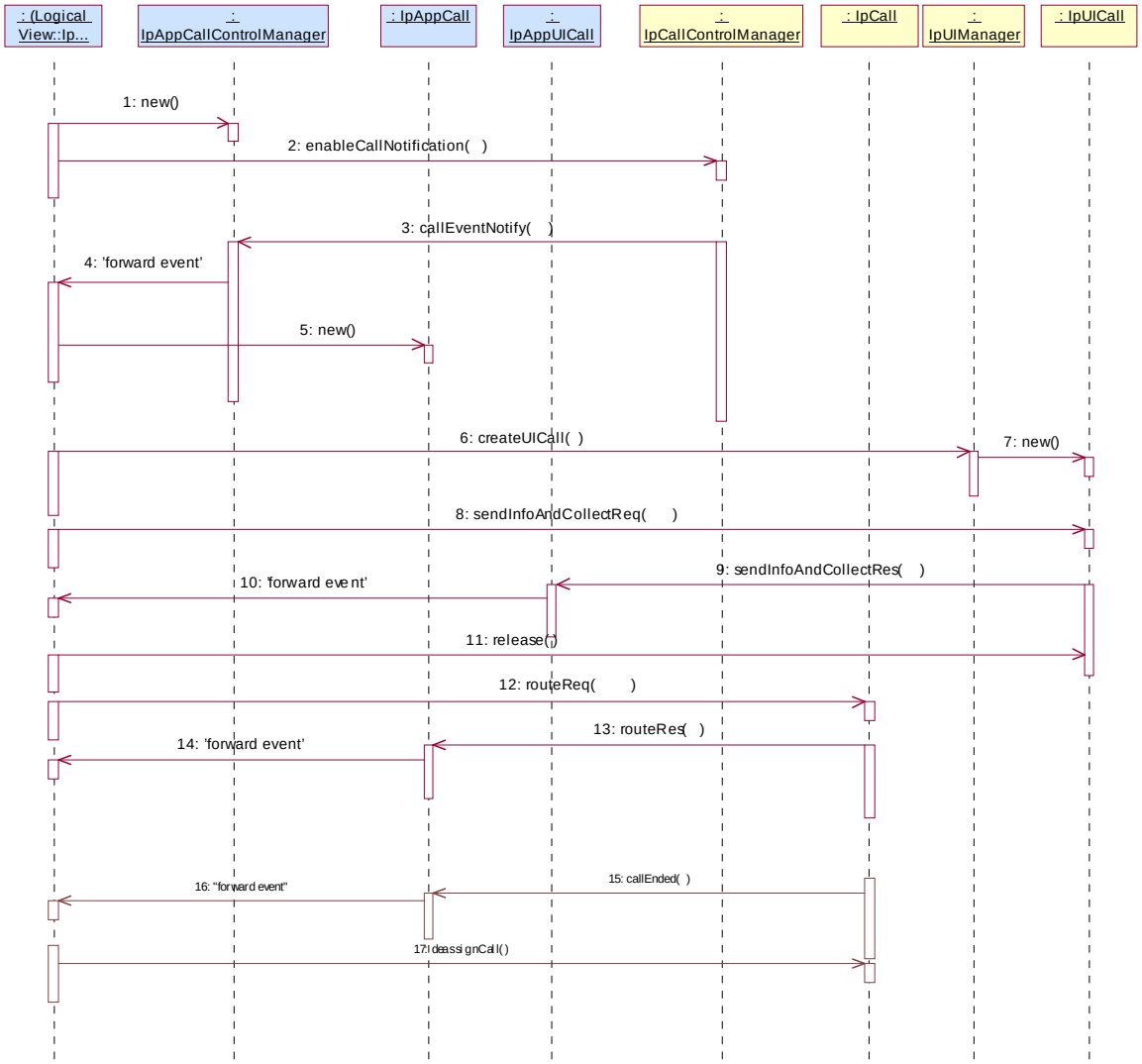
11: The event is forwarded to the application logic.

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

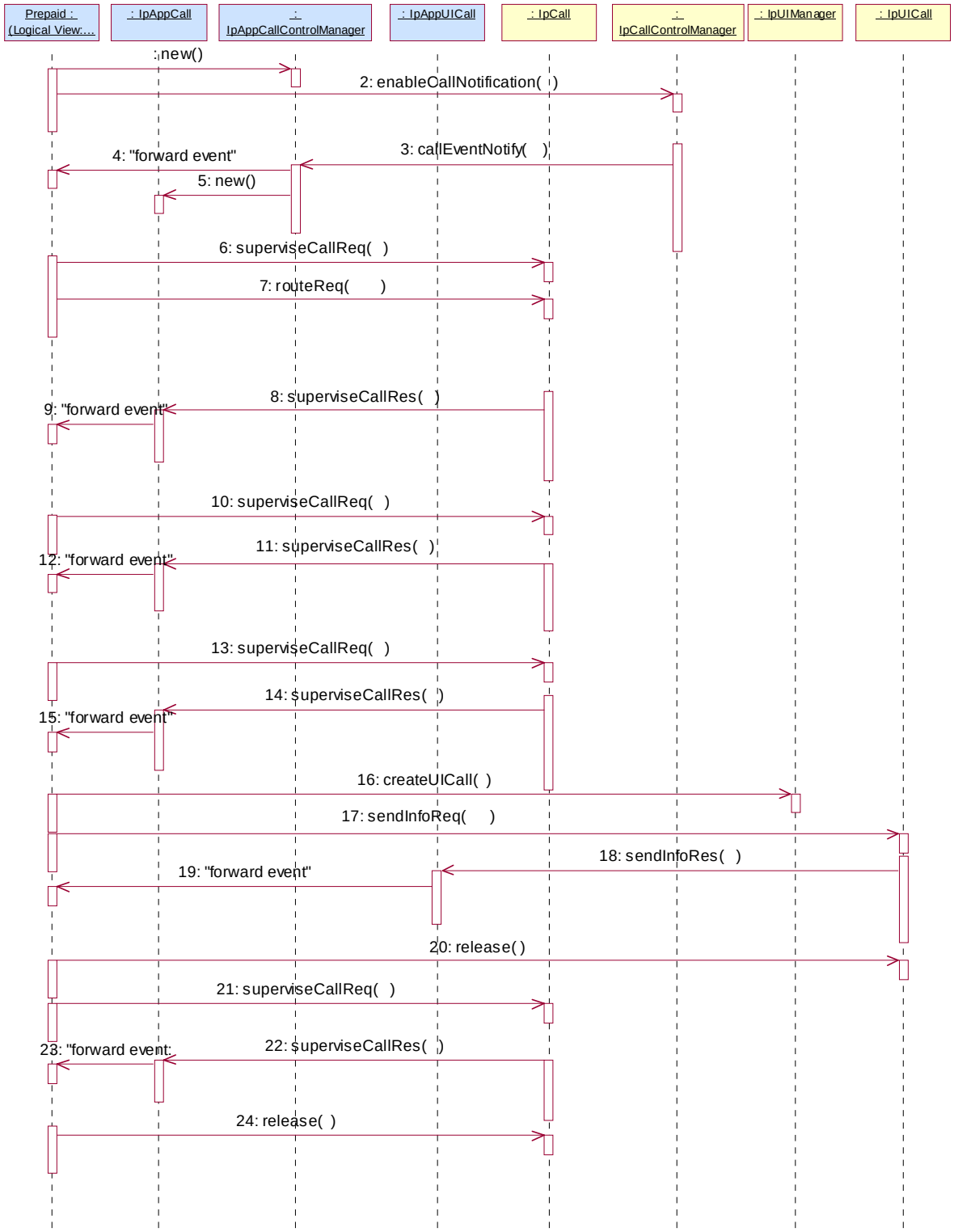
15: When the call is terminated in the network, the application will receive a notification. This notification will always be received when the call is terminated by the network in a normal way, the application does not have to request this event explicitly.

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.

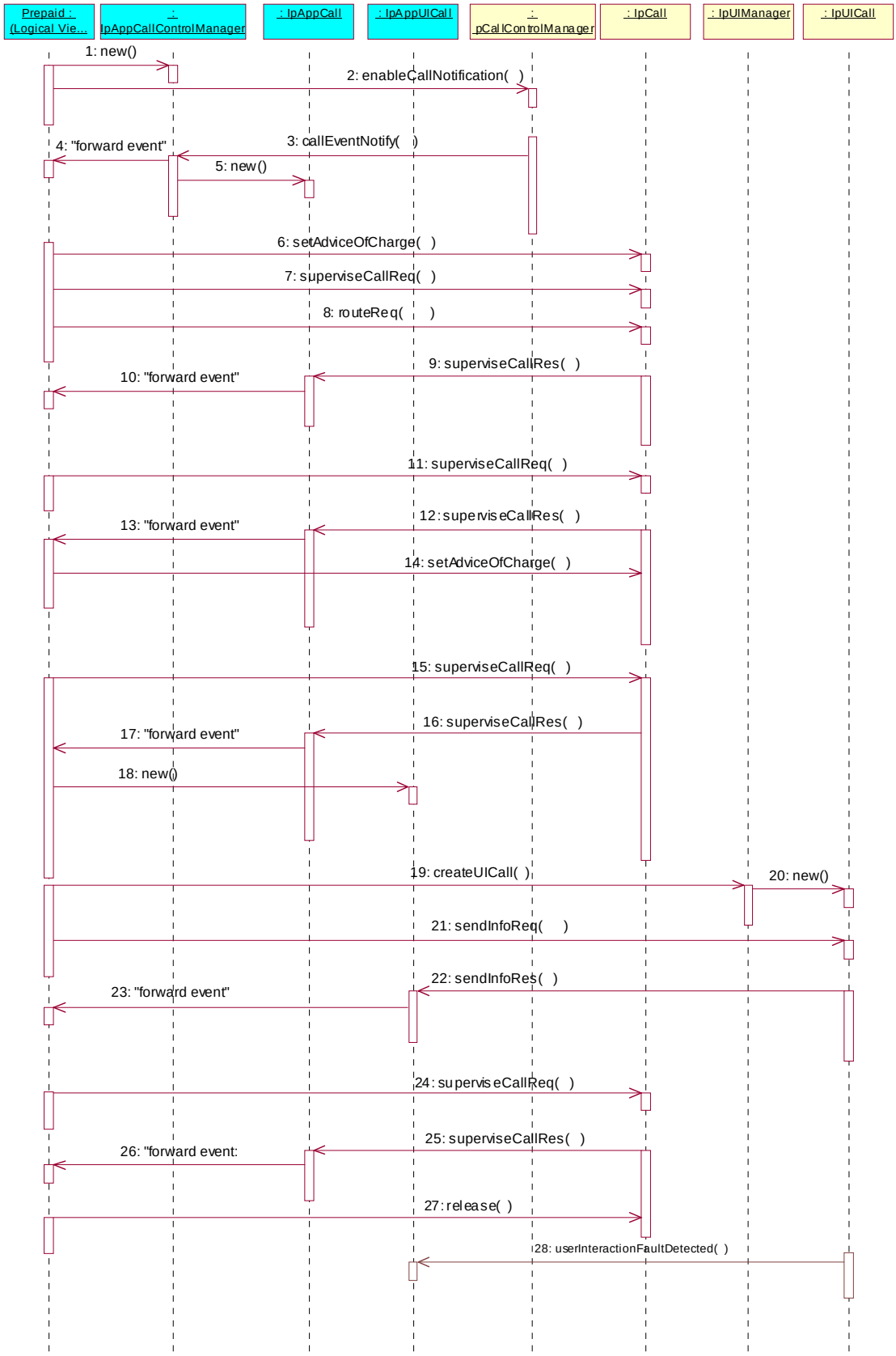
22: The supervision period ends

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.

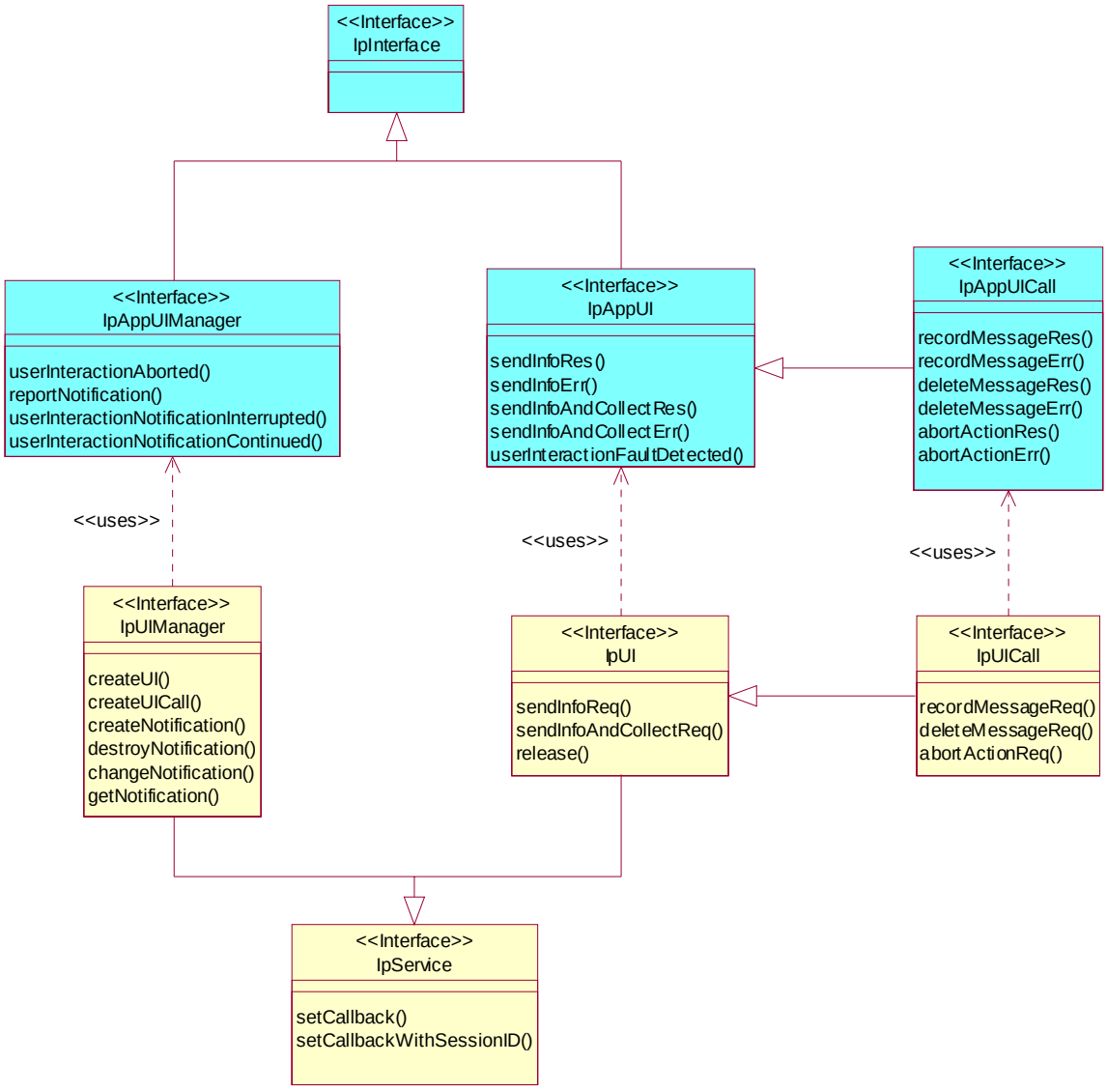


Figure: Generic User Interaction Package Overview

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.

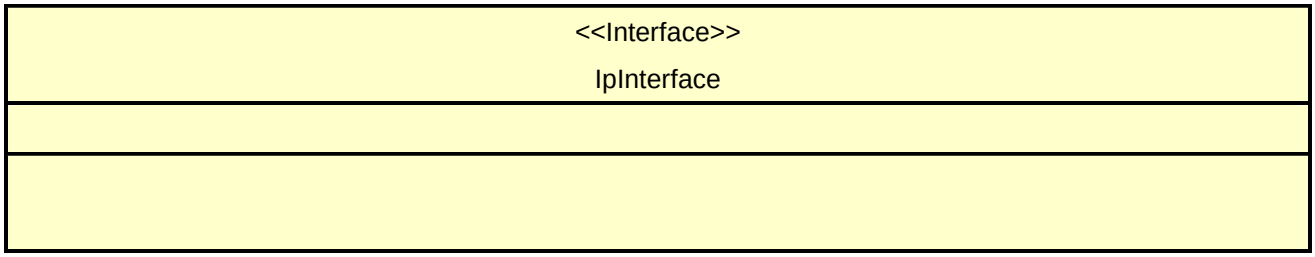
7.1.4 State Model

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.

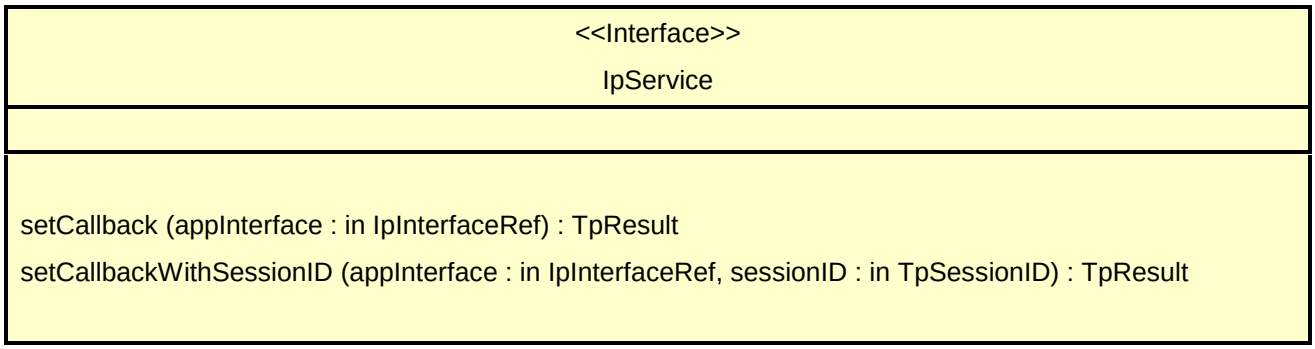
The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

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

TpCommonExceptions

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

TpCommonExceptions

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

TpCommonExceptions,P_INVALID_NETWORK_STATE

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

TpCommonExceptions,P_INVALID_NETWORK_STATE

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_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

TpCommonExceptions,P_INVALID_CRITERIA

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_ASSIGNMENT_ID.

Raises

TpCommonExceptions,P_INVALID_ASSIGNMENT_ID

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

TpCommonExceptions,P_INVALID_ASSIGNMENT_ID,P_INVALID_CRITERIA

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.

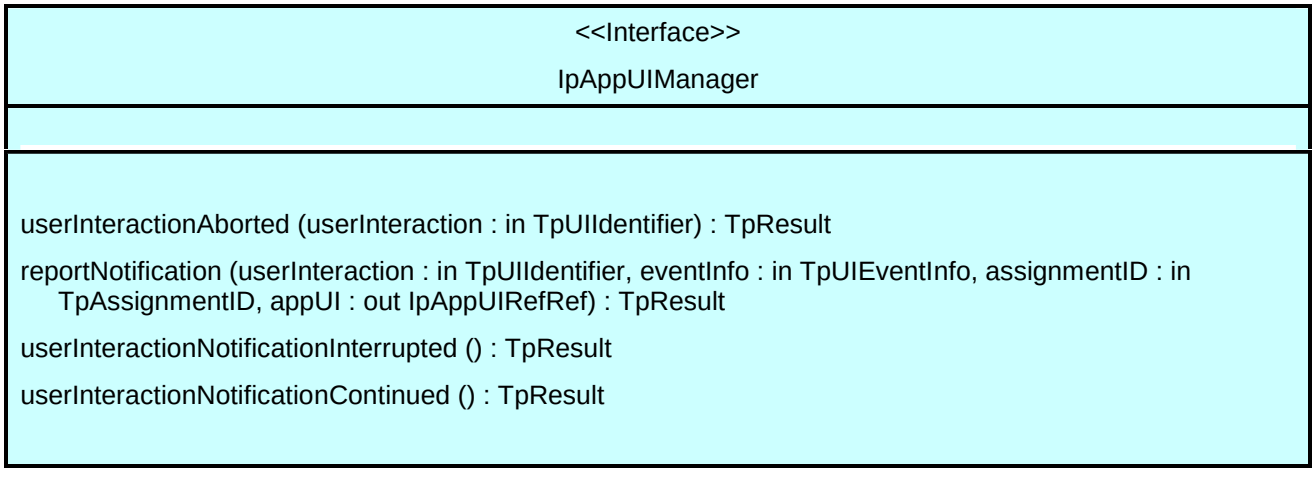
Raises

TpCommonExceptions,P_INVALID_CRITERIA

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.

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.

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

Method

userInteractionNotificationContinued()

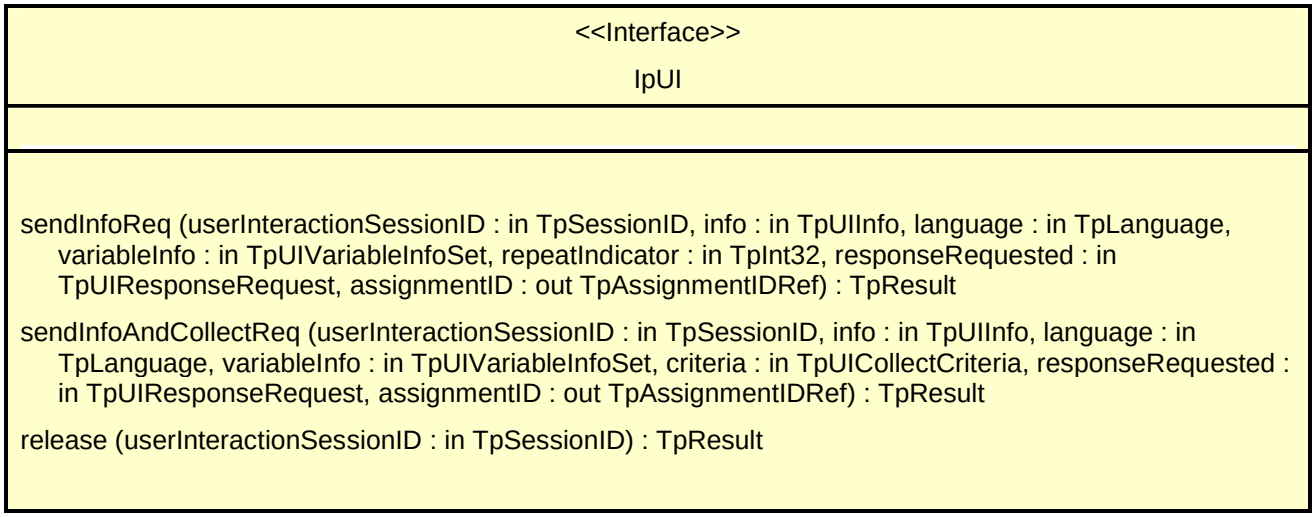
This method indicates to the application that event notifications will again be possible.

