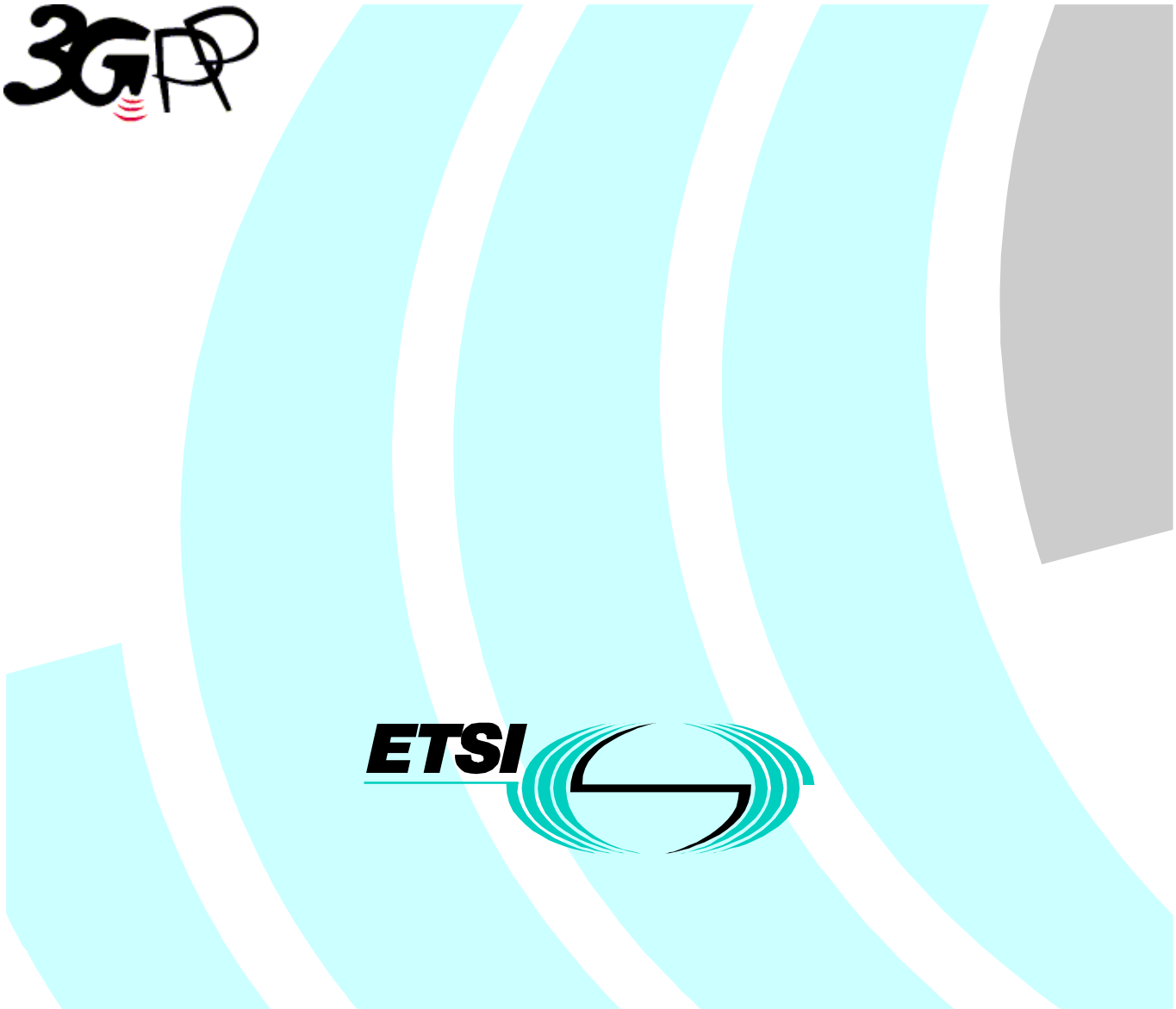
ETSI TS 129 198 V3.4.0 (2001-06)

Technical Specification

Universal Mobile Telecommunications System (UMTS);
Open Service Architecture (OSA)
Application Programming Interface (API) - Part 1
(3GPP TS 29.198 version 3.4.0 Release 1999)



Reference
RTS/TSGN-0529198UR4

Keywords
UMTS

ETSI

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

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

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

Intellectual Property Rights

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

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

Foreword

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

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under www.etsi.org/key .

Contents

Foreword

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

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

Version x.y.z

where:

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

1 Scope

This document specifies the stage 3 of the Open Service Architecture (OSA) Application Programming Interface (API). The concepts and the functional architecture of the Open Service Architecture (API) are described by 3GPP TS 23.127 [2]. This document describes the stage 3 specification of the Open Service Architecture API.

The Open Service Architecture defines an architecture that enables service providers to make use of network functionality through an open standardised interface, i.e. the OSA API. The network functionality is describes as Service Capability Servers. Within the OSA concepts the following Service Capability Servers are identified:

- CAMEL Service Environment (see in 3GPP TS 23.078 [4])
- WAP execution platform (i.e. WAP Gateway & WAP Push Proxy, see in [13])
- Home Location Register (HLR)

The stage 3 documentation of the OSA R'99 API consists of two parts:

- **The API specification (Part 1).**
  This is a normative stage 3 specification of the capabilities of the OSA R'99 API and describes the OSA API interface classes, containing class diagrams (see section 6), state transition diagrams (see section 7), data type definitions (section 8), and the IDLs (see section 9).

- **The Mapping specification of the OSA R'99 API and the network protocols (Part2).**
  This is an informative specification to provide an example how the OSA API can be mapped on the network protocols (i.e. MAP [7], CAP[8] and WAP[9]). It is an informative document, since this mapping is considered as implementation/vendor dependent. On the other hand this mapping will provide potential service designers with a better understanding of the relationship of the OSA API interface classes and the behavior of the network associated to these interface classes.

The OSA API Stage 3 activity is performed jointly with ETSI SPAN3's Service Provider Access Requirements activity. The contents of this document is related to the jointly owned 3GPP & ETSI document referred as the API Master document, which contains the API interface descriptions that are common and differentiated between ETSI & 3GPP.

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 TR 21.905: "3G Vocabulary"

[2] 3GPP TS 23.127: "Virtual Home Environment / Open Service Architecture"

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

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

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

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

[7] 3GPP TS 29.002: “Mobile Application Part (MAP)”

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

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

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

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

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

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

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

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

[16] 3GPP TS 22.002: “Circuit Bearer Services supported by a PLMN”

[17] 3GPP TS 22.003: “Circuit Teleservices supported by a PLMN”

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

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

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

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

[22] ISO 4217: “Codes for the representation of currencies and funds ”

3 Definitions and abbreviations

3.1 Definitions

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

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

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

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

Home Environment: responsible for overall provision of services to users

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

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

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

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

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

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

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

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

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

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

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

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

3.2 Abbreviations

For the purposes of this TS the following abbreviations apply:

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

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

4 Open Service Architecture