Parameters
No Parameters were identified for this method

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

TpCommonExceptions,P_INVALID_SESSION_ID,P_INVALID_NETWORK_STATE,P_ILLEGAL_ID,P_ID_NOT_FOUND

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

TpCommonExceptions,P_INVALID_SESSION_ID,P_INVALID_NETWORK_STATE,P_ILLEGAL_ID,P_ID_NOT_FOUND,P_INVALID_CRITERIA,P_ILLEGAL_RANGE,P_INVALID_COLLECTION_CRITERIA

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

TpCommonExceptions,P_INVALID_SESSION_ID

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.

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.

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.

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.

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.

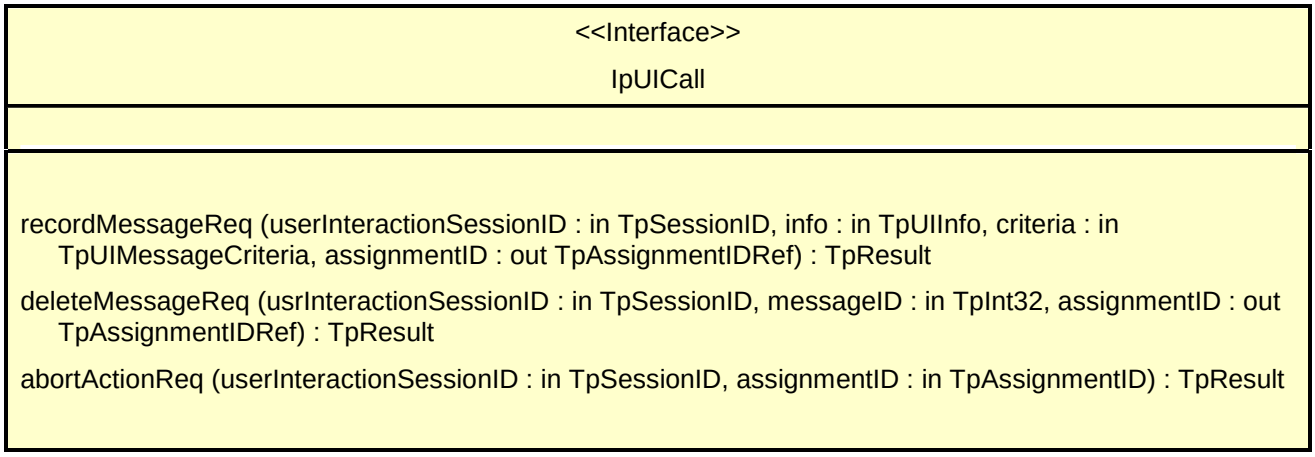
fault : in TpUIFault

Specifies the fault that has been detected.

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

TpCommonExceptions,P_INVALID_SESSION_ID,P_INVALID_NETWORK_STATE,P_ILLEGAL_ID,P_ID_NOT_FOUND,P_INVALID_CRITERIA

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

TpCommonExceptions,P_INVALID_SESSION_ID,P_ILLEGAL_ID,P_ID_NOT_FOUND

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.

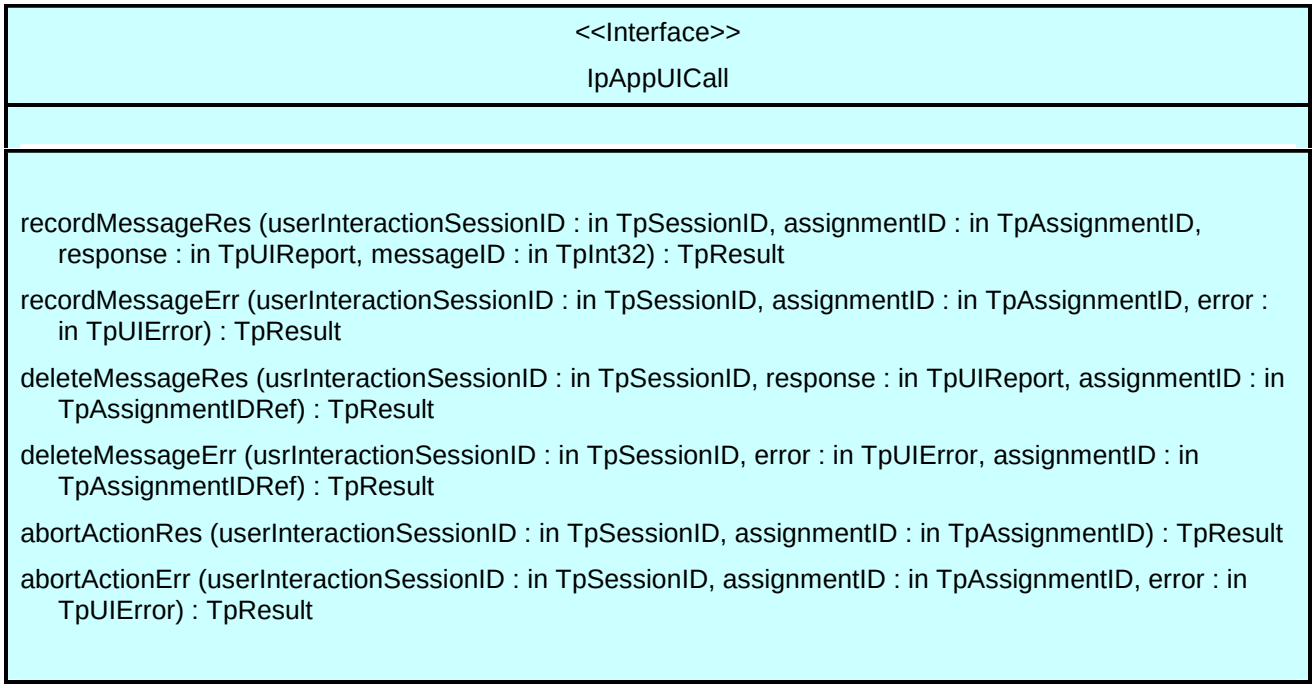
Raises

TpCommonExceptions,P_INVALID_SESSION_ID,P_INVALID_ASSIGNMENT_ID

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.

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.

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.

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.

9 State Transition Diagrams

9.1 State Transition Diagrams for IpUIManager

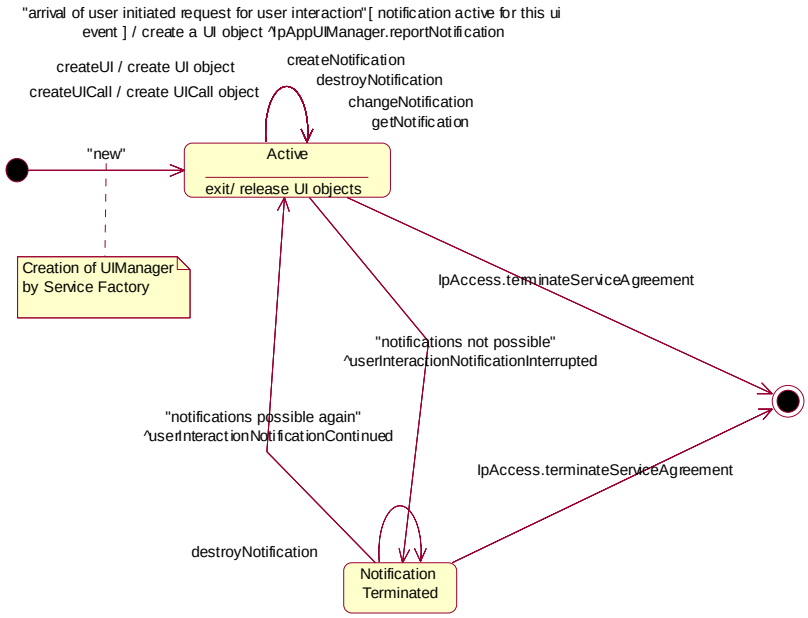


Figure : Application view on the UI Manager

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.

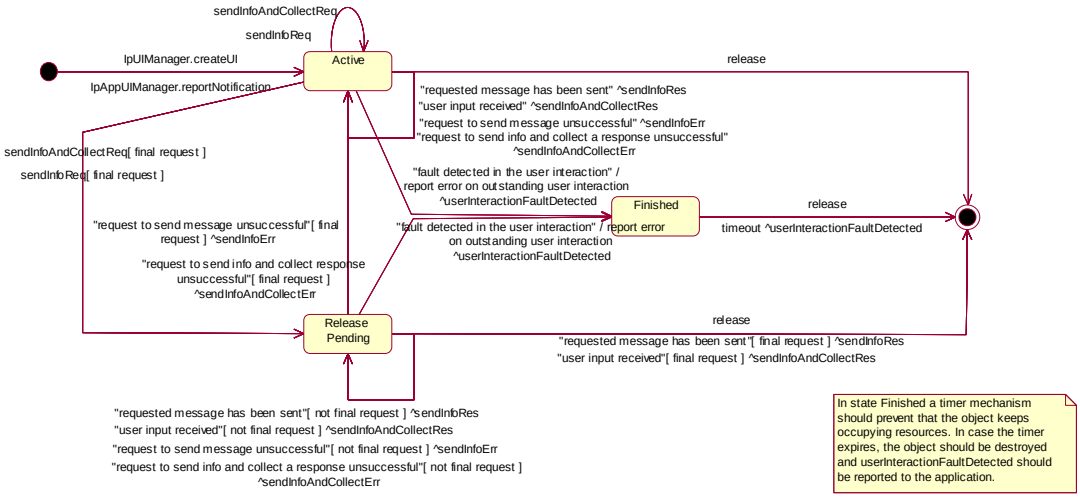


Figure : Application view on the UI object

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.

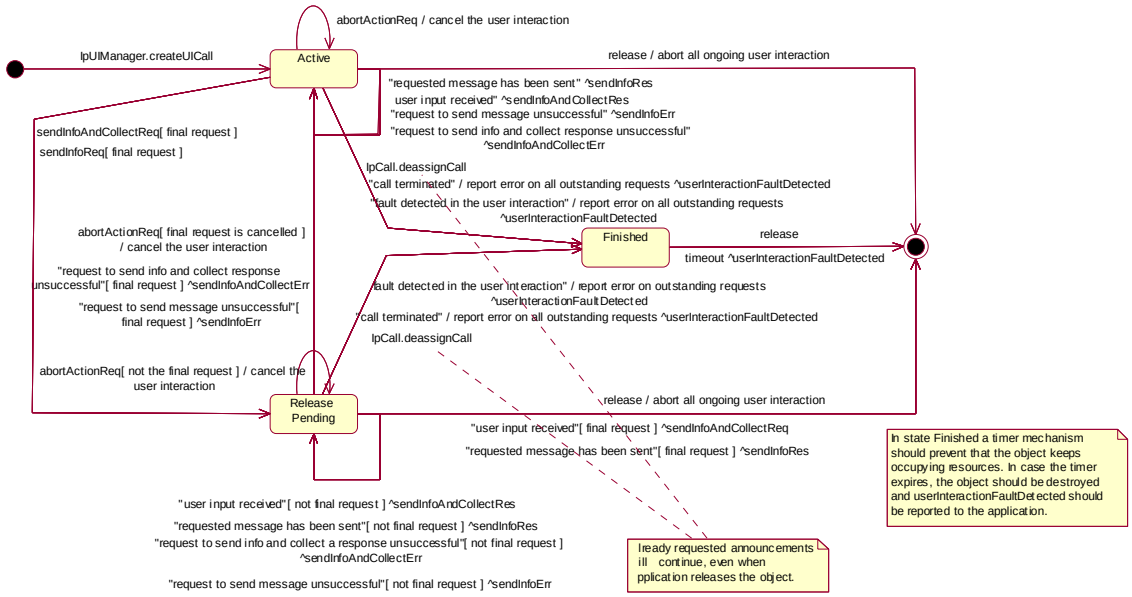


Figure : Application view on the UICall object

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	Structure Element Description
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 shall 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

12 Exception Classes

The following are the list of exception classes which are used in this interface of the API.

Name	Description
P_ILLEGAL_ID	Information id specified is invalid
P_ID_NOT_FOUND	A legal information id is not known to the User Interaction Service
P_ILLEGAL_RANGE	The values for minimum and maximum collection length are out of range.
P_INVALID_COLLECTION_CRITERIA	Invalid collection criteria specified

Each exception class contains the following structure:

Structure Element Name	Structure Element Type	Structure Element Description
extraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name

Annex A (normative): OMG IDL Description of User Interaction SCF

The OMG IDL representation of this interface specification is contained in ~~a~~ text file<u>s</u> (ui<u>_data</u>.idl <u>and ui_interfaces.idl</u> contained in archive 2919805IDL.ZIP) which accompanies the present document.

Annex B (informative): Differences between this draft and 3GPP TS 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 All Interfaces

All methods on IpApp interfaces no longer throw exceptions.

All methods on the other interfaces throw TpCommonExceptions and individual, identified exceptions

B.8~~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.9~~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.109 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.110 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.121 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

Annex C (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
16 Mar 2001	CN_11	NP-010134	047	-	CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158)	3.2.0	4.0.0
1 Jun 2001	CN_12		001		CR 29.198: adding detailed exceptions for each method	4.0.0	4.0.1

```
//Source file: ui_data.idl
//Date:  8 June 2001

#ifndef __UI_DATA_DEFINED
#define __UI_DATA_DEFINED



#include "osa.idl"

module org {

      module csapi {



            module ui {

                  enum TpUIVariablePartType {
                        P_UI_VARIABLE_PART_INT,
                        P_UI_VARIABLE_PART_ADDRESS,
                        P_UI_VARIABLE_PART_TIME,
                        P_UI_VARIABLE_PART_DATE,
                        P_UI_VARIABLE_PART_PRICE
                  };

                  union TpUIVariableInfo switch(TpUIVariablePartType) {
                        case P_UI_VARIABLE_PART_INT: TpInt32
VariablePartInteger;
                        case P_UI_VARIABLE_PART_ADDRESS: TpString
VariablePartAddress;
                        case P_UI_VARIABLE_PART_TIME: TpTime VariablePartTime;
                        case P_UI_VARIABLE_PART_DATE: TpDate VariablePartDate;
                        case P_UI_VARIABLE_PART_PRICE: TpPrice
VariablePartPrice;
                  };

                  typedef sequence <TpUIVariableInfo> TpUIVariableInfoSet;

                  typedef TpInt32 TpUIResponseRequest;

                  enum TpUIReport {
                        P_UI_REPORT_UNDEFINED,
                        P_UI_REPORT_INFO_SENT,
                        P_UI_REPORT_INFO_COLLECTED,
                        P_UI_REPORT_NO_INPUT,
                        P_UI_REPORT_TIMEOUT,
                        P_UI_REPORT_MESSAGE_STORED,
                        P_UI_REPORT_MESSAGE_NOT_STORED,
                        P_UI_REPORT_MESSAGE_DELETED,
                        P_UI_REPORT_MESSAGE_NOT_DELETED
                  };

                  struct TpUIMessageCriteria {
                        TpString EndSequence;
                        TpDuration MaxMessageTime;
                        TpInt32 MaxMessageSize;
                  };

                  enum TpUIInfoType {
                        P_UI_INFO_ID,
```

```
      P_UI_INFO_DATA,
      P_UI_INFO_ADDRESS
};

union TpUIInfo switch(TpUIInfoType) {
      case P_UI_INFO_ID: TpInt32 InfoId;
      case P_UI_INFO_DATA: TpString InfoData;
      case P_UI_INFO_ADDRESS: TpURL InfoAddress;
};

enum TpUIFault {
      P_UI_FAULT_UNDEFINED,
      P_UI_CALL_ENDED
};

enum TpUIEventInfoDataType {
      P_UI_EVENT_DATA_TYPE_UNDEFINED,
      P_UI_EVENT_DATA_TYPE_UNSPECIFIED,
      P_UI_EVENT_DATA_TYPE_TEXT,
      P_UI_EVENT_DATA_TYPE_USSD_DATA
};

struct TpUIEventInfo {
      TpAddress OriginatingAddress;
      TpAddress DestinationAddress;
      TpString ServiceCode;
      TpUIEventInfoDataType DataTypeIndication;
      TpString DataString;
};

struct TpUIEventCriteria {
      TpAddressRange OriginatingAddress;
      TpAddressRange DestinationAddress;
      TpString ServiceCode;
};

enum TpUIError {
      P_UI_ERROR_UNDEFINED,
      P_UI_ERROR_ILLEGAL_INFO,
      P_UI_ERROR_ID_NOT_FOUND,
      P_UI_ERROR_RESOURCE_UNAVAILABLE,
      P_UI_ERROR_ILLEGAL_RANGE,
      P_UI_ERROR_IMPROPER_USER_RESPONSE,
      P_UI_ERROR_ABANDON,
      P_UI_ERROR_NO_OPERATION_ACTIVE,
      P_UI_ERROR_NO_SPACE_AVAILABLE,
      P_UI_ERROR_RESOURCE_TIMEOUT
};

struct TpUICollectCriteria {
      TpInt32 MinLength;
      TpInt32 MaxLength;
      TpString EndSequence;
      TpDuration StartTimeout;
      TpDuration InterCharTimeout;
};

const TpInt32 P_UI_RESPONSE_REQUIRED = 1;
const TpInt32 P_UI_LAST_ANNOUNCEMENT_IN_A_ROW = 2;
const TpInt32 P_UI_FINAL_REQUEST = 4;
```

```
                  struct TpUIEventCriteriaResult {

                        TpUIEventCriteria EventCriteria;

                        TpInt32 AssignmentID;
                  };

                  typedef sequence <TpUIEventCriteriaResult>
TpUIEventCriteriaResultSet;

                  exception P_ID_NOT_FOUND {
                        TpString extraInformation;
                  };

                  exception P_ILLEGAL_ID {
                        TpString extraInformation;
                  };

                  exception P_ILLEGAL_RANGE {
                        TpString extraInformation;
                  };

                  exception P_INVALID_COLLECTION_CRITERIA {
                        TpString extraInformation;
                  };



            };

      };

};

#endif
```

```
//Source file: ui_interfaces.idl
//Date:  8 June 2001


#ifndef __UI_INTERFACES_DEFINED
#define __UI_INTERFACES_DEFINED



#include "osa.idl"
#include "ui_data.idl"
#include "gcc_interfaces.idl"
#include "mpcc_interfaces.idl"

module org {

      module csapi {



            module ui {

                  interface IpUI;
                  interface IpUICall;
                  interface IpAppUI;
                  interface IpAppUICall;

                  struct TpUIIdentifier {
                        IpUI UIRef;
                        TpSessionID UserInteractionSessionID;
                  };

                  struct TpUICallIdentifier {
                        IpUICall UICallRef;
                        TpSessionID UserInteractionSessionID;
                  };

                  enum TpUITargetObjectType {
                        P_UI_TARGET_OBJECT_CALL,
                        P_UI_TARGET_OBJECT_MULTI_PARTY_CALL,
                        P_UI_TARGET_OBJECT_CALL_LEG
                  };

                  union TpUITargetObject switch(TpUITargetObjectType) {
                        case P_UI_TARGET_OBJECT_CALL: cc::gccs::TpCallIdentifier
Call;
                        case P_UI_TARGET_OBJECT_MULTI_PARTY_CALL:
cc::mpccs::TpMultiPartyCallIdentifier MultiPartyCall;
                        case P_UI_TARGET_OBJECT_CALL_LEG:
cc::mpccs::TpCallLegIdentifier CallLeg;
                  };



                  interface IpAppUIManager : IpInterface {

                        void userInteractionAborted (
                              in TpUIIdentifier userInteraction
                              );
```

```
                        void reportNotification (
                              in TpUIIdentifier userInteraction,
                              in TpUIEventInfo eventInfo,
                              in TpAssignmentID assignmentID,
                              out IpAppUI appUI
                              );


                        void userInteractionNotificationInterrupted ();


                        void userInteractionNotificationContinued ();

                  };



                  interface IpUIManager : IpService {

                        void createUI (
                              in IpAppUI appUI,
                              in TpAddress userAddress,
                              out TpUIIdentifier userInteraction
                              )
                              raises
(TpCommonExceptions,P_INVALID_NETWORK_STATE);


                        void createUICall (
                              in IpAppUICall appUI,
                              in TpUITargetObject uiTargetObject,
                              out TpUICallIdentifier userInteraction
                              )
                              raises
(TpCommonExceptions,P_INVALID_NETWORK_STATE);


                        void createNotification (
                              in IpAppUIManager appUIManager,
                              in TpUIEventCriteria eventCriteria,
                              out TpAssignmentID assignmentID
                              )
                              raises (TpCommonExceptions,P_INVALID_CRITERIA);


                        void destroyNotification (
                              in TpAssignmentID assignmentID
                              )
                              raises
(TpCommonExceptions,P_INVALID_ASSIGNMENT_ID);


                        void changeNotification (
                              in TpAssignmentID assignmentID,
                              in TpUIEventCriteria evenCriteria
                              )
                              raises
(TpCommonExceptions,P_INVALID_ASSIGNMENT_ID,P_INVALID_CRITERIA);


                        void getNotification (
```

```
            out TpUIEventCriteriaResultSet eventCriteria
            )
            raises (TpCommonExceptions,P_INVALID_CRITERIA);

};


interface IpAppUI : IpInterface {

      void sendInfoRes (
            in TpSessionID userInteractionSessionID,
            in TpAssignmentID assignmentID,
            in TpUIReport response
            );


      void sendInfoErr (
            in TpSessionID userInteractionSessionID,
            in TpAssignmentID assignmentID,
            in TpUIError error
            );


      void sendInfoAndCollectRes (
            in TpSessionID userInteractionSessionID,
            in TpAssignmentID assignmentID,
            in TpUIReport response,
            in TpString collectedInfo
            );


      void sendInfoAndCollectErr (
            in TpSessionID userInteractionSessionID,
            in TpAssignmentID assignmentID,
            in TpUIError error
            );


      void userInteractionFaultDetected (
            in TpSessionID userInteractionSessionID,
            in TpUIFault fault
            );

};


interface IpUI : IpService {

      void sendInfoReq (
            in TpSessionID userInteractionSessionID,
            in TpUIInfo info,
            in TpLanguage language,
            in TpUIVariableInfoSet variableInfo,
            in TpInt32 repeatIndicator,
            in TpUIResponseRequest responseRequested,
            out TpAssignmentID assignmentID
            )
```

```
                              raises
(TpCommonExceptions,P_INVALID_SESSION_ID,P_INVALID_NETWORK_STATE,P_ILLEGAL_ID,P_
ID_NOT_FOUND);


                        void sendInfoAndCollectReq (
                              in TpSessionID userInteractionSessionID,
                              in TpUIInfo info,
                              in TpLanguage language,
                              in TpUIVariableInfoSet variableInfo,
                              in TpUICollectCriteria criteria,
                              in TpUIResponseRequest responseRequested,
                              out TpAssignmentID assignmentID
                              )
                              raises
(TpCommonExceptions,P_INVALID_SESSION_ID,P_INVALID_NETWORK_STATE,P_ILLEGAL_ID,P_
ID_NOT_FOUND,P_INVALID_CRITERIA,P_ILLEGAL_RANGE,P_INVALID_COLLECTION_CRITERIA);


                        void release (
                              in TpSessionID userInteractionSessionID
                              )
                              raises (TpCommonExceptions,P_INVALID_SESSION_ID);

                  };



                  interface IpAppUICall : IpAppUI {

                        void recordMessageRes (
                              in TpSessionID userInteractionSessionID,
                              in TpAssignmentID assignmentID,
                              in TpUIReport response,
                              in TpInt32 messageID
                              );


                        void recordMessageErr (
                              in TpSessionID userInteractionSessionID,
                              in TpAssignmentID assignmentID,
                              in TpUIError error
                              );


                        void deleteMessageRes (
                              in TpSessionID usrInteractionSessionID,
                              in TpUIReport response,
                              in TpAssignmentID assignmentID
                              );


                        void deleteMessageErr (
                              in TpSessionID usrInteractionSessionID,
                              in TpUIError error,
                              in TpAssignmentID assignmentID
                              );


                        void abortActionRes (
                              in TpSessionID userInteractionSessionID,
```

```
                              in TpAssignmentID assignmentID
                              );


                        void abortActionErr (
                              in TpSessionID userInteractionSessionID,
                              in TpAssignmentID assignmentID,
                              in TpUIError error
                              );

                  };



                  interface IpUICall : IpUI {

                        void recordMessageReq (
                              in TpSessionID userInteractionSessionID,
                              in TpUIInfo info,
                              in TpUIMessageCriteria criteria,
                              out TpAssignmentID assignmentID
                              )
                              raises
(TpCommonExceptions,P_INVALID_SESSION_ID,P_INVALID_NETWORK_STATE,P_ILLEGAL_ID,P_
ID_NOT_FOUND,P_INVALID_CRITERIA);


                        void deleteMessageReq (
                              in TpSessionID usrInteractionSessionID,
                              in TpInt32 messageID,
                              out TpAssignmentID assignmentID
                              )
                              raises
(TpCommonExceptions,P_INVALID_SESSION_ID,P_ILLEGAL_ID,P_ID_NOT_FOUND);


                        void abortActionReq (
                              in TpSessionID userInteractionSessionID,
                              in TpAssignmentID assignmentID
                              )
                              raises
(TpCommonExceptions,P_INVALID_SESSION_ID,P_INVALID_ASSIGNMENT_ID);

                  };

            };

      };

};

#endif
```

CR-Form-v3

CHANGE REQUEST

⌘ **29.198-6** **CR** **001** ⌘ rev **-** ⌘ Current version: **4.0.0** ⌘

*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:	⌘	Corrections to OSA API Rel4
Source:	⌘	CN5
Work item code:	⌘	OSA1
Date:	⌘	07/06/2001
Category:	⌘	**F**
Release:	⌘	Rel4

Use one of the following categories:
F *(essential correction)*
A *(corresponds to a correction in an earlier release)*
B *(Addition of feature),*
C *(Functional modification of feature)*
D *(Editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
REL-4	*(Release 4)*
REL-5	*(Release 5)*

Reason for change:	⌘	Exception handling mechanism in 29.198 requires correction to enable it to be correctly used, without ambiguity
Summary of change:	⌘	Replace TpGeneralException, TpUIException with detailed exception classes which can be thrown for each method
Consequences if not approved:	⌘	29.198-6 will be ambiguous and difficult to implement correctly - inter-working might be jeopardised

Clauses affected:	⌘				
Other specs affected:	⌘	**X**	Other core specifications	⌘	All other parts of 29.198 except part 1 have similar changes
		☐	Test specifications		
		☐	O&M Specifications		
Other comments:	⌘				

3GPP TS 29.198-6 V.4.0.~~0~~ 1 (2001-~~03~~06)

Technical Specification

3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API); Part 6: Mobility (Release 4)

Keywords
UMTS, 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.

Introduction

The present document is part 2 of a multi-part TS covering the 3rd Generation Partnership Project: Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API), as identified below. The **API specification** (3GPP TS 29.198) is structured in the following Parts:

Part 1:	Overview	
Part 2:	**Common Data Definitions**	
Part 3:	Framework	
Part 4:	Call Control SCF	
Part 5:	User Interaction SCF	
Part 6:	Mobility SCF	
Part 7:	Terminal Capabilities SCF	
Part 8:	Data Session Control SCF	
Part 9:	Generic Messaging SCF	(not part of 3GPP Release 4)
Part 10:	Connectivity Manager SCF	(not part of 3GPP Release 4)
Part 11:	Account Management SCF	
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.

OSA API specifications 29.198-family		OSA API Mapping - 29.998-family	
29.198-1	**Part 1: Overview**	**29.998-1**	**Part 1: Overview**
29.198-2	**Part 2: Common Data Definitions**	29.998-2	Not Applicable
29.198-3	**Part 3: Framework**	29.998-3	Not Applicable
29.198-4	**Part 4: Call Control SCF**	**29.998-4-1**	**Subpart 1: Generic Call Control – CAP mapping**
		29.998-4-2	
29.198-5	**Part 5: User Interaction SCF**	**29.998-5-1**	**Subpart 1: User Interaction – CAP mapping**
		29.998-5-2	
		29.998-5-3	
		29.998-5-4	**Subpart 4: User Interaction – SMS mapping**
29.198-6	**Part 6: Mobility SCF**	**29.998-6**	**User Status and User Location – MAP mapping**
29.198-7	**Part 7: Terminal Capabilities SCF**	29.998-7	Not Applicable
29.198-8	**Part 8: Data Session Control SCF**	**29.998-8**	**Data Session Control – CAP mapping**
29.198-9	Part 9: Generic Messaging SCF	29.998-9	Not Applicable
29.198-10	Part 10: Connectivity Manager SCF	29.998-10	Not Applicable
29.198-11	**Part 11: Account Management SCF**	29.998-11	Not Applicable
29.198-12	**Part 12: Charging SCF**	29.998-12	Not Applicable

1 Scope

The present document is Part 6 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 APIs. The concepts and the functional architecture for the OSA are contained in 3GPP TS 23.127 [3]. The requirements for OSA are contained in 3GPP TS 22.127 [2].

The present 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, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document.*

[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] 3GPP TS 29.002: "".

3 Definitions and abbreviations

3.1 Definitions

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

3.2 Abbreviations

For the purposes of the present document, the abbreviations given 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:

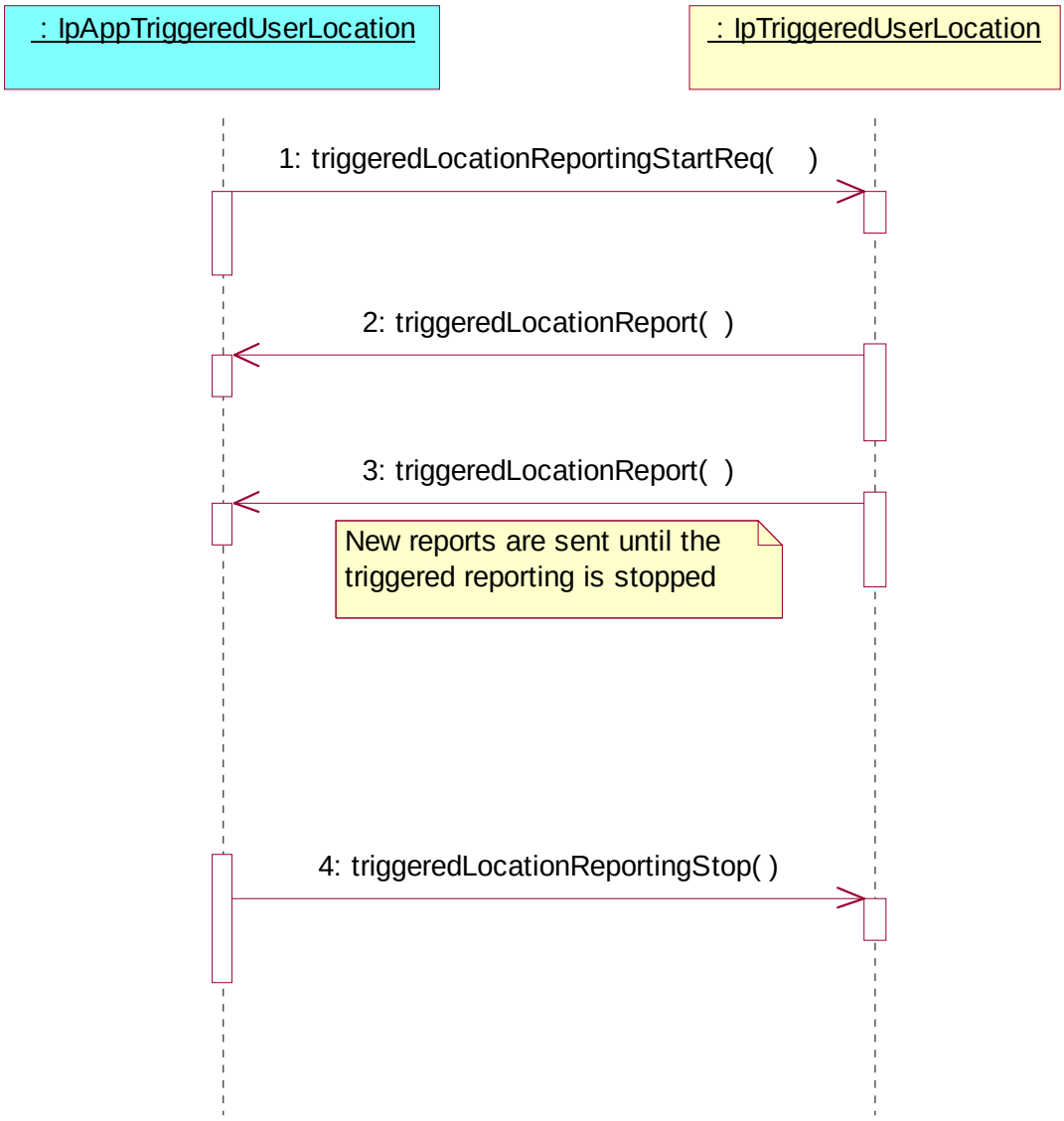
- The Sequence diagrams give the reader a practical idea of how each of the ~~service capability feature~~SCF is implemented.
- The Class relationships section show how each of the interfaces applicable to the SCF, relate to one another
- The Interface specification section describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the transition between states in the SCF. The states and transitions are well-defined; either methods specified in the Interface specification or events occurring in the underlying networks cause state transitions.~~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.

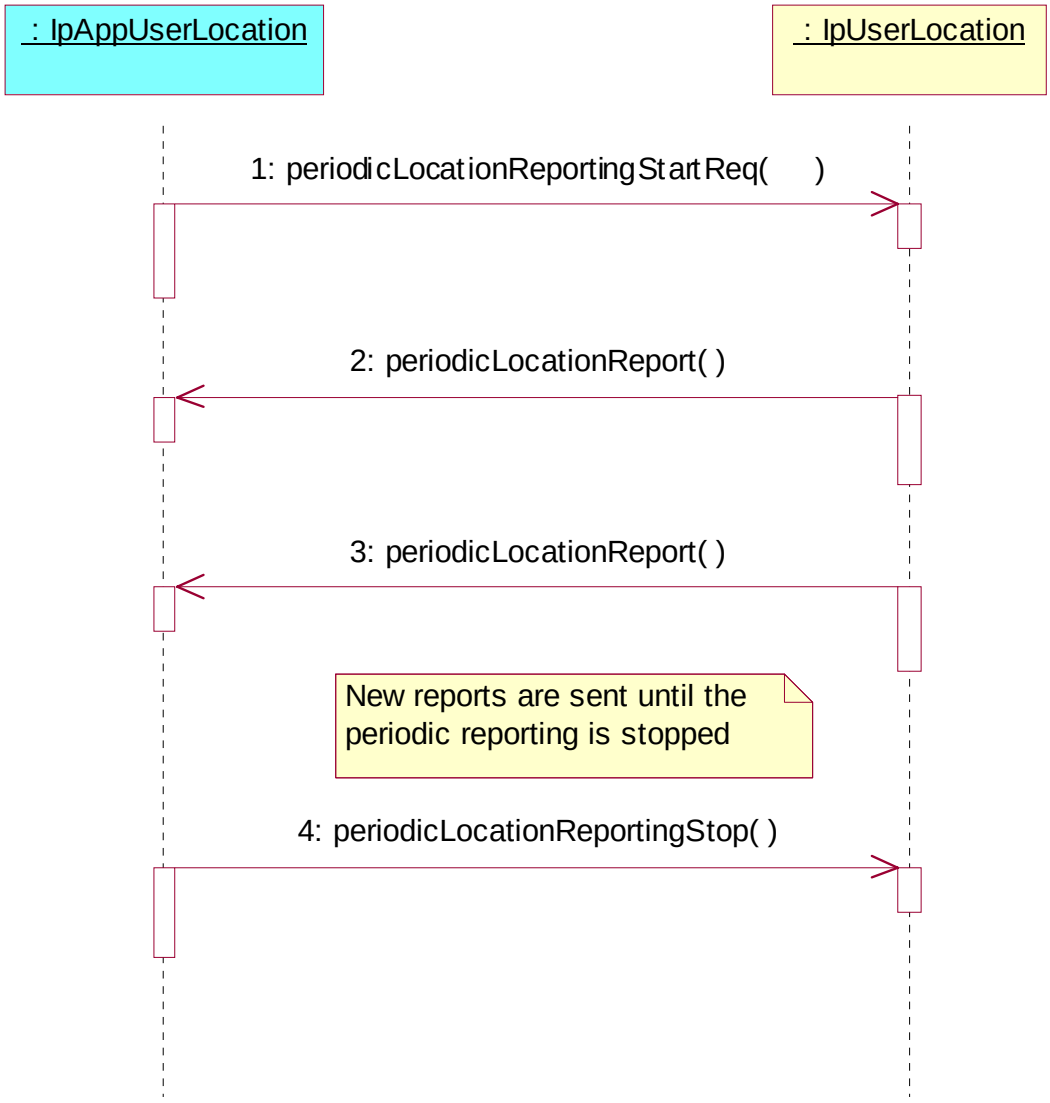
2: When the trigger condition is fulfilled then this message passes the location of the affected user to its callback object.

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.

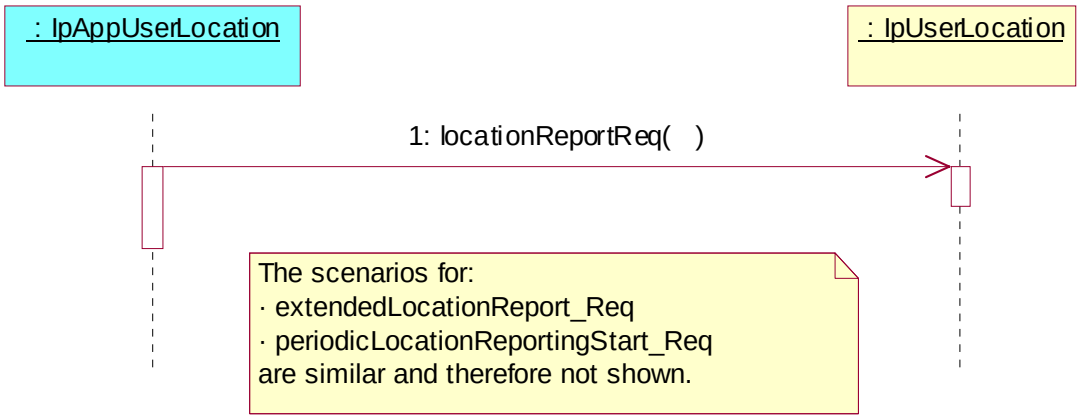
5.1.3 User Location Interrogation - Parameter Error

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.

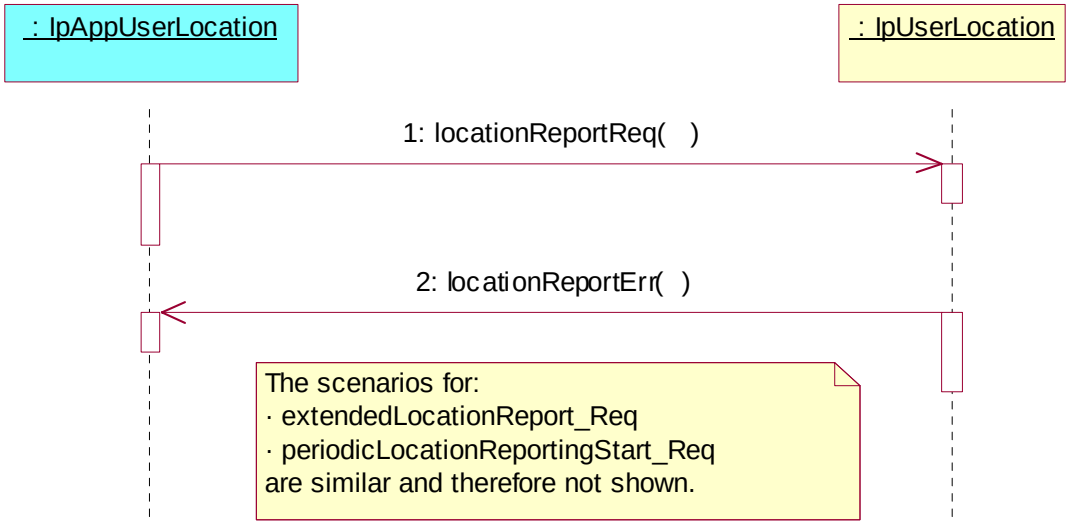
5.1.4 User Location Interrogation - Network Error

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.

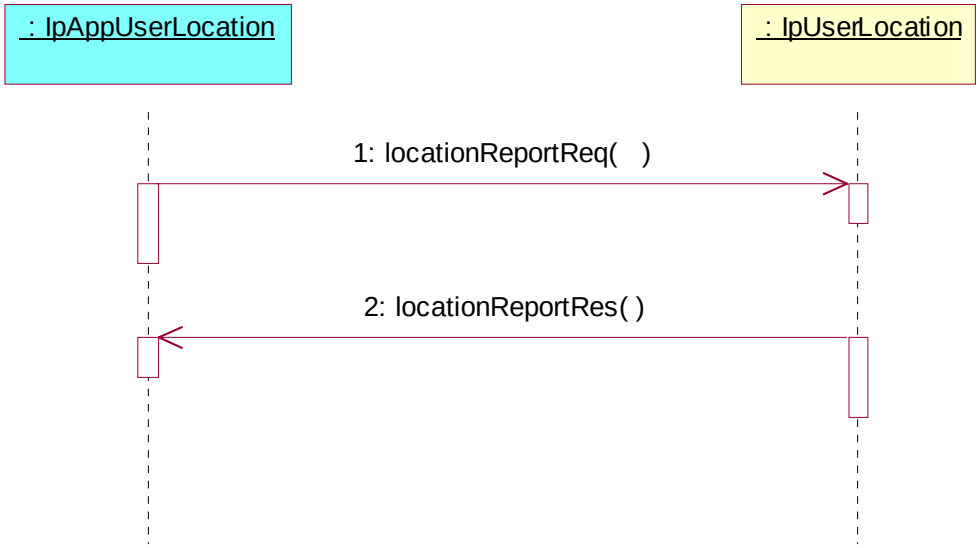


1: This message is used to request the location of one or several users.

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.



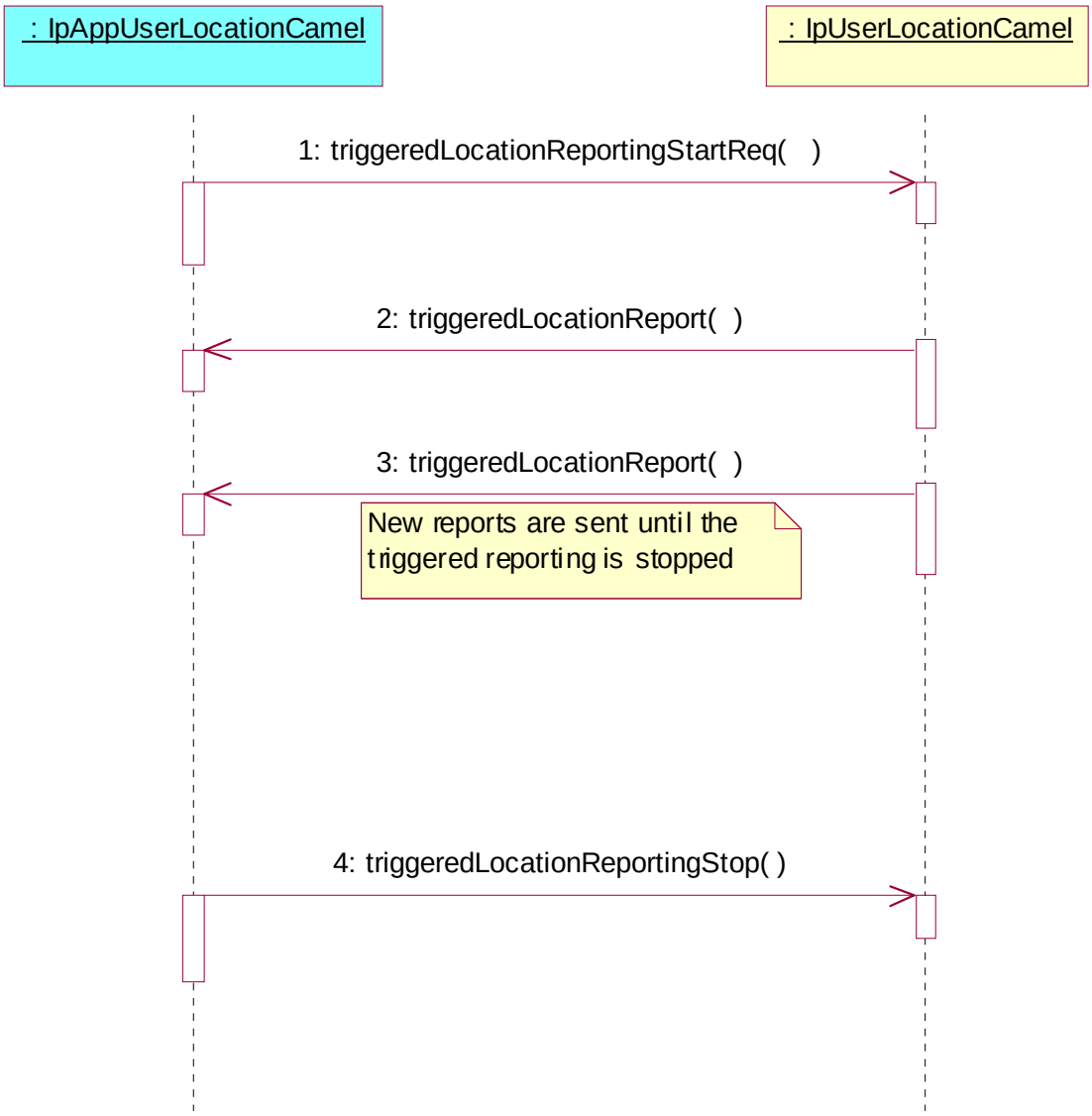
1: This message is used to request the location of one or several users.

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.

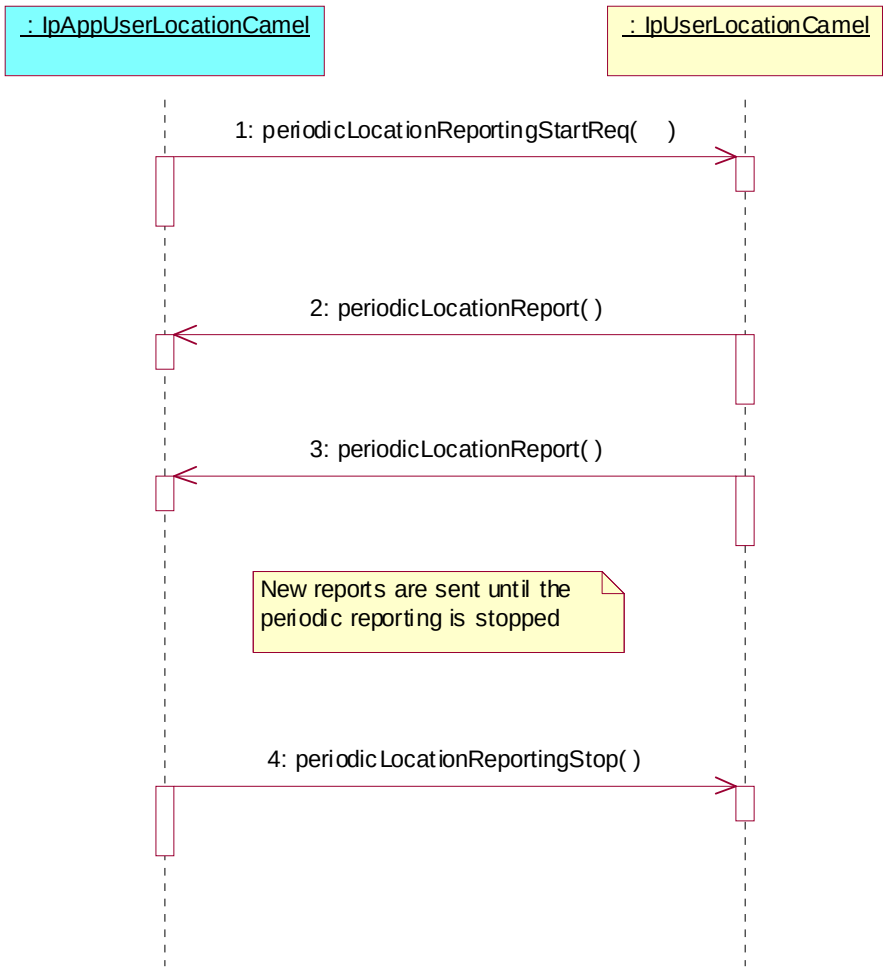
2: When the trigger condition is fulfilled then this message passes the location of the affected user to its callback object.

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).

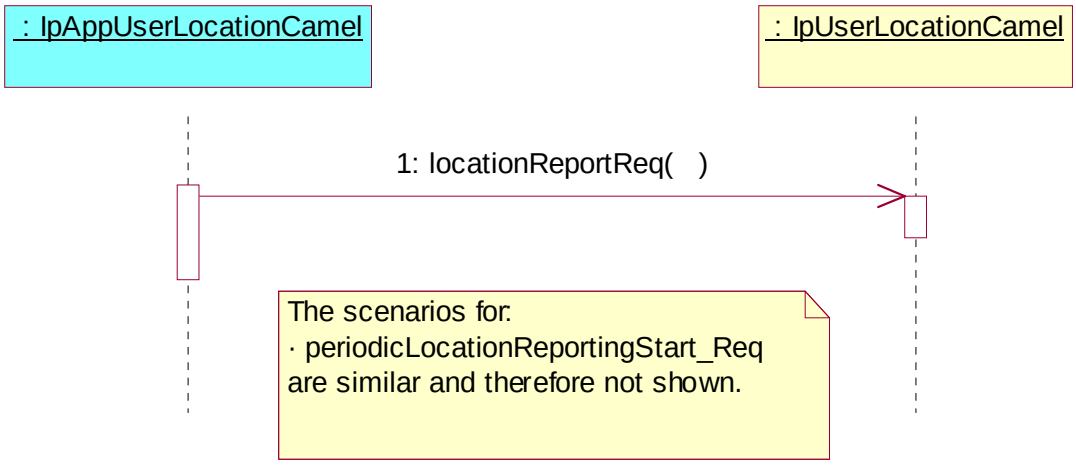
4: This message is used to stop periodic location reporting.

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.



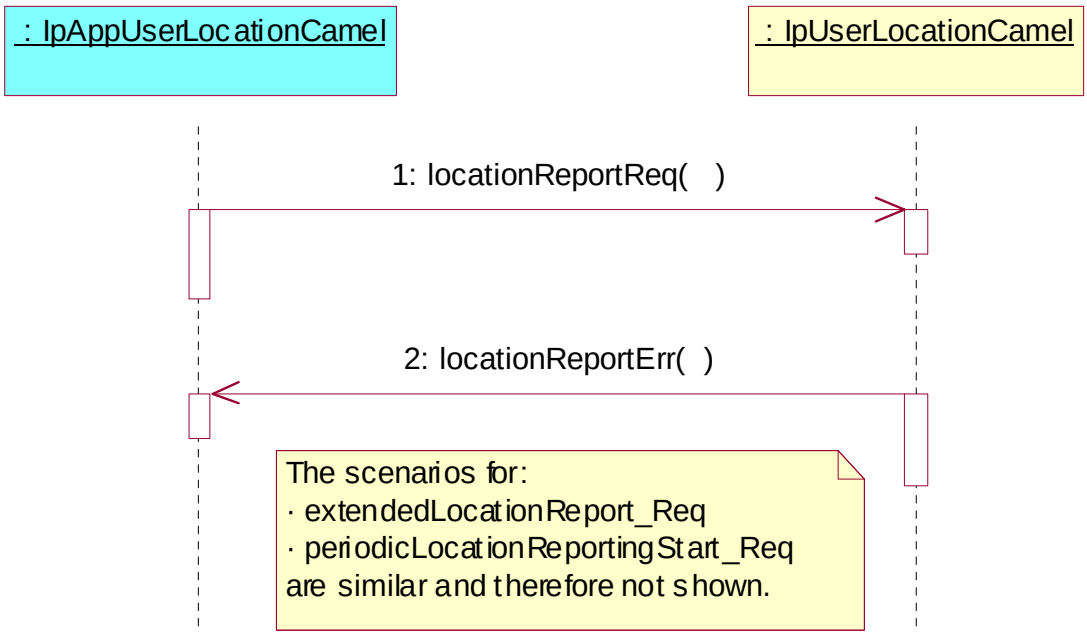
1: This message is used to request the location of one or several users, but the service returns an error and the execution of the request is aborted.

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.

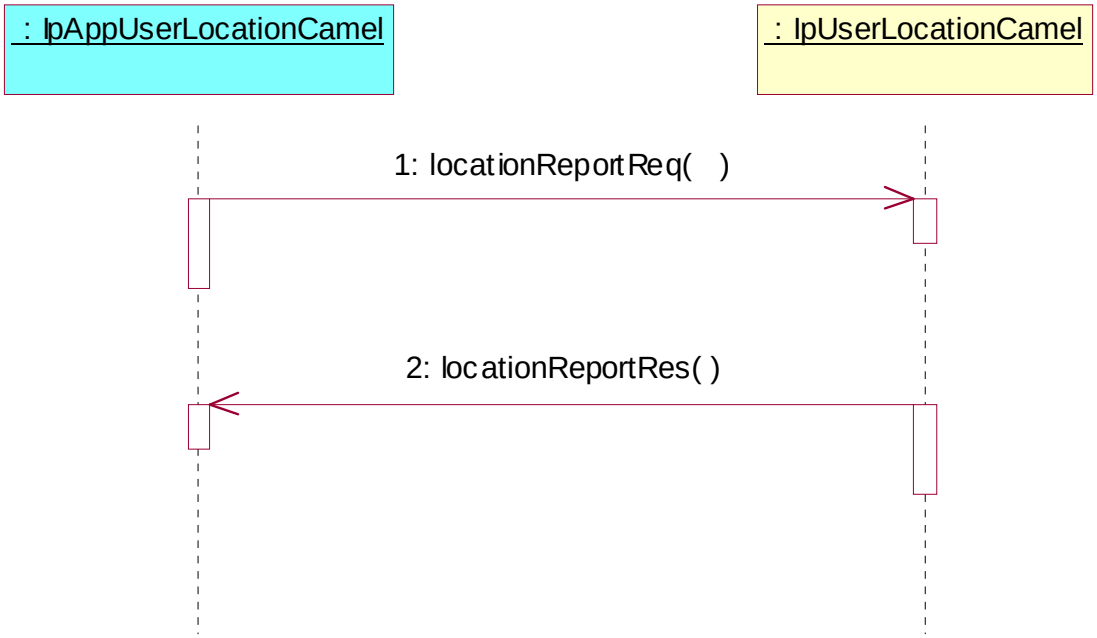


1: This message is used to request the location of one or several users.

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.



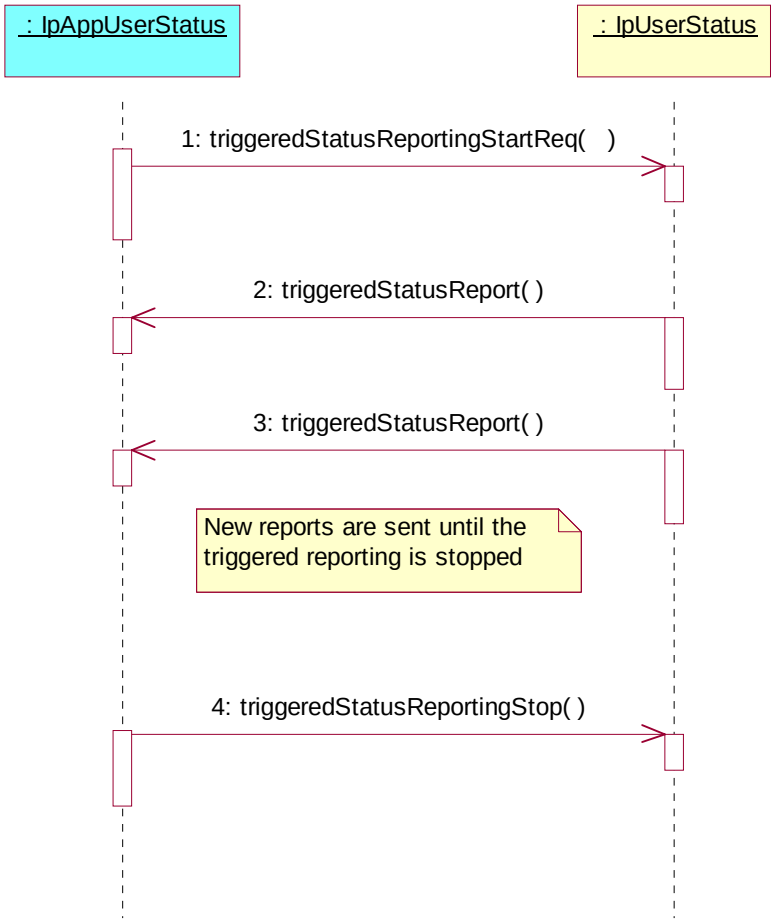
1: This message is used to request the location of one or several users.

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.

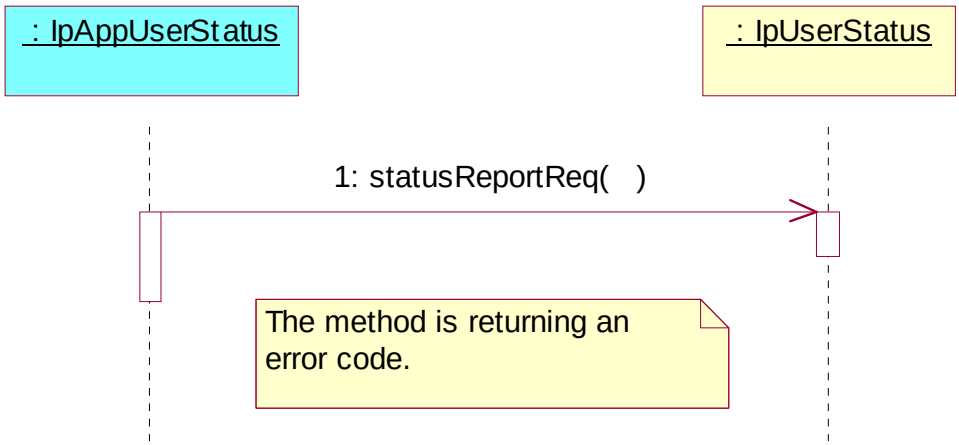
2: When a user's status changes, this message passes the status to its callback object.

3: This is repeated until the application stops triggered status reporting (see next message).

4: This message is used to stop triggered status reporting.

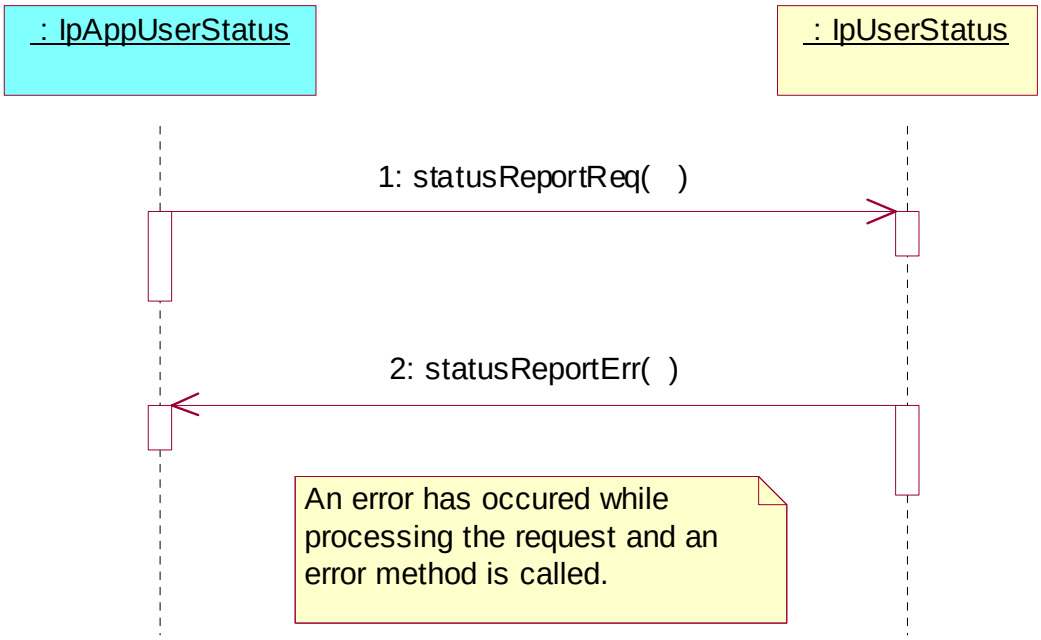
5.3.2 Interactive Request Parameter Error

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.



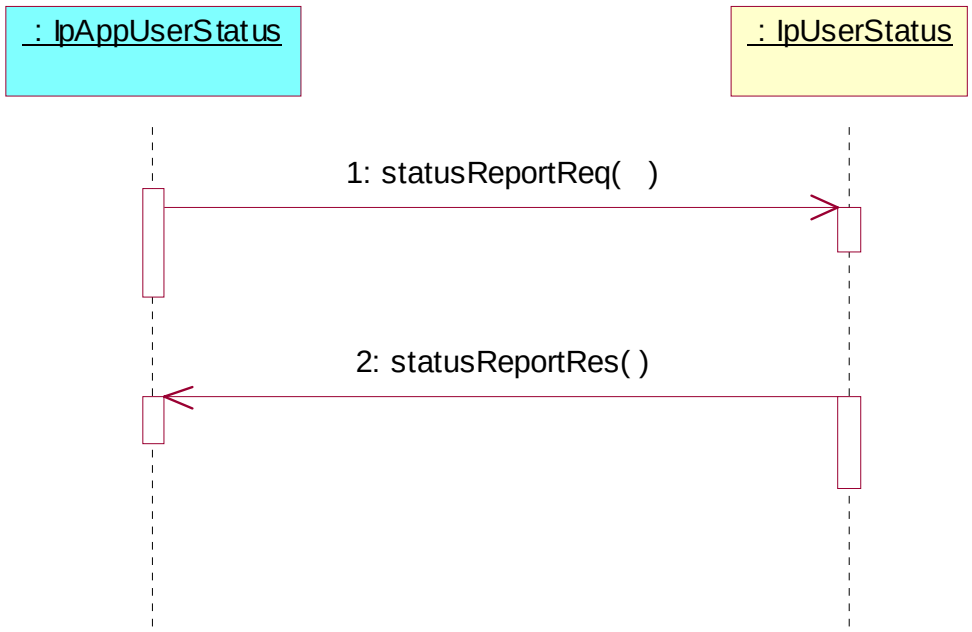
5.3.3 Interactive Request Network Error

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.

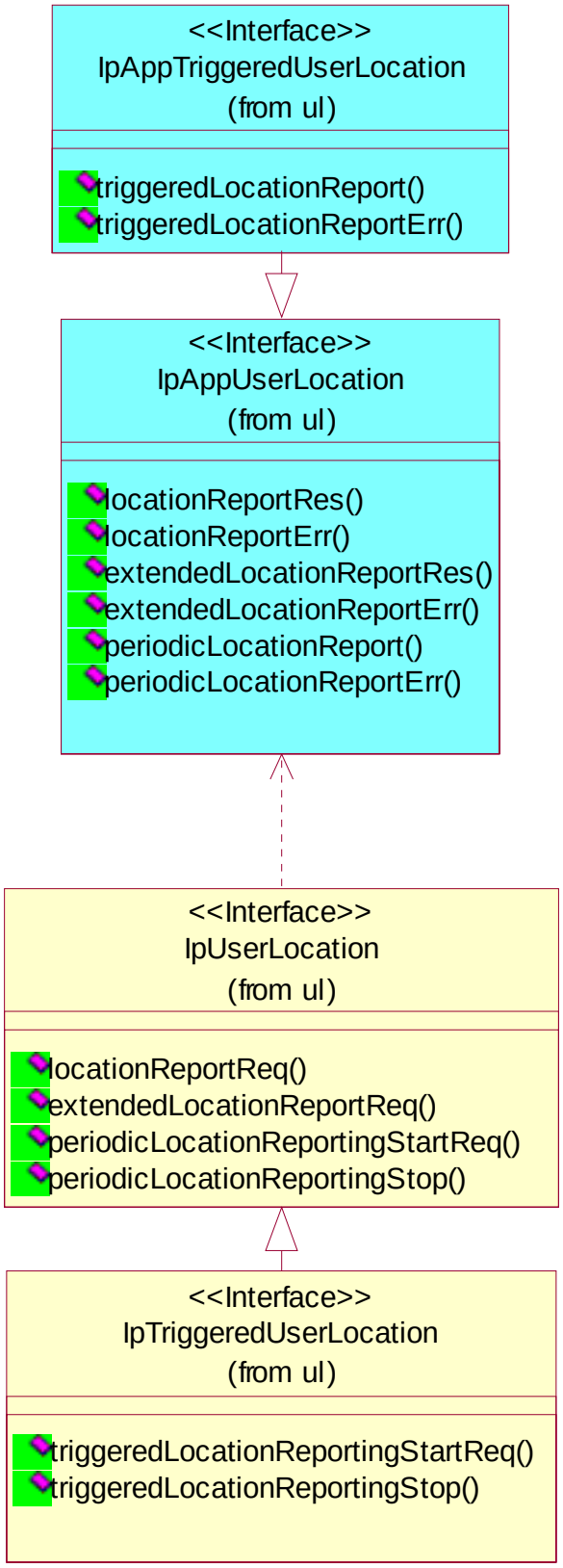


Figure: User Location Class Diagram

6.2 User Location Camel Class Diagrams

This class diagram shows the interfaces for the User Location Camel service.

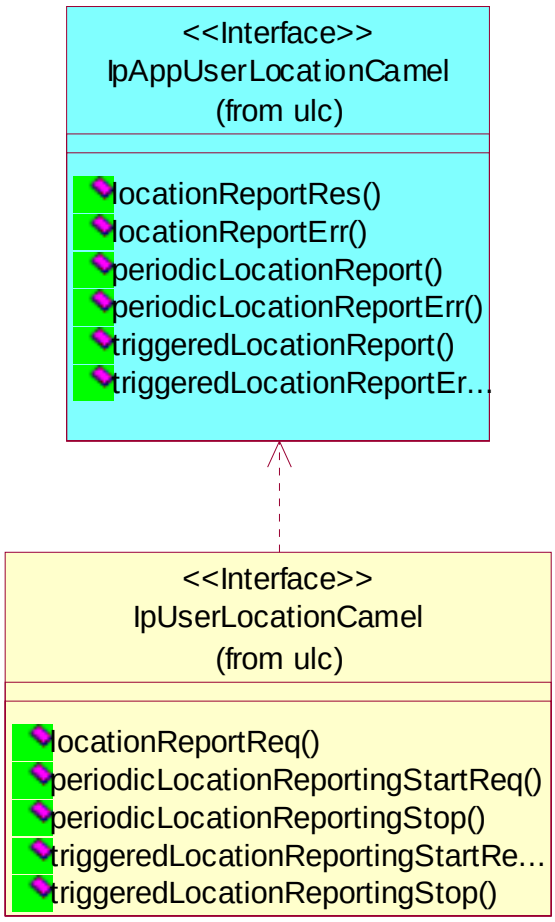


Figure: User Location Camel Class Diagram

6.3 User Status Class Diagrams

This class diagram shows the interfaces for the User Status service.

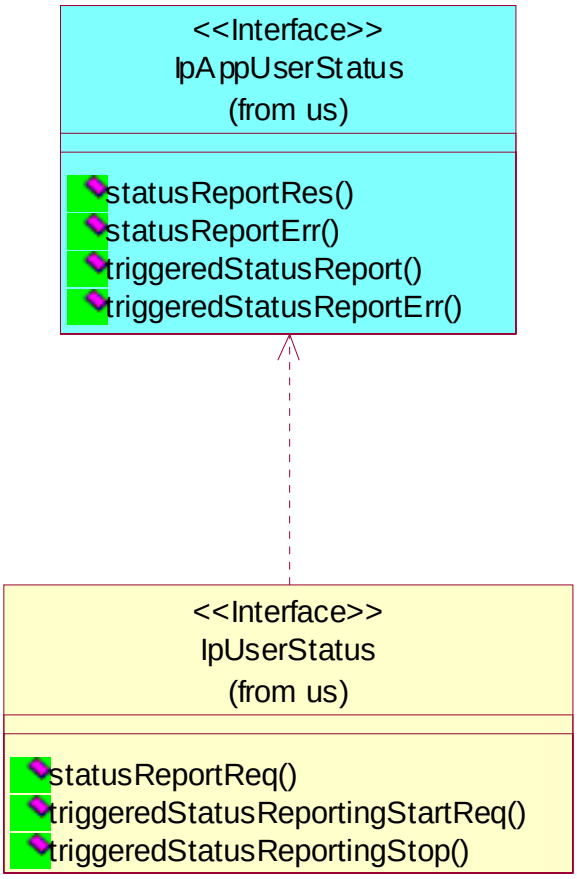


Figure: User Status Class Diagram

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.

<<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. It is not allowed to invoke this method on an interface that uses SessionID's.

Parameters

appInterface : in IpInterfaceRef

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

Raises

TpCommonExceptions

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. It is not allowed to invoke this method on an interface that does not uses SessionID's.

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

TpCommonExceptions

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.

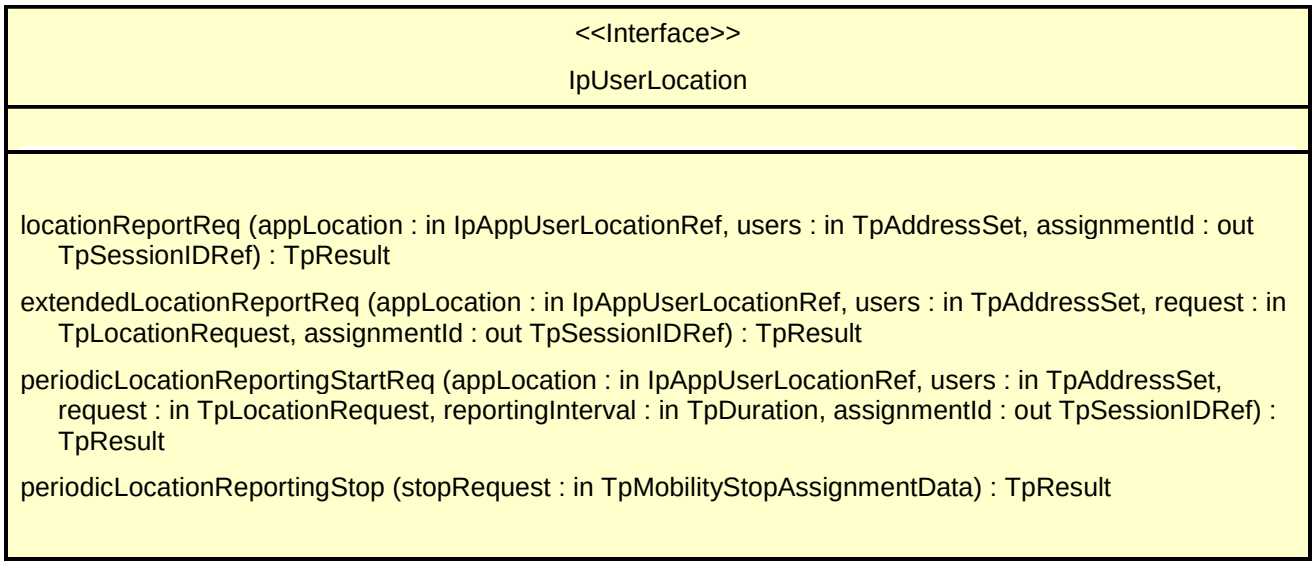
When periodic or triggered location reporting is used, errors may be reported either when the recurrent reporting is requested, as an error per user in reports or in the corresponding err-method when the error concerns all subscribers in an assignment.

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_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_UNKNOWN_SUBSCRIBER

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_INFORMATION_NOT_AVAILABLE

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

TpCommonExceptions, P_APPLICATION_NOT_ACTIVATED, P_INFORMATION_NOT_AVAILABLE, P_UNKNOWN_SUBSCRIBER

Method

extendedLocationReportReq()

Advanced request of report on the location for one or several users.

Raises the following exceptions:

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_UNKNOWN_SUBSCRIBER

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_INFORMATION_NOT_AVAILABLE

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

TpCommonExceptions,P_APPLICATION_NOT_ACTIVATED,P_REQUESTED_ACCURACY_CANNOT_BE_DELIVERED,P_REQUESTED_RESPONSE_TIME_CANNOT_BE_DELIVERED,P_UNKNOWN_SUBSCRIBER,P_INFORMATION_NOT_AVAILABLE

Method

periodicLocationReportingStartReq()

Request of periodic reports on the location for one or several users.

Raises the following exceptions:

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_UNKNOWN_SUBSCRIBER

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_INFORMATION_NOT_AVAILABLE

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

TpCommonExceptions, P_INVALID_REPORTING_INTERVAL, P_REQUESTED_ACCURACY_CANNOT_BE_DELIVERED, P_REQUESTED_RESPONSE_TIME_CANNOT_BE_DELIVERED, P_UNKNOWN_SUBSCRIBER, P_APPLICATION_NOT_ACTIVATED, P_INFORMATION_NOT_AVAILABLE

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.

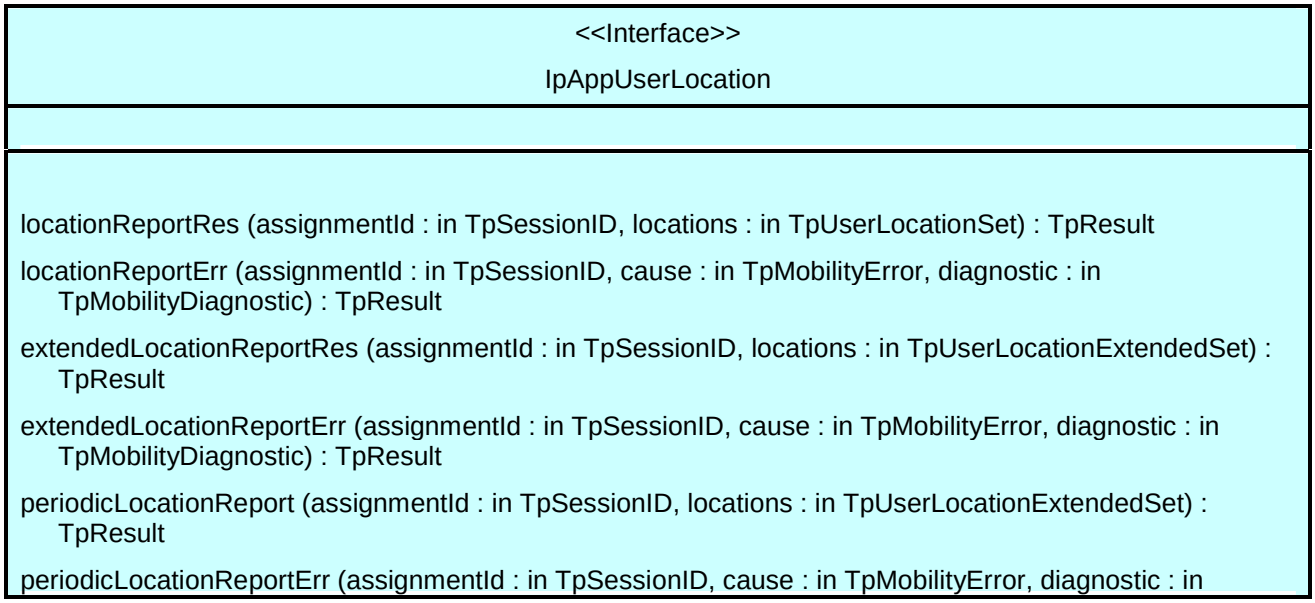
Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

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.



TpMobilityDiagnostic) : TpResult

Method

locationReportRes()

A report containing locations for one or several users is delivered.

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.

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.

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.

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.

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.

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

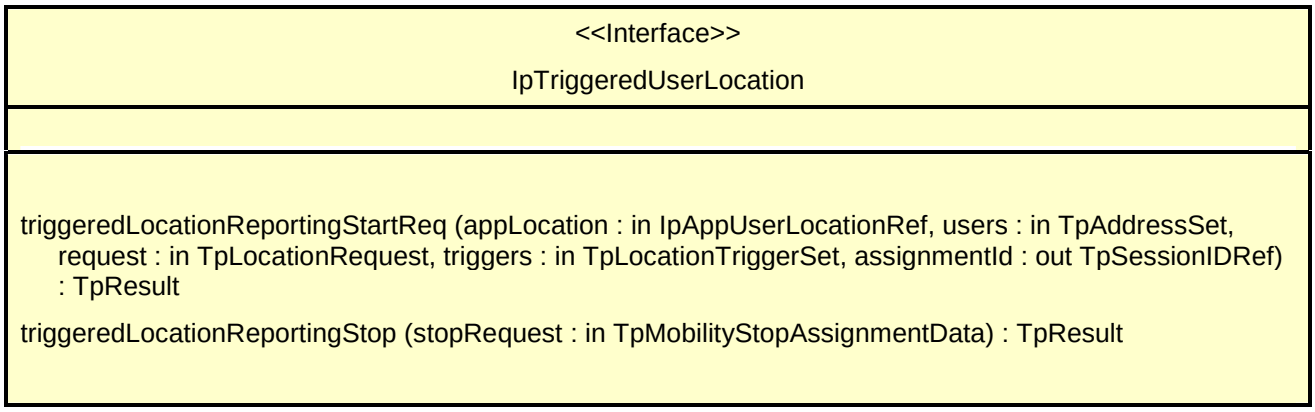
Specifies additional information about the error that led to the failure.

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

TpCommonExceptions,P_REQUESTED_ACCURACY_CANNOT_BE_DELIVERED,P_REQUESTED_RESPONSE_TIME_CANNOT_BE_DELIVERED,P_TRIGGER_CONDITIONS_NOT_SUBSCRIBED,P_UNKNOWN_SUBSCRIBER,P_APPLICATION_NOT_ACTIVATED,P_INFORMATION_NOT_AVAILABLE

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

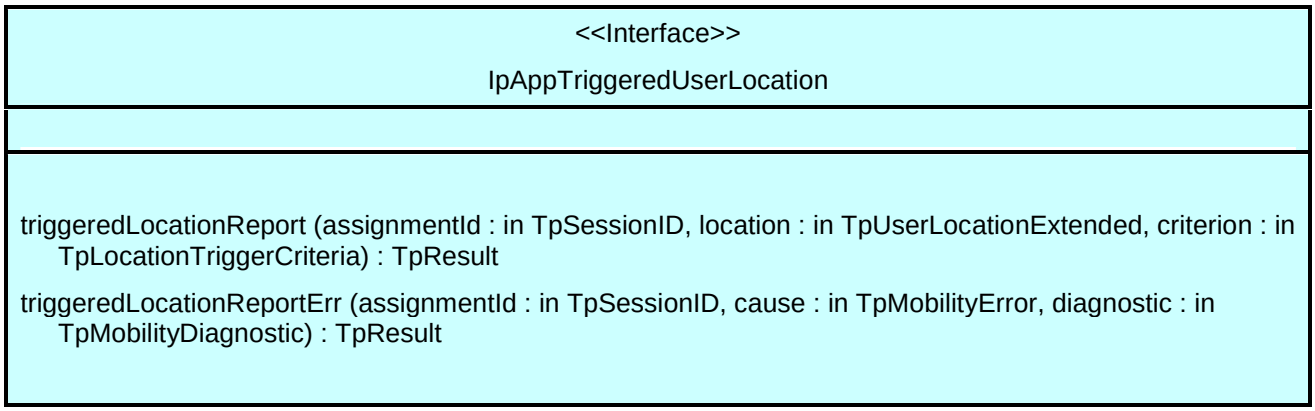
TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

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.

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.

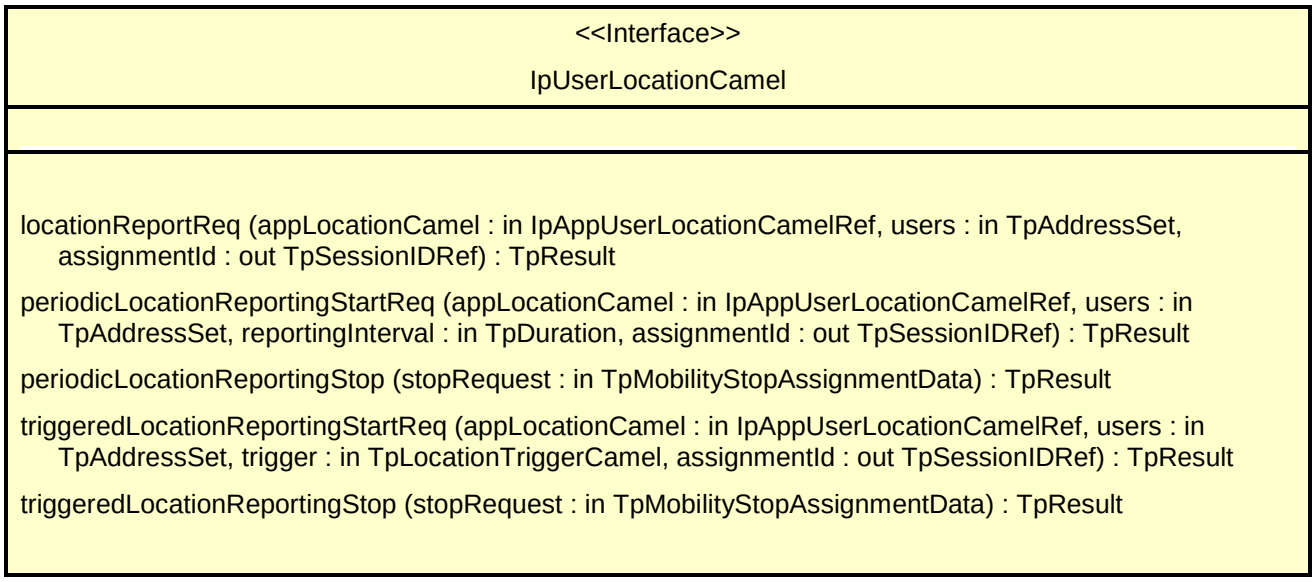
Using the ULC functions, an application programmer can request the VLR Number, the location Area Identification and the Cell Global Identification and other mobile-telephony-specific location information

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_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_UNKNOWN_SUBSCRIBER

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_INFORMATION_NOT_AVAILABLE

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

TpCommonExceptions, P_UNKNOWN_SUBSCRIBER, P_APPLICATION_NOT_ACTIVATED, P_INFORMATION_NOT_AVAILABLE

Method

periodicLocationReportingStartReq()

Request for periodic mobile location reports on one or several users.

Raises the following exceptions:

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_UNKNOWN_SUBSCRIBER

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_INFORMATION_NOT_AVAILABLE

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

TpCommonExceptions, P_INVALID_REPORTING_INTERVAL, P_REQUESTED_ACCURACY_CANNOT_BE_DELIVERED, P_REQUESTED_RESPONSE_TIME_CANNOT_BE_DELIVERED, P_UNKNOWN_SUBSCRIBER, P_APPLICATION_NOT_ACTIVATED, P_INFORMATION_NOT_AVAILABLE

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

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

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_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_UNKNOWN_SUBSCRIBER

The end-user is not subscribed to the application.

P_APPLICATION_NOT_ACTIVATED

The end-user has de-activated the application.

P_INFORMATION_NOT_AVAILABLE

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

TpCommonExceptions,P_UNKNOWN_SUBSCRIBER,P_APPLICATION_NOT_ACTIVATED,P_INFORMATION_NOT_AVAILABLE

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.

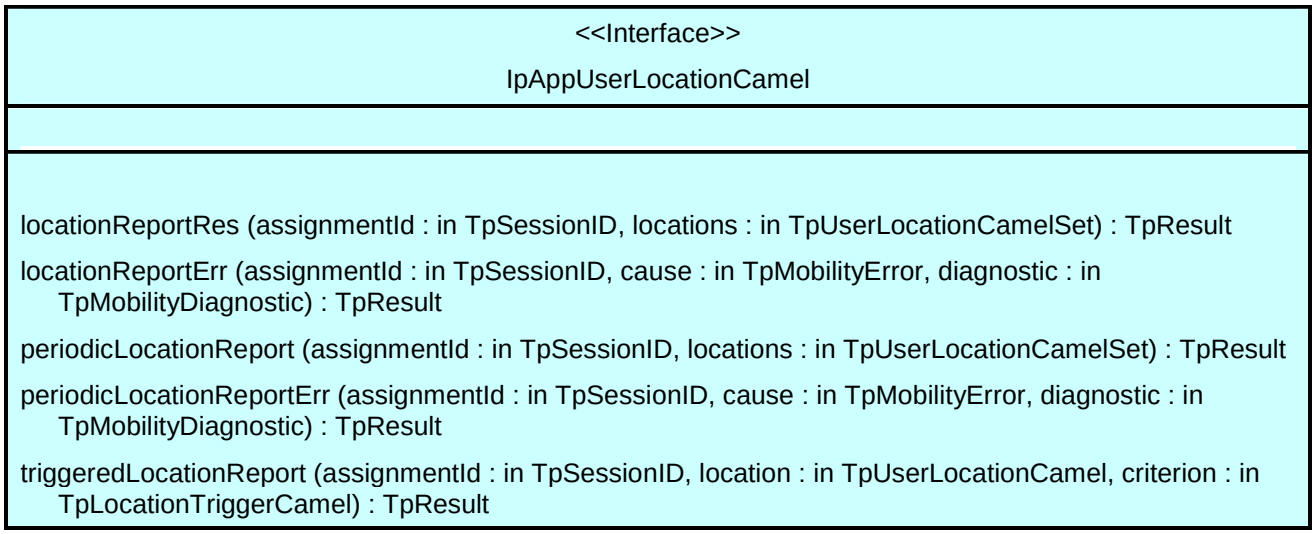
Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

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.



triggeredLocationReportErr (assignmentId : in TpSessionID, cause : in TpMobilityError, diagnostic : in TpMobilityDiagnostic) : TpResult

Method

locationReportRes()

Delivery of a mobile location report. The report is containing mobile-related location information for one or several users.

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.

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.

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.

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.

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.

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

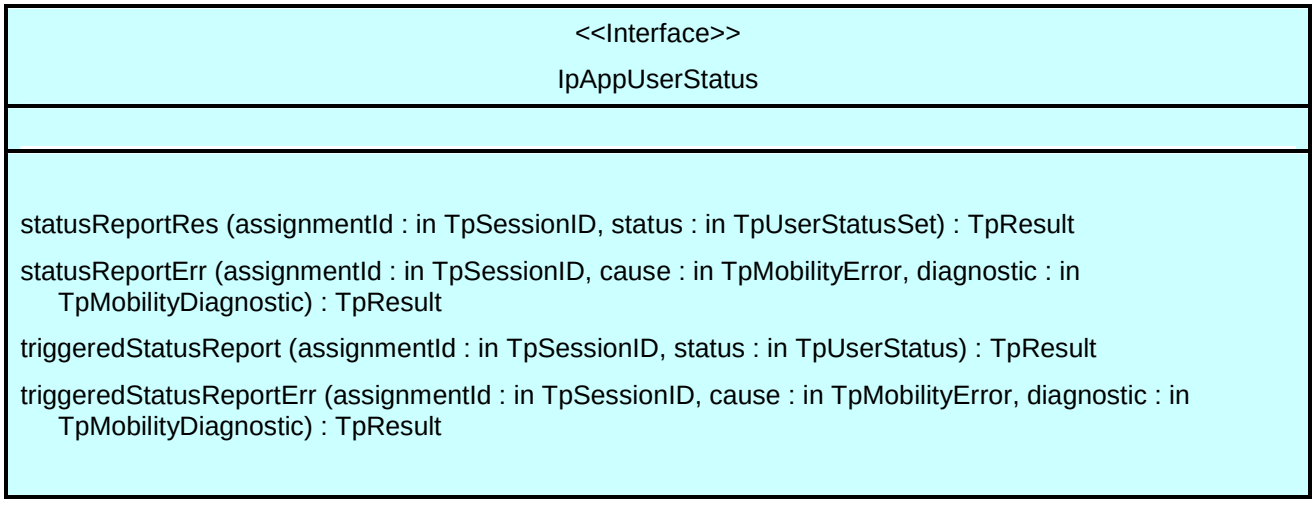
The User Status Service (US) provides a general user status service. US allow applications to obtain the status of fixed, mobile and IP-based telephony users.

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.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the status-report request.

status : in TpUserStatusSet

Specifies the status of one or several users.

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.

Parameters

assignmentId : in TpSessionID

Specifies the assignment ID of the triggered status-reporting request.

status : in TpUserStatus

Specifies the status of the user.

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.

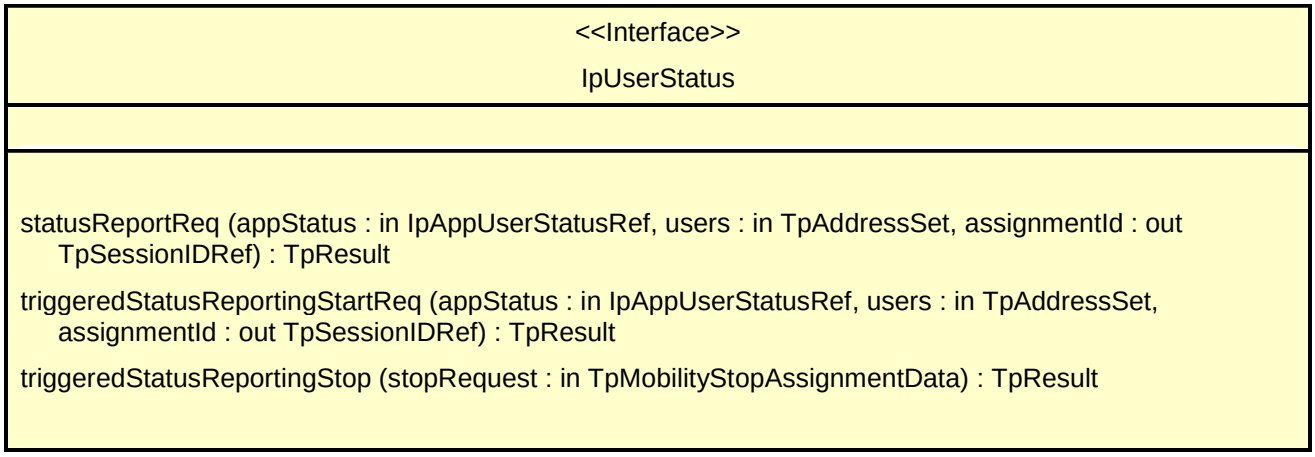
diagnostic : in TpMobilityDiagnostic

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_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

TpCommonExceptions

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_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

TpCommonExceptions

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

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

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.

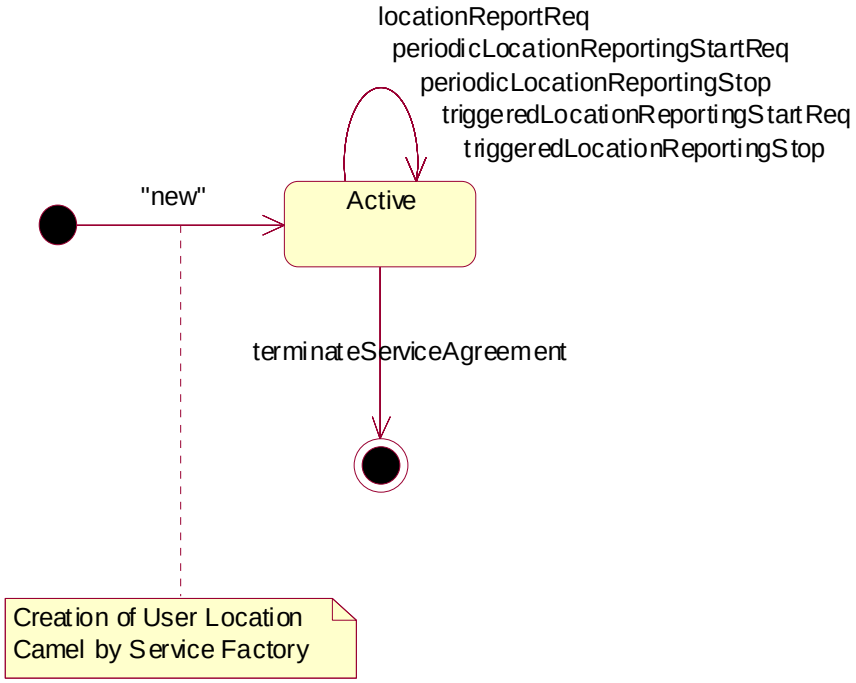


Figure : State Transition Diagram for User Location Camel

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

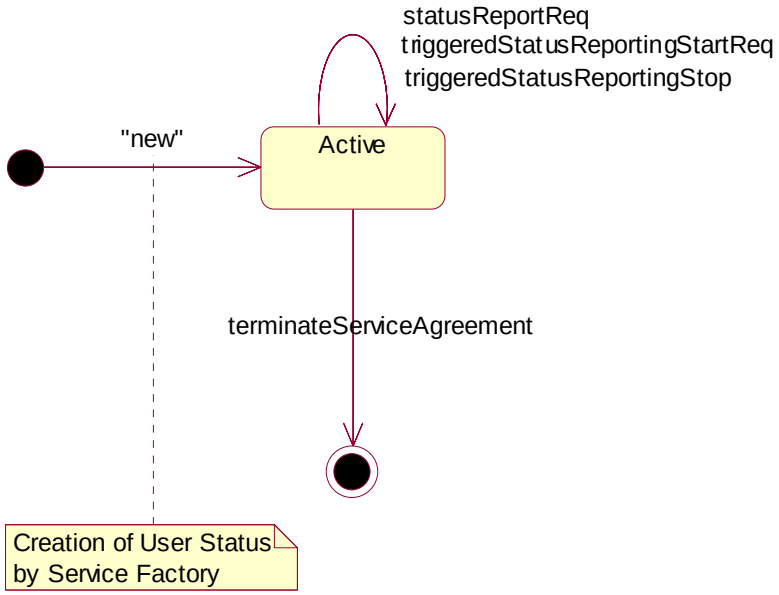


Figure : State Transition Diagram for User Status

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 Application Subtypes

Emergency (see definition of 'LCS Client Type' in GSM 09.02) 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"

10.1.2 Value Added Application Subtypes

Value Added (see definition of 'LCS Client Type' in GSM 09.02) 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 Application Subtypes

PLMN Operator (see definition of 'LCS Client Type' in GSM 09.02.) 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 Application Subtypes

Lawful Intercept (See definition of 'LCS Client Type' in GSM 09.02.) 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"
- "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 (see definition of 'LCSClientType' in GSM 09.02.)
- Altitude obtainable
- Location methods
- Max interactive requests
- Max triggered users
- Max periodic users
- Min periodic interval duration

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

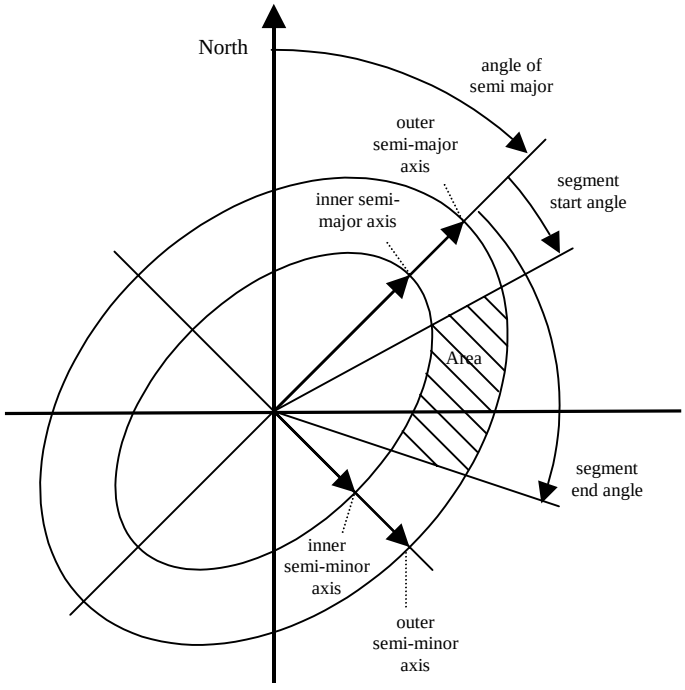


Figure 1 Description of an Ellipse Arc

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 requirement 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 3GPP TS 29.002 [4] 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 (see Note)	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.
NOTE: The location number is the number to the MSC or in rare cases the roaming number.		

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.

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 (see Note)	2	User is busy (only applicable for interactive user status request, not when triggers are used)
NOTE: Only applicable to mobile (Camel) telephony users.		

11.6 Units and Validations of Parameters

This clause 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 clause Location Methods.

12 Exception Classes

The following are the list of exception classes which are used in this interface of the API.

Name	Description
P_INVALID_REPORTING_INTERVAL	The requested reporting interval is not valid
P_REQUESTED_ACCURACY_CANNOT_BE_DELIVERED	The requested location accuracy cannot be delivered
P_REQUESTED_RESPONSE_TIME_CANNOT_BE_DELIVERED	The requested response time cannot be delivered
P_TRIGGER_CONDITIONS_NOT_SUBSCRIBED	Trigger conditions not subscribed

Each exception class contains the following structure:

Structure Element Name	Structure Element Type	Structure Element Description
extraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name

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 TS 29.198 R99

B.1 All Interfaces

All methods on IpApp interfaces no longer throw exceptions.

All methods on the other interfaces throw TpCommonExceptions and individual, identified exceptions

~~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.~~

Annex C (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
16 Mar 2001	CN_11	NP-010134	047	-	CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158)	3.2.0	4.0.0
9 June 2001	CN#12		001		CR 29.198-6: Corrections to OSA API Rel4	4.0.0	4.0.1

```
//Source file: mm.idl
//Date:  9 June 2001


#ifndef __MM_DEFINED
#define __MM_DEFINED



#include "osa.idl"

module org {

      module csapi {

            module mm {

                  enum TpLocationPriority {
                        P_M_NORMAL,
                        P_M_HIGH
                  };

                  enum TpLocationResponseIndicator {
                        P_M_NO_DELAY,
                        P_M_LOW_DELAY,
                        P_M_DELAY_TOLERANT,
                        P_M_USE_TIMER_VALUE
                  };

                  struct TpLocationResponseTime {
                        TpLocationResponseIndicator ResponseTime;

                        TpInt32 TimerValue;


                  };

                  enum TpLocationTriggerCriteria {
                        P_UL_ENTERING_AREA,
                        P_UL_LEAVING_AREA
                  };

                  struct TpLocationTrigger {
                        TpFloat Longitude;

                        TpFloat Latitude;

                        TpFloat AreaSemiMajor;

                        TpFloat AreaSemiMinor;

                        TpInt32 AngleOfSemiMajor;

                        TpLocationTriggerCriteria Criterion;

                        TpDuration ReportingInterval;


                  };

                  typedef sequence <TpLocationTrigger> TpLocationTriggerSet;
```

```
enum TpLocationType {
      P_M_CURRENT,
      P_M_CURRENT_OR_LAST_KNOWN,
      P_M_INITIAL
};



struct TpLocationRequest {
      TpFloat RequestedAccuracy;

      TpLocationResponseTime RequestedResponseTime;

      TpBoolean AltitudeRequested;

      TpLocationType Type;

      TpLocationPriority Priority;

      TpString RequestedLocationMethod;


};

enum TpLocationUncertaintyShape {
      P_M_SHAPE_NONE,
      P_M_SHAPE_CIRCLE,
      P_M_SHAPE_CIRCLE_SECTOR,
      P_M_SHAPE_CIRCLE_ARC_STRIPE,
      P_M_SHAPE_ELLIPSE,
      P_M_SHAPE_ELLIPSE_SECTOR,
      P_M_SHAPE_ELLIPSE_ARC_STRIPE
};

struct TpGeographicalPosition {
      TpFloat Longitude;

      TpFloat Latitude;

      TpLocationUncertaintyShape TypeOfUncertaintyShape;

      TpFloat UncertaintyInnerSemiMajor;

      TpFloat UncertaintyOuterSemiMajor;

      TpFloat UncertaintyInnerSemiMinor;

      TpFloat UncertaintyOuterSemiMinor;

      TpInt32 AngleOfSemiMajor;

      TpInt32 SegmentStartAngle;

      TpInt32 SegmentEndAngle;


};
```

```
enum TpMobilityDiagnostic {
      P_M_NO_INFORMATION,
      P_M_APPL_NOT_IN_PRIV_EXCEPT_LST,
      P_M_CALL_TO_USER_NOT_SETUP,
      P_M_PRIVACY_OVERRIDE_NOT_APPLIC,
      P_M_DISALL_BY_LOCAL_REGULAT_REQ,
      P_M_CONGESTION,
      P_M_INSUFFICIENT_RESOURCES,
      P_M_INSUFFICIENT_MEAS_DATA,
      P_M_INCONSISTENT_MEAS_DATA,
      P_M_LOC_PROC_NOT_COMPLETED,
      P_M_LOC_PROC_NOT_SUPP_BY_USER,
      P_M_QOS_NOT_ATTAINABLE
};



enum TpMobilityError {
      P_M_OK,
      P_M_SYSTEM_FAILURE,
      P_M_UNAUTHORIZED_NETWORK,
      P_M_UNAUTHORIZED_APPLICATION,
      P_M_UNKNOWN_SUBSCRIBER,
      P_M_ABSENT_SUBSCRIBER,
      P_M_POSITION_METHOD_FAILURE
};

enum TpMobilityStopScope {
      P_M_ALL_IN_ASSIGNMENT,
      P_M_SPECIFIED_USERS
};

struct TpMobilityStopAssignmentData {

      TpSessionID AssignmentId;

      TpMobilityStopScope StopScope;

      TpAddressSet Users;


};

enum TpTerminalType {
      P_M_FIXED,
      P_M_MOBILE,
      P_M_IP
};

struct TpUlExtendedData {
      TpGeographicalPosition GeographicalPosition;

      TpTerminalType TerminalType;

      TpBoolean AltitudePresent;

      TpFloat Altitude;

      TpFloat UncertaintyAltitude;

      TpBoolean TimestampPresent;
```

```
            TpDateAndTime Timestamp;

            TpString UsedLocationMethod;


      };

      typedef sequence <TpUlExtendedData> TpUlExtendedDataSet;

      struct TpUserLocation {
            TpAddress UserID;

            TpMobilityError StatusCode;

            TpGeographicalPosition GeographicalPosition;


      };

      struct TpUserLocationExtended {
            TpAddress UserID;

            TpMobilityError StatusCode;

            TpUlExtendedDataSet Locations;


      };

      typedef sequence <TpUserLocationExtended>
TpUserLocationExtendedSet;

      typedef sequence <TpUserLocation> TpUserLocationSet;

      typedef TpString TpLocationCellIDOrLAI;

      struct TpLocationTriggerCamel {
            TpBoolean UpdateInsideVlr;

            TpBoolean UpdateOutsideVlr;


      };

      struct TpUserLocationCamel {
            TpAddress UserID;

            TpMobilityError StatusCode;

            TpBoolean GeographicalPositionPresent;

            TpGeographicalPosition GeographicalPosition;

            TpBoolean TimestampPresent;

            TpDateAndTime Timestamp;

            TpBoolean VlrNumberPresent;

            TpAddress VlrNumber;
```

```
        TpBoolean LocationNumberPresent;

        TpAddress LocationNumber;

        TpBoolean CellIdOrLaiPresent;

        TpLocationCellIDOrLAI CellIdOrLai;


};

typedef sequence <TpUserLocationCamel> TpUserLocationCamelSet;

typedef TpString TpIMEI;

typedef TpString TpNaESRD;

typedef TpString TpNaESRK;

struct TpUserLocationEmergencyRequest {
        TpBoolean UserAddressPresent;

        TpAddress UserAddress;

        TpBoolean NaEsrdPresent;

        TpNaESRD NaEsrd;

        TpBoolean NaEsrkPresent;

        TpNaESRK NaEsrk;

        TpBoolean ImeiPresent;

        TpIMEI Imei;

        TpLocationRequest LocationReq;


};

enum TpUserLocationEmergencyTrigger {
        P_ULE_CALL_ORIGINATION,
        P_ULE_CALL_RELEASE,
        P_ULE_LOCATION_REQUEST
};

struct TpUserLocationEmergency {
        TpMobilityError StatusCode;

        TpBoolean UserIdPresent;

        TpAddress UserId;

        TpBoolean NaEsrdPresent;

        TpNaESRD NaEsrd;

        TpBoolean NaEsrkPresent;
```

```
        TpNaESRK NaEsrk;

        TpBoolean ImeiPresent;

        TpIMEI Imei;

        TpUserLocationEmergencyTrigger TriggeringEvent;

        TpBoolean GeographicalPositionPresent;

        TpGeographicalPosition GeographicalPosition;

        TpBoolean AltitudePresent;

        TpFloat Altitude;

        TpFloat UncertaintyAltitude;

        TpBoolean TimestampPresent;

        TpDateAndTime Timestamp;

        TpString UsedLocationMethod;


};

enum TpUserStatusIndicator {
        P_US_REACHABLE,
        P_US_NOT_REACHABLE,
        P_US_BUSY
};

struct TpUserStatus {
        TpAddress UserID;

        TpMobilityError StatusCode;

        TpUserStatusIndicator Status;

        TpTerminalType TerminalType;


};

typedef sequence <TpUserStatus> TpUserStatusSet;

exception P_REQUESTED_ACCURACY_CANNOT_BE_DELIVERED {
        TpString extraInformation;
};

exception P_REQUESTED_RESPONSE_TIME_CANNOT_BE_DELIVERED {
        TpString extraInformation;
};

exception P_INVALID_REPORTING_INTERVAL {
        TpString extraInformation;
};

exception P_TRIGGER_CONDITIONS_NOT_SUBSCRIBED {
        TpString extraInformation;
```

```
};


module ul {


      interface IpAppUserLocation : IpInterface {

            void locationReportRes (
                  in TpSessionID assignmentId,
                  in TpUserLocationSet locations
                  );


            void locationReportErr (
                  in TpSessionID assignmentId,
                  in TpMobilityError cause,
                  in TpMobilityDiagnostic diagnostic
                  );


            void extendedLocationReportRes (
                  in TpSessionID assignmentId,
                  in TpUserLocationExtendedSet locations
                  );


            void extendedLocationReportErr (
                  in TpSessionID assignmentId,
                  in TpMobilityError cause,
                  in TpMobilityDiagnostic diagnostic
                  );


            void periodicLocationReport (
                  in TpSessionID assignmentId,
                  in TpUserLocationExtendedSet locations
                  );


            void periodicLocationReportErr (
                  in TpSessionID assignmentId,
                  in TpMobilityError cause,
                  in TpMobilityDiagnostic diagnostic
                  );

      };


      interface IpUserLocation : IpService {

            void locationReportReq (
                  in IpAppUserLocation appLocation,
                  in TpAddressSet users,
                  out TpSessionID assignmentId
                  )
                  raises (TpCommonExceptions,
P_APPLICATION_NOT_ACTIVATED, P_INFORMATION_NOT_AVAILABLE, P_UNKNOWN_SUBSCRIBER);
```

```
                        void extendedLocationReportReq (
                              in IpAppUserLocation appLocation,
                              in TpAddressSet users,
                              in TpLocationRequest request,
                              out TpSessionID assignmentId
                              )
                              raises
(TpCommonExceptions,P_APPLICATION_NOT_ACTIVATED,P_REQUESTED_ACCURACY_CANNOT_BE_D
ELIVERED,P_REQUESTED_RESPONSE_TIME_CANNOT_BE_DELIVERED,P_UNKNOWN_SUBSCRIBER,P_IN
FORMATION_NOT_AVAILABLE);


                        void periodicLocationReportingStartReq (
                              in IpAppUserLocation appLocation,
                              in TpAddressSet users,
                              in TpLocationRequest request,
                              in TpDuration reportingInterval,
                              out TpSessionID assignmentId
                              )
                              raises (TpCommonExceptions,
P_INVALID_REPORTING_INTERVAL, P_REQUESTED_ACCURACY_CANNOT_BE_DELIVERED,
P_REQUESTED_RESPONSE_TIME_CANNOT_BE_DELIVERED, P_UNKNOWN_SUBSCRIBER,
P_APPLICATION_NOT_ACTIVATED, P_INFORMATION_NOT_AVAILABLE);


                        void periodicLocationReportingStop (
                              in TpMobilityStopAssignmentData stopRequest

                              )
                              raises (TpCommonExceptions,
P_INVALID_ASSIGNMENT_ID);

                  };



                  interface IpTriggeredUserLocation : IpUserLocation {

                        void triggeredLocationReportingStartReq (
                              in IpAppUserLocation appLocation,
                              in TpAddressSet users,
                              in TpLocationRequest request,
                              in TpLocationTriggerSet triggers,
                              out TpSessionID assignmentId
                              )
                              raises
(TpCommonExceptions,P_REQUESTED_ACCURACY_CANNOT_BE_DELIVERED,P_REQUESTED_RESPONS
E_TIME_CANNOT_BE_DELIVERED,P_TRIGGER_CONDITIONS_NOT_SUBSCRIBED,P_UNKNOWN_SUBSCRI
BER,P_APPLICATION_NOT_ACTIVATED,P_INFORMATION_NOT_AVAILABLE);


                        void triggeredLocationReportingStop (
                              in TpMobilityStopAssignmentData stopRequest

                              )
                              raises (TpCommonExceptions,
P_INVALID_ASSIGNMENT_ID);

                  };
```

```
                        interface IpAppTriggeredUserLocation : IpAppUserLocation
{
                              void triggeredLocationReport (
                                    in TpSessionID assignmentId,
                                    in TpUserLocationExtended location,
                                    in TpLocationTriggerCriteria criterion
                                    );


                              void triggeredLocationReportErr (
                                    in TpSessionID assignmentId,
                                    in TpMobilityError cause,
                                    in TpMobilityDiagnostic diagnostic
                                    );

                        };


                  };


                  module ulc {


                        interface IpAppUserLocationCamel : IpInterface {

                              void locationReportRes (
                                    in TpSessionID assignmentId,
                                    in TpUserLocationCamelSet locations
                                    );


                              void locationReportErr (
                                    in TpSessionID assignmentId,
                                    in TpMobilityError cause,
                                    in TpMobilityDiagnostic diagnostic
                                    );


                              void periodicLocationReport (
                                    in TpSessionID assignmentId,
                                    in TpUserLocationCamelSet locations
                                    );


                              void periodicLocationReportErr (
                                    in TpSessionID assignmentId,
                                    in TpMobilityError cause,
                                    in TpMobilityDiagnostic diagnostic
                                    );


                              void triggeredLocationReport (
                                    in TpSessionID assignmentId,
```

```
                              in TpUserLocationCamel location,
                              in TpLocationTriggerCamel criterion
                              );


                        void triggeredLocationReportErr (
                              in TpSessionID assignmentId,
                              in TpMobilityError cause,
                              in TpMobilityDiagnostic diagnostic
                              );

                  };



                  interface IpUserLocationCamel : IpService {

                        void locationReportReq (
                              in IpAppUserLocationCamel appLocationCamel,

                              in TpAddressSet users,
                              out TpSessionID assignmentId
                              )
                              raises (TpCommonExceptions,
P_UNKNOWN_SUBSCRIBER, P_APPLICATION_NOT_ACTIVATED, P_INFORMATION_NOT_AVAILABLE);


                        void periodicLocationReportingStartReq (
                              in IpAppUserLocationCamel appLocationCamel,

                              in TpAddressSet users,
                              in TpDuration reportingInterval,
                              out TpSessionID assignmentId
                              )
                              raises (TpCommonExceptions,
P_INVALID_REPORTING_INTERVAL, P_REQUESTED_ACCURACY_CANNOT_BE_DELIVERED,
P_REQUESTED_RESPONSE_TIME_CANNOT_BE_DELIVERED, P_UNKNOWN_SUBSCRIBER,
P_APPLICATION_NOT_ACTIVATED, P_INFORMATION_NOT_AVAILABLE);


                        void periodicLocationReportingStop (
                              in TpMobilityStopAssignmentData stopRequest

                              )
                              raises (TpCommonExceptions,
P_INVALID_ASSIGNMENT_ID);


                        void triggeredLocationReportingStartReq (
                              in IpAppUserLocationCamel appLocationCamel,

                              in TpAddressSet users,
                              in TpLocationTriggerCamel trigger,
                              out TpSessionID assignmentId
                              )
                              raises
(TpCommonExceptions,P_UNKNOWN_SUBSCRIBER,P_APPLICATION_NOT_ACTIVATED,P_INFORMATI
ON_NOT_AVAILABLE);


                        void triggeredLocationReportingStop (
```

```
                              in TpMobilityStopAssignmentData stopRequest

                              )
                              raises (TpCommonExceptions,
P_INVALID_ASSIGNMENT_ID);

                        };


                  };



                  module us {


                        interface IpAppUserStatus : IpInterface {

                              void statusReportRes (
                                    in TpSessionID assignmentId,
                                    in TpUserStatusSet status
                                    );


                              void statusReportErr (
                                    in TpSessionID assignmentId,
                                    in TpMobilityError cause,
                                    in TpMobilityDiagnostic diagnostic
                                    );


                              void triggeredStatusReport (
                                    in TpSessionID assignmentId,
                                    in TpUserStatus status
                                    );


                              void triggeredStatusReportErr (
                                    in TpSessionID assignmentId,
                                    in TpMobilityError cause,
                                    in TpMobilityDiagnostic diagnostic
                                    );

                        };


                        interface IpUserStatus : IpService {

                              void statusReportReq (
                                    in IpAppUserStatus appStatus,
                                    in TpAddressSet users,
                                    out TpSessionID assignmentId
                                    )
                                    raises (TpCommonExceptions);
```

```
                              void triggeredStatusReportingStartReq (
                                    in IpAppUserStatus appStatus,
                                    in TpAddressSet users,
                                    out TpSessionID assignmentId
                                    )
                                    raises (TpCommonExceptions);


                              void triggeredStatusReportingStop (
                                    in TpMobilityStopAssignmentData stopRequest

                                    )
                                    raises (TpCommonExceptions,
P_INVALID_ASSIGNMENT_ID);

                        };



                  };

            };

      };

};

#endif
```

CR-Form-v3

CHANGE REQUEST

⌘ **29.198-7** **CR** **001** ⌘ rev **-** ⌘ Current version: **4.0.0** ⌘

*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:	⌘	Corrections to OSA API Rel4
Source:	⌘	CN5
Work item code:	⌘	OSA1
Date:	⌘	07/06/2001
Category:	⌘	**F**
Release:	⌘	Rel4

Use one of the following categories:
F *(essential correction)*
A *(corresponds to a correction in an earlier release)*
B *(Addition of feature),*
C *(Functional modification of feature)*
D *(Editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
REL-4	*(Release 4)*
REL-5	*(Release 5)*

Reason for change:	⌘	Exception handling mechanism in 29.198 requires correction to enable it to be correctly used, without ambiguity
Summary of change:	⌘	Replace TpGeneralException, TpTermCapException with detailed exception classes which can be thrown for each method
Consequences if not approved:	⌘	29.198-7 will be ambiguous and difficult to implement correctly - inter-working might be jeopardised

Clauses affected:	⌘				
Other specs affected:	⌘	**X**	Other core specifications	⌘	All other parts of 29.198 except part 1 have similar changes
		☐	Test specifications		
		☐	O&M Specifications		
Other comments:	⌘				

3GPP TS 29.198-7 V4.0.~~0~~ 1 (2001-~~03~~06)

Technical Specification

3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API); Part 7: Terminal Capabilities; (Release 4)

Keywords
UMTS, 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.

Introduction

The present document is part 7 of a multi-part TS covering the 3rd Generation Partnership Project: Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API), as identified below. The **API specification** (3GPP TS 29.198) is structured in the following Parts:

Part 1:	Overview	
Part 2:	Common Data Definitions	
Part 3:	Framework	
Part 4:	Call Control SCF	
Part 5:	User Interaction SCF	
Part 6:	Mobility SCF	
Part 7:	**Terminal Capabilities SCF**	
Part 8:	Data Session Control SCF	
Part 9:	Generic Messaging SCF	(not part of 3GPP Release 4)
Part 10:	Connectivity Manager SCF	(not part of 3GPP Release 4)
Part 11:	Account Management SCF	
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.

OSA API specifications 29.198-family		OSA API Mapping - 29.998-family	
29.198-1	**Part 1: Overview**	**29.998-1**	**Part 1: Overview**
29.198-2	**Part 2: Common Data Definitions**	29.998-2	Not Applicable
29.198-3	**Part 3: Framework**	29.998-3	Not Applicable
29.198-4	**Part 4: Call Control SCF**	**29.998-4-1**	**Subpart 1: Generic Call Control – CAP mapping**
		29.998-4-2	
29.198-5	**Part 5: User Interaction SCF**	**29.998-5-1**	**Subpart 1: User Interaction – CAP mapping**
		29.998-5-2	
		29.998-5-3	
		29.998-5-4	**Subpart 4: User Interaction – SMS mapping**
29.198-6	**Part 6: Mobility SCF**	**29.998-6**	**User Status and User Location – MAP mapping**
29.198-7	**Part 7: Terminal Capabilities SCF**	29.998-7	Not Applicable
29.198-8	**Part 8: Data Session Control SCF**	**29.998-8**	**Data Session Control – CAP mapping**
29.198-9	Part 9: Generic Messaging SCF	29.998-9	Not Applicable
29.198-10	Part 10: Connectivity Manager SCF	29.998-10	Not Applicable
29.198-11	**Part 11: Account Management SCF**	29.998-11	Not Applicable
29.198-12	**Part 12: Charging SCF**	29.998-12	Not Applicable

1 Scope

The present 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 APIs. The concepts and the functional architecture for the OSA are contained in 3GPP TS 23.127 [3]. The requirements for OSA are contained in 3GPP TS 22.127 [2].

The present 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, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document.*

[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 and abbreviations

3.1 Definitions

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

3.2 Abbreviations

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

4 Terminal Capabilities SCF

The following clauses 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 SCF is implemented.
- The Class relationships clause show how each of the interfaces applicable to the SCF, relate to one another.
- The Interface specification clause describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the the transition between states in the SCF. The states and transitions are well-defined; either methods specified in the Interface specification or events occurring in the underlying networks cause state transitions.~~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:

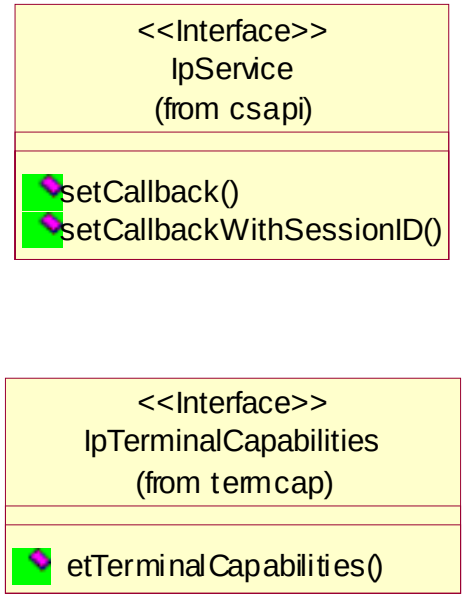


Figure: Package Overview

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. It is not allowed to invoke this method on an interface that uses SessionID's.

Parameters

appInterface : in IpInterfaceRef

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

Raises

TpCommonExceptions

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. It is not allowed to invoke this method on an interface that does not uses SessionID's.

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

TpCommonExceptions

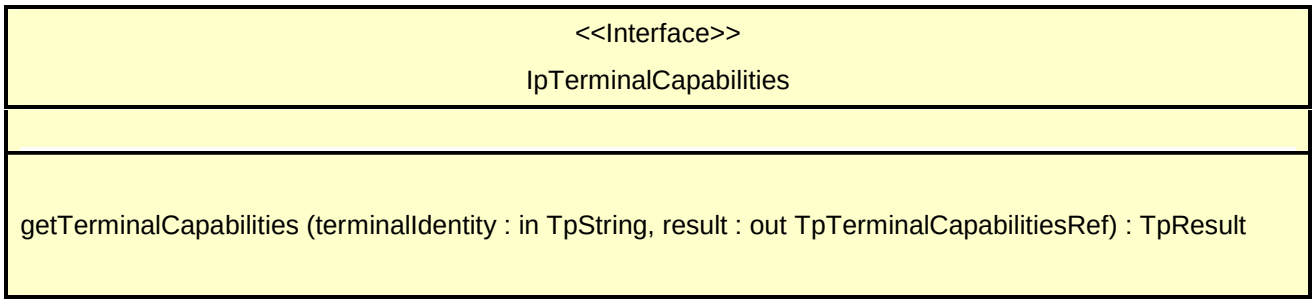
8 Terminal Capabilities Interface Classes

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).



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

TpCommonExceptions, P_INVALID_TERMINAL_ID

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 clauses 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 [4] and adopted in the WAP UAProf specification [5]. 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.

11 Exception Classes

The following are the list of exception classes which are used in this interface of the API.

Name	Description
P_INVALID_TERMINAL_ID	The request can not be handled because the terminal id specified is not valid.

Each exception class contains the following structure:

Structure Element Name	Structure Element Type	Structure Element Description
extraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name

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 TS 29.198 R99

getTerminalCapabilities now throws TpCommonExceptions and individual, identified exceptions

~~None recorded.~~

Annex C (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
16 Mar 2001	CN_11	NP-010134	047	-	CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158)	3.2.0	4.0.0
9 June 2001	CN#12		001		CR 29.198-7: Corrections to OSA API Rel4	4.0.0	4.0.1

```
//Source file: termcap.idl
//Date:  9 June 2001


#ifndef __TERMCAP_DEFINED
#define __TERMCAP_DEFINED



#include "osa.idl"

module org {

      module csapi {



            module termcap {




                  struct TpTerminalCapabilities {
                        TpBoolean StatusCode;
                        TpString TerminalCapabilities;
                  };

                  enum TpTerminalCapabilitiesError {
                        P_TERMCAP_ERROR_UNDEFINED,
                        P_TERMCAP_INVALID_TERMINALID,
                        P_TERMCAP_SYSTEM_FAILURE
                  };

                  exception TpTermCapException {
                        TpTerminalCapabilitiesError error;
                  };

                  exception P_INVALID_TERMINAL_ID {
                        TpString extraInformation;
                  };


                  interface IpTerminalCapabilities : IpInterface {

                        void getTerminalCapabilities (
                              in TpString terminalIdentity,
                              out TpTerminalCapabilities result
                              )
                              raises (TpCommonExceptions,
P_INVALID_TERMINAL_ID);

                  };

            };

      };

};

#endif
```

3GPP TSG_CN5 (Open Service Access – OSA) **Tdoc N5-010266**
Meeting #11, San Diego, CA, USA, 21 – 24 May 2001

CR-Form-v3

CHANGE REQUEST

⌘ **29.198-8** **CR** **001** ⌘ rev **-** ⌘ Current version: **4.0.0** ⌘

*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:	⌘	Corrections to OSA API Rel4
Source:	⌘	CN5
Work item code:	⌘	OSA1
Date:	⌘	07/06/2001
Category:	⌘	**F**
Release:	⌘	Rel4

Use one of the following categories:
F *(essential correction)*
A *(corresponds to a correction in an earlier release)*
B *(Addition of feature),*
C *(Functional modification of feature)*
D *(Editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
REL-4	*(Release 4)*
REL-5	*(Release 5)*

Reason for change:	⌘	Exception handling mechanism in 29.198 requires correction to enable it to be correctly used, without ambiguity; Addition of QoS information reporting; Addition of Service Properties
Summary of change:	⌘	Replace TpGeneralException, TpDSCSException with detailed exception classes which can be thrown for each method; Add QoS information in reports to application; Add service properties table in clause 10.
Consequences if not approved:	⌘	29.198-8 will be ambiguous and difficult to implement correctly - inter-working might be jeopardised

Clauses affected:	⌘			
Other specs affected:	⌘	**X** Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	All other parts of 29.198 except part 1 have similar changes
Other comments:	⌘			

3GPP TS 29.198-8 V4.0.~~0~~ 1 (2001-06~~3~~)

Technical Specification

3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API); Part 8: Data Session Control (Release 4)

Keywords
UMTS, 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.

Introduction

The present document is part 8 of a multi-part TS covering the 3rd Generation Partnership Project: Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API), as identified below. The **API specification** (3GPP TS 29.198) is structured in the following Parts:

Part 1:	Overview	
Part 2:	Common Data Definitions	
Part 3:	Framework	
Part 4:	Call Control SCF	
Part 5:	User Interaction SCF	
Part 6:	Mobility SCF	
Part 7:	Terminal Capabilities SCF	
Part 8:	**Data Session Control SCF**	
Part 9:	Generic Messaging SCF	(not part of 3GPP Release 4)
Part 10:	Connectivity Manager SCF	(not part of 3GPP Release 4)
Part 11:	Account Management SCF	
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.

OSA API specifications 29.198-family		OSA API Mapping - 29.998-family	
29.198-1	**Part 1: Overview**	**29.998-1**	**Part 1: Overview**
29.198-2	**Part 2: Common Data Definitions**	29.998-2	Not Applicable
29.198-3	**Part 3: Framework**	29.998-3	Not Applicable
29.198-4	**Part 4: Call Control SCF**	**29.998-4-1**	**Subpart 1: Generic Call Control – CAP mapping**
		29.998-4-2	
29.198-5	**Part 5: User Interaction SCF**	**29.998-5-1**	**Subpart 1: User Interaction – CAP mapping**
		29.998-5-2	
		29.998-5-3	
		29.998-5-4	**Subpart 4: User Interaction – SMS mapping**
29.198-6	**Part 6: Mobility SCF**	**29.998-6**	**User Status and User Location – MAP mapping**
29.198-7	**Part 7: Terminal Capabilities SCF**	29.998-7	Not Applicable
29.198-8	**Part 8: Data Session Control SCF**	**29.998-8**	**Data Session Control – CAP mapping**
29.198-9	Part 9: Generic Messaging SCF	29.998-9	Not Applicable
29.198-10	Part 10: Connectivity Manager SCF	29.998-10	Not Applicable
29.198-11	**Part 11: Account Management SCF**	29.998-11	Not Applicable
29.198-12	**Part 12: Charging SCF**	29.998-12	Not Applicable

1 Scope

The present document is Part 8 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 APIs. The concepts and the functional architecture for the OSA are contained in 3GPP TS 23.127 [3]. The requirements for OSA are contained in 3GPP TS 22.127 [2].

The present 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 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, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document.*

[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] ISO-4217:1995: "".

3 Definitions and abbreviations

3.1 Definitions

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

3.2 Abbreviations

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

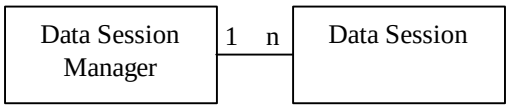
4 Data Session Control SCF

The Data Session Control network SCF 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 SCFs 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.

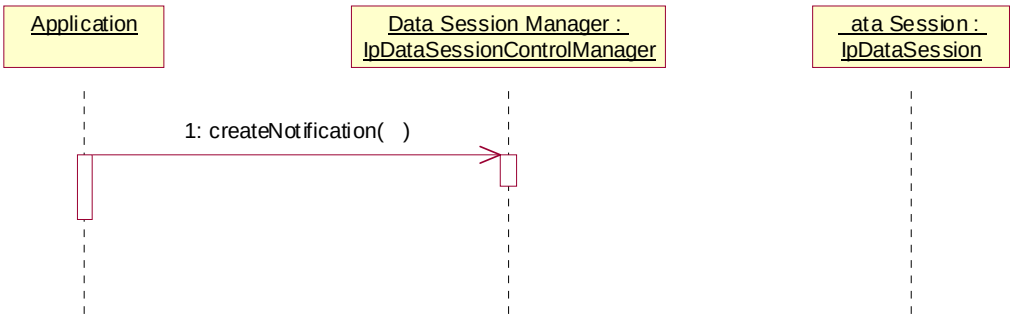
The following clauses describe each aspect of the Data Session Control Service Capability Feature (SCF).

The order is as follows:

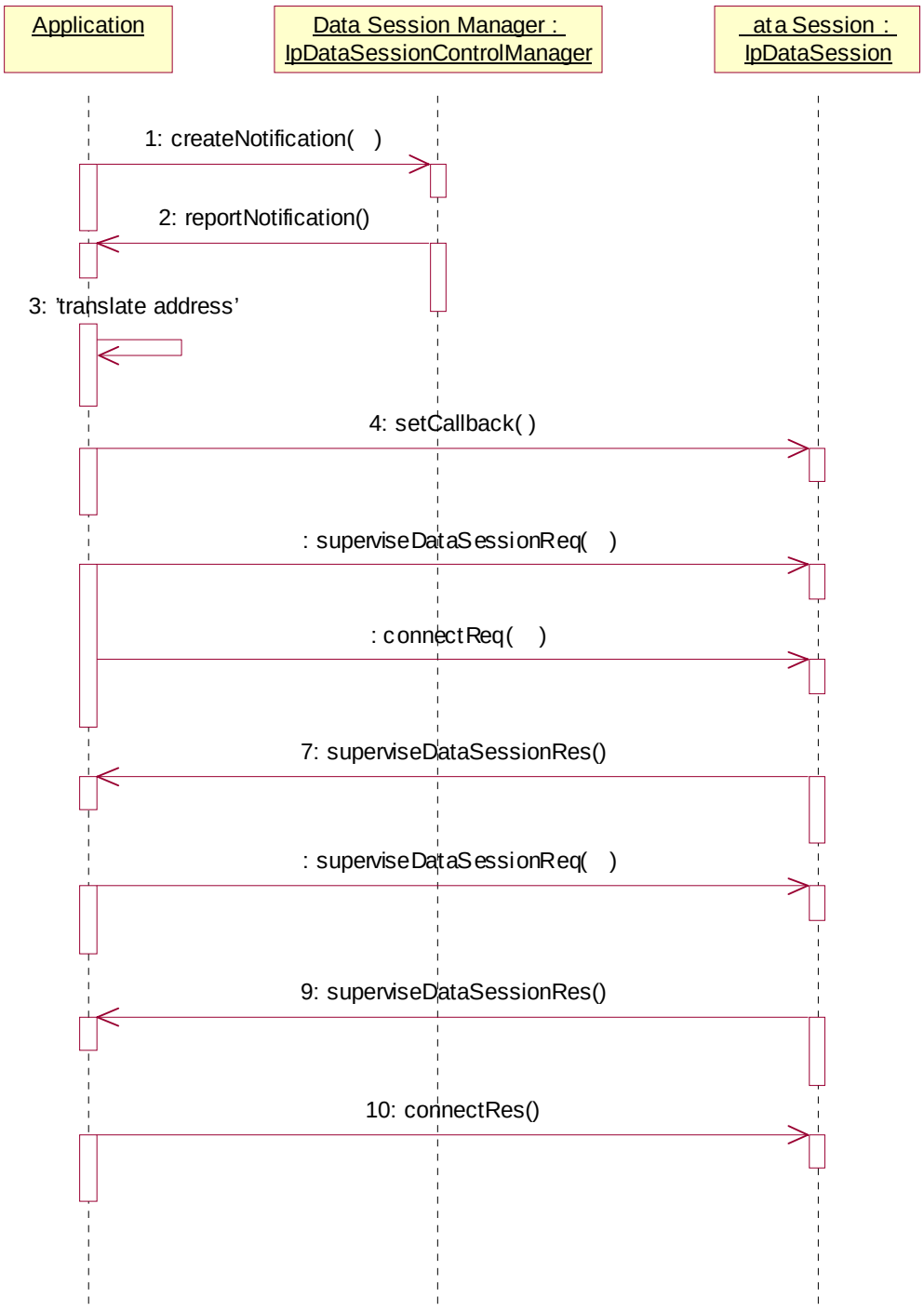
- the Sequence diagrams give the reader a practical idea of how each of the SCF is implemented;
- the Class relationships clause shows how each of the interfaces applicable to the SCF, relate to one another;
- the Interface specification clause describes in detail each of the interfaces shown within the Class diagram part;
- the State Transition Diagrams (STD) show the transition between states in the SCF. The states and transitions are well-defined; either methods specified in the Interface specification or events occurring in the underlying networks cause state transitions;~~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:

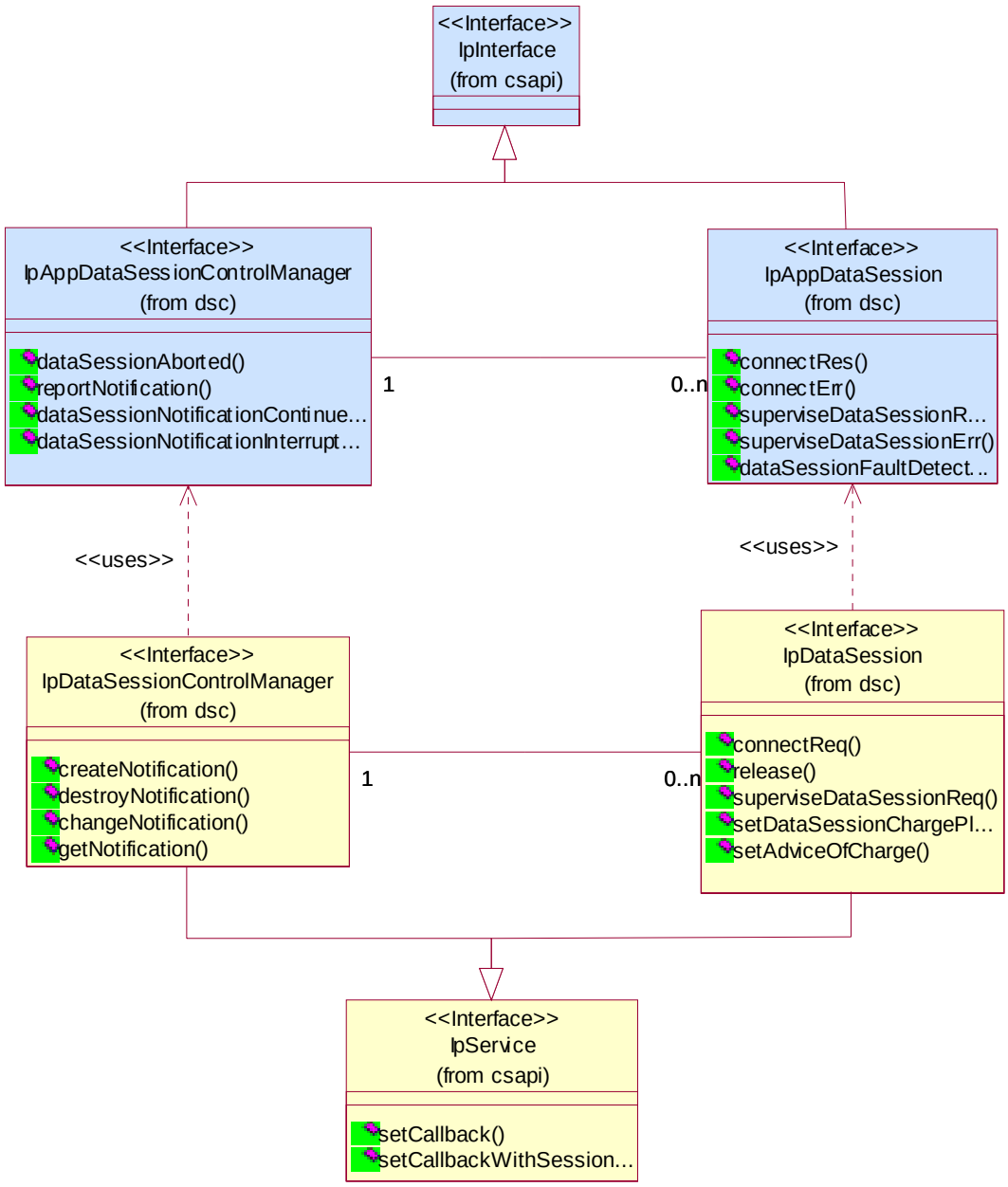


Figure: Package Overview

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.

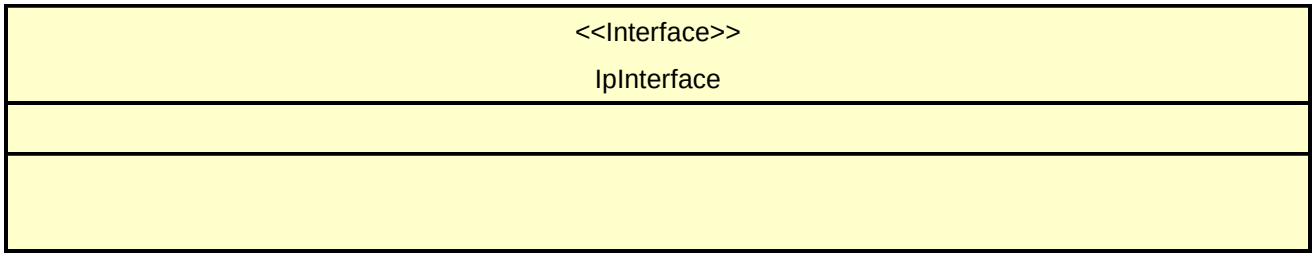
7.1.4 State Model

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. It is not allowed to invoke this method on an interface that uses SessionID's.

Parameters

appInterface: in IpInterfaceRef

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

Raises

TpCommonExceptions

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. It is not allowed to invoke this method on an interface that does not uses SessionID's.

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

TpCommonExceptions

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.

<<Interface>> IpAppDataSession
connectRes (dataSessionID : in TpSessionID, eventReport : in TpDataSessionReport, assignmentID : in TpAssignmentID) : TpResult connectErr (dataSessionID : in TpSessionID, errorIndication : in TpDataSessionError, assignmentID : in TpAssignmentID) : TpResult superviseDataSessionRes (dataSessionID : in TpSessionID, report : in TpDataSessionSuperviseReport, usedVolume : in TpDataSessionSuperviseVolume, qualityOfService : in TpDataSessionQosClass) : TpResult superviseDataSessionErr (dataSessionID : in TpSessionID, errorIndication : in TpDataSessionError) : TpResult dataSessionFaultDetected (dataSessionID : in TpSessionID, fault : in TpDataSessionFault) : TpResult

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, negotiated quality of service and event specific information such as release cause.

assignmentID: in TpAssignmentID

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

Method

superviseDataSessionRes()

This asynchronous method reports a data session supervision event to the application. In addition, it may also be used to notify the application of a newly negotiated set of Quality of Service parameters during the active life of the data session.

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).

qualityOfService: in TpDataSessionQosClass

Specifies the newly negotiated Quality of Service parameters for the data session.

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.

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

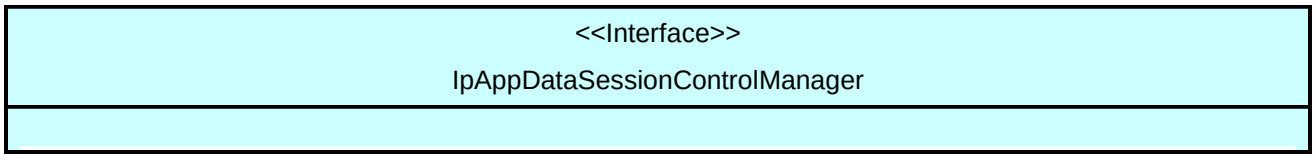
fault: in TpDataSessionFault

Specifies the fault that has been detected.

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.



dataSessionAborted (dataSession : in TpSessionID) : TpResult

reportNotification (dataSessionReference : in TpDataSessionIdentifier, eventInfo : in TpDataSessionEventInfo, assignmentID : in TpAssignmentID, appDataSession : out IpAppDataSessionRefRef) : TpResult

dataSessionNotificationContinued () : TpResult

dataSessionNotificationInterrupted () : TpResult

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.

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 and the quality of service requested or negotiated for the data session.

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.

Method

dataSessionNotificationContinued()

This method indicates to the application that all event notifications are resumed.

Parameters
No Parameters were identified for this method

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

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

TpCommonExceptions, P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED, P_INVALID_NETWORK_STATE, P_INVALID_ADDRESS, P_INVALID_SESSION_ID

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

TpCommonExceptions, P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED, P_INVALID_NETWORK_STATE, P_INVALID_SESSION_ID

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

TpCommonExceptions, P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED, P_INVALID_NETWORK_STATE, P_INVALID_SESSION_ID

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

TpCommonExceptions, P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED, P_INVALID_NETWORK_STATE, P_INVALID_SESSION_ID

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.

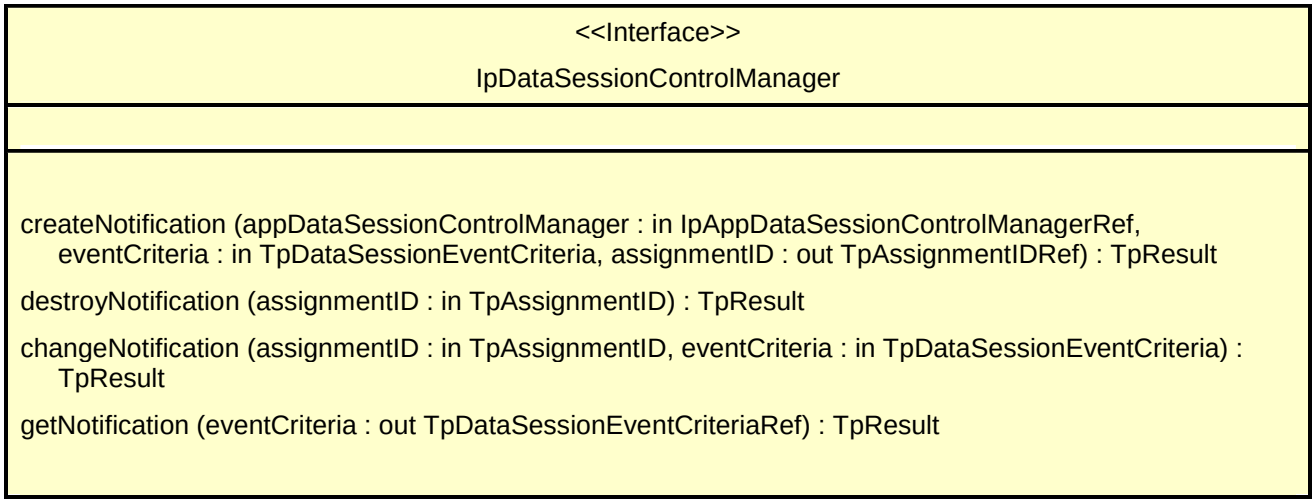
Raises

TpCommonExceptions, P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED, P_INVALID_NETWORK_STATE, P_INVALID_TIME_AND_DATE_FORMAT

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

TpCommonExceptions, P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED, P_INVALID_NETWORK_STATE, P_INVALID_ADDRESS, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE

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

TpCommonExceptions, P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED, P_INVALID_NETWORK_STATE, P_INVALID_ASSIGNMENT_ID

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

TpCommonExceptions, P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED, P_INVALID_NETWORK_STATE, P_INVALID_ASSIGNMENT_ID, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE

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.

Raises

TpCommonExceptions, P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED, P_INVALID_NETWORK_STATE

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.

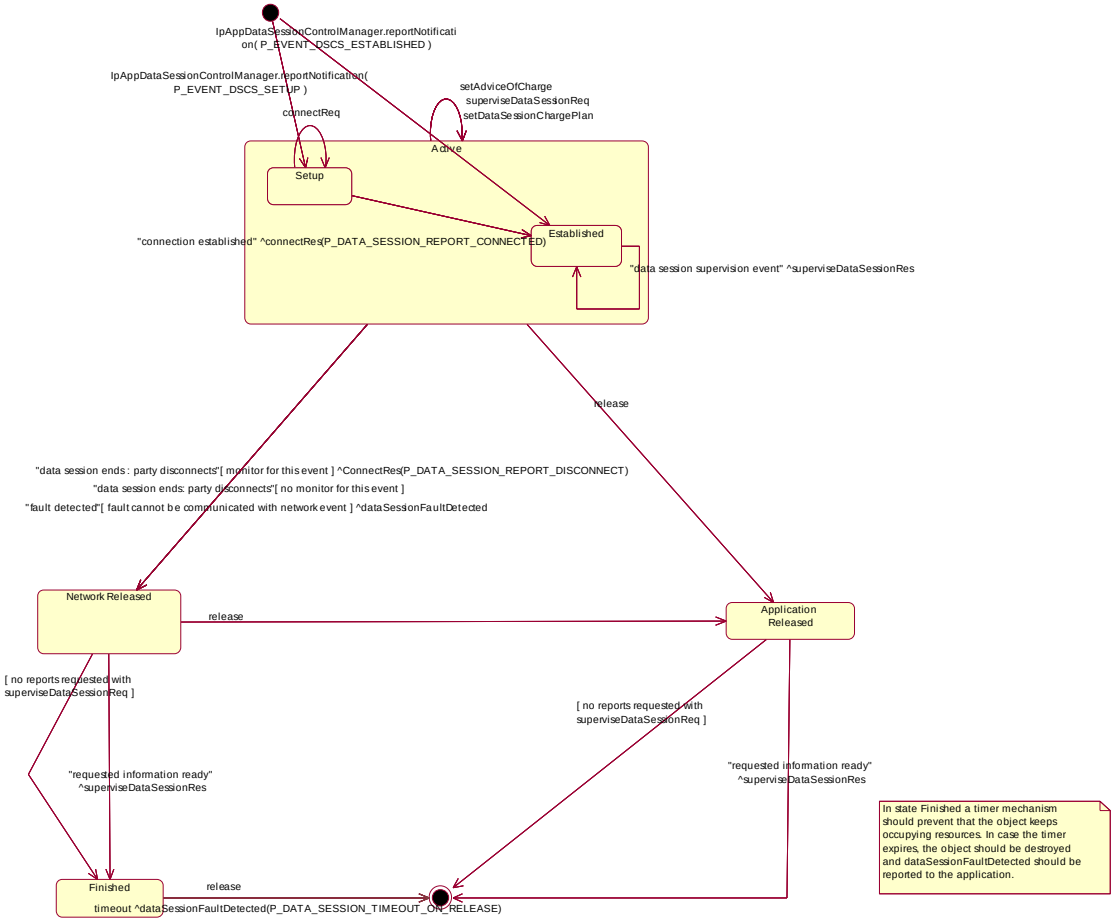


Figure: Application view on the Data Session object

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().

Data Session Control Service Properties

The following table lists properties relevant for the Data Session 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_IP.

The previous table lists properties related to the 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/Interpretation
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_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_TARGET_NUMBER}.
P_CHARGEPLAN_ALLOWED	INTEGER_SET	Indicates which charging is allowed in the setDataSessionChargePlan indicator. Allowed values: {P_CHARGE_PER_VOLUME, P_TRANSPARANT_CHARGING, P_CHARGE_PLAN}
P_CHARGEPLAN_MAPPING	INTEGER_INTEGER_MAP	Indicates the mapping of charge plans (we assume they can be indicated with integers) to a logical network charge plan indicator. When the P_CHARGEPLAN_ALLOWED property indicates P_CHARGE_PLAN, then only charge plans in this mapping are allowed.
P_CURRENCY_ALLOWED	STRING_SET	Indicates the currencies that are allowed to be set for the charge plan in the setDataSessionChargePlan. The valid values for the string set are according to ISO-4217:1995. E.g. {“EUR”, “NLG”}.

1011 Data Definitions

10.111.1 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~~11.2 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.
P_EVENT_DSCS_QOS_CHANGED	4	A change in QoS class has taken place during the life of the data session.

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 establishment 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	Description
DestinationAddress	TpAddress	Defines the destination address for which the notification is reported.
OriginatingAddress	TpAddress	Defines the origination address for which the notification is reported.
DataSessionEventName	TpDataSessionEventName	Name of the event(s)
MonitorMode	TpDataSessionMonitorMode	Defines the mode in which the Data Session is reporting the notification. Monitor mode P_DATA_SESSION_MONITOR_MODE_DO_NOT_MONITOR is not a legal value here.
QoSClass	TpDataSessionQosClass	Defines the Quality of Service (QoS) class for the Data Session. QoSClass NULL is not a legal value when DataSessionEventName is set to P_EVENT_DSCS_QOS_CHANGED. For this particular event, the QoSClass defines the new QoS class effective after the change.

TpDataSessionQosClass

Defines the Quality of Service (QoS) classes for a data session.

Name	Value	Description
P_DATA_SESSION_QOS_CLASS_CONVERSATIONAL	0	Specifies the Conversational QoS class, as specified in 3G TS 23.107.
P_DATA_SESSION_QOS_CLASS_STREAMING	1	Specifies the Streaming QoS class, as specified in 3G TS 23.107.
P_DATA_SESSION_QOS_CLASS_INTERACTIVE	2	Specifies the Interactive QoS class, as specified in 3G TS 23.107.
P_DATA_SESSION_QOS_CLASS_BACKGROUND	3	Specifies the Background QoS class, as specified in 3G TS 23.107.

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 [4]
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_SESSION_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 shall 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

11 Exception Classes

The following are the list of exception classes, which are used in this interface of the API.

Name	Description
P_SERVICE_INFORMATION_MISSING	Information relating to the Data Session Control SCF could not be found
P_SERVICE_FAULT_ENCOUNTERED	Fault detected in the Data Session Control SCF

Each exception class contains the following structure:

Structure Element Name	Structure Element Type	Structure Element Description
extraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name

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 TS 29.198 R99

BC.1 Interface IpAppDataSessionControlManager

reportNotification~~dataSessionEventNotify~~ (dataSessionReference : in TpDataSessionIdentifier, eventInfo : in TpDataSessionEventInfo, assignmentID : in TpAssignmentID, app~~Interface~~DataSession : out IpAppDataSessionRefRef) : TpResult

BC.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

B.3 All Interfaces

All methods on IpApp interfaces no longer throw exceptions.

All methods on the other interfaces throw TpCommonExceptions and individual, identified exceptions

Annex C (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
16 Mar 2001	CN_11	NP-010134	047	-	CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158)	3.2.0	1.0.0
10 June 2001	CN#12		001		CR 29.198-8: Corrections to OSA API Rel4	4.0.0	4.0.1

```
//Source file: dsc.idl
//Date:  10 June 2001


#ifndef __DSC_DEFINED
#define __DSC_DEFINED



#include "osa.idl"

module org {

      module csapi {



            module dsc {
                  interface IpAppDataSessionControlManager;
                  interface IpDataSessionControlManager;
                  interface IpDataSession;

                  const TpInt32 P_DATA_SESSION_SUPERVISE_INFORM = 4;


                  const TpInt32 P_DATA_SESSION_SUPERVISE_DATA_SESSION_ENDED = 2;


                  const TpInt32 P_DATA_SESSION_SUPERVISE_MESSAGE_SENT = 4;


                  const TpInt32 P_DATA_SESSION_SUPERVISE_RELEASE = 1;


                  const TpInt32 P_DATA_SESSION_SUPERVISE_RESPOND = 2;


                  const TpInt32 P_DATA_SESSION_SUPERVISE_VOLUME_REACHED = 1;

                  struct TpChargePerVolume {
                        TpInt32 InitialCharge;
                        TpInt32 CurrentChargePerKilobyte;
                        TpInt32 NextChargePerKilobyte;
                  };

                  enum TpDataSessionChargeOrderCategory {
                        P_DATA_SESSION_CHARGE_PER_VOLUME,
                        P_DATA_SESSION_CHARGE_NETWORK
                  };

                  union TpDataSessionChargeOrder
switch(TpDataSessionChargeOrderCategory) {
                        case P_DATA_SESSION_CHARGE_PER_VOLUME: TpChargePerVolume
ChargePerVolume;
                        case P_DATA_SESSION_CHARGE_NETWORK: TpString
NetworkCharge;
                  };

                  struct TpDataSessionChargePlan {
                        TpDataSessionChargeOrder ChargeOrderType;
                        TpString Currency;
```

```
                  TpString AdditionalInfo;
            };


            enum TpDataSessionErrorType {
                  P_DATA_SESSION_ERROR_UNDEFINED,
                  P_DATA_SESSION_ERROR_INVALID_ADDRESS,

                  P_DATA_SESSION_ERROR_INVALID_STATE
            };


            union TpDataSessionAdditionalErrorInfo
switch(TpDataSessionErrorType) {
                  case P_DATA_SESSION_ERROR_INVALID_ADDRESS:
TpAddressError DataSessionErrorInvalidAddress;
            };


            struct TpDataSessionError {
                  TpDateAndTime ErrorTime;
                  TpDataSessionErrorType ErrorType;
                  TpDataSessionAdditionalErrorInfo AdditionalErrorInfo;
            };


            typedef TpInt32 TpDataSessionEventName;


            enum TpDataSessionFault {
                  P_DATA_SESSION_FAULT_UNDEFINED,
                  P_DATA_SESSION_FAULT_USER_ABORTED,
                  P_DATA_SESSION_TIMEOUT_ON_RELEASE,
                  P_DATA_SESSION_TIMEOUT_ON_INTERRUPT
            };


            enum TpDataSessionMonitorMode {
                  P_DATA_SESSION_MONITOR_MODE_INTERRUPT,

                  P_DATA_SESSION_MONITOR_MODE_NOTIFY,
                  P_DATA_SESSION_MONITOR_MODE_DO_NOT_MONITOR

            };


            struct TpDataSessionEventCriteria {
                  TpAddressRange DestinationAddress;

                  TpAddressRange OriginationAddress;

                  TpDataSessionEventName DataSessionEventName;

                  TpDataSessionMonitorMode MonitorMode;
```

```
};


struct TpDataSessionReleaseCause {
      TpInt32 Value;
      TpInt32 Location;
};


enum TpDataSessionReportType {
      P_DATA_SESSION_REPORT_UNDEFINED,
      P_DATA_SESSION_REPORT_CONNECTED,
      P_DATA_SESSION_REPORT_DISCONNECT
};


union TpDataSessionAdditionalReportInfo
switch(TpDataSessionReportType) {
      case P_DATA_SESSION_REPORT_DISCONNECT:
TpDataSessionReleaseCause DataSessionDisconnect;
};

struct TpDataSessionReport {
      TpDataSessionMonitorMode MonitorMode;
      TpDateAndTime DataSessionEventTime;
      TpDataSessionReportType DataSessionReportType;
      TpDataSessionAdditionalReportInfo AdditionalReportInfo;
};


struct TpDataSessionReportRequest {
      TpDataSessionMonitorMode MonitorMode;
      TpDataSessionReportType DataSessionReportType;
};


typedef sequence <TpDataSessionReportRequest>
TpDataSessionReportRequestSet;


typedef TpInt32 TpDataSessionSuperviseReport;


typedef TpInt32 TpDataSessionSuperviseTreatment;


struct TpDataSessionSuperviseVolume {
      TpInt32 VolumeQuantity;

      TpInt32 VolumeUnit;
```

```
};

exception TpDSCSException {
      TpInt32 exceptionType;
};



const TpInt32 P_EVENT_NAME_UNDEFINED = 0;


const TpInt32 P_EVENT_DSCS_SETUP = 1;


const TpInt32 P_EVENT_DSCS_ESTABLISHED = 2;
const TpInt32 P_EVENT_NAME_QOD_CHANGED = 4;
enum TpDataSessionQosClass {
      P_DATA_SESSION_QOS_CLASS_CONVERSATIONAL,

      P_DATA_SESSION_QOS_CLASS_STREAMING,
      P_DATA_SESSION_QOS_CLASS_INTERACTIVE,

      P_DATA_SESSION_QOS_CLASS_BACKGROUND
};



struct TpDataSessionEventInfo {
      TpAddress DestinationAddress;
      TpAddress OriginatingAddress;
      TpDataSessionEventName DataSessionEventName;
      TpDataSessionMonitorMode MonitorMode;
      TpDataSessionQosClass QoSClass;
};

exception P_SERVICE_INFORMATION_MISSING {
      TpString extraInformation;
};

exception P_SERVICE_FAULT_ENCOUNTERED {
      TpString extraInformation;
};


struct TpDataSessionIdentifier {
      TpSessionID DataSessionID;
      IpDataSession DataSessionReference;
};



interface IpAppDataSession : IpInterface {


      void connectRes (
            in TpSessionID dataSessionID,
            in TpDataSessionReport eventReport,
            in TpAssignmentID assignmentID
            );
```

```
        void connectErr (
              in TpSessionID dataSessionID,
              in TpDataSessionError errorIndication,
              in TpAssignmentID assignmentID
              );


        void superviseDataSessionRes (
              in TpSessionID dataSessionID,
              in TpDataSessionSuperviseReport report,
              in TpDataSessionSuperviseVolume usedVolume,
              in TpDataSessionQosClass qualityOfService
              );


        void superviseDataSessionErr (
              in TpSessionID dataSessionID,
              in TpDataSessionError errorIndication
              );


        void dataSessionFaultDetected (
              in TpSessionID dataSessionID,
              in TpDataSessionFault fault
              );

  };


  interface IpAppDataSessionControlManager : IpInterface {


        void dataSessionAborted (
              in TpSessionID dataSession
              );


        void reportNotification (
              in TpDataSessionIdentifier dataSessionReference,

              in TpDataSessionEventInfo eventInfo,
              in TpAssignmentID assignmentID,
              out IpAppDataSession appDataSession
              );


        void dataSessionNotificationContinued ();


        void dataSessionNotificationInterrupted ();

  };



  interface IpDataSession : IpService {


        void connectReq (
```

```
                              in TpSessionID dataSessionID,
                              in TpDataSessionReportRequestSet
responseRequested,
                              in TpAddress targetAddress,
                              out TpAssignmentID assignmentID
                              )
                              raises (TpCommonExceptions,
P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED,
P_INVALID_NETWORK_STATE, P_INVALID_ADDRESS, P_INVALID_SESSION_ID);


                        void release (
                              in TpSessionID dataSessionID,
                              in TpDataSessionReleaseCause cause
                              )
                              raises (TpCommonExceptions,
P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED,
P_INVALID_NETWORK_STATE, P_INVALID_SESSION_ID);


                        void superviseDataSessionReq (
                              in TpSessionID dataSessionID,
                              in TpDataSessionSuperviseTreatment treatment,
                              in TpDataSessionSuperviseVolume bytes
                              )
                              raises (TpCommonExceptions,
P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED,
P_INVALID_NETWORK_STATE, P_INVALID_SESSION_ID);


                        void setDataSessionChargePlan (
                              in TpSessionID dataSessionID,
                              in TpDataSessionChargePlan dataSessionChargePlan

                              )
                              raises (TpCommonExceptions,
P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED,
P_INVALID_NETWORK_STATE, P_INVALID_SESSION_ID);


                        void setAdviceOfCharge (
                              in TpSessionID dataSessionID,
                              in TpAoCInfo aoCInfo,
                              in TpDuration tariffSwitch
                              )
                              raises (TpCommonExceptions,
P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED,
P_INVALID_NETWORK_STATE, P_INVALID_TIME_AND_DATE_FORMAT);

                  };



                  interface IpDataSessionControlManager : IpService {


                        void createNotification (
                              in IpAppDataSessionControlManager
appDataSessionControlManager,
                              in TpDataSessionEventCriteria eventCriteria,
                              out TpAssignmentID assignmentID
```

```
                              )
                              raises (TpCommonExceptions,
P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED,
P_INVALID_NETWORK_STATE, P_INVALID_ADDRESS, P_INVALID_CRITERIA,
P_INVALID_EVENT_TYPE);


                        void destroyNotification (
                              in TpAssignmentID assignmentID
                              )
                              raises (TpCommonExceptions,
P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED,
P_INVALID_NETWORK_STATE, P_INVALID_ASSIGNMENT_ID);


                        void changeNotification (
                              in TpAssignmentID assignmentID,
                              in TpDataSessionEventCriteria eventCriteria
                              )
                              raises (TpCommonExceptions,
P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED,
P_INVALID_NETWORK_STATE, P_INVALID_ASSIGNMENT_ID, P_INVALID_CRITERIA,
P_INVALID_EVENT_TYPE);


                        void getNotification (
                              out TpDataSessionEventCriteria eventCriteria
                              )
                              raises (TpCommonExceptions,
P_SERVICE_INFORMATION_MISSING, P_SERVICE_FAULT_ENCOUNTERED,
P_INVALID_NETWORK_STATE);

                  };


            };

      };

};

#endif
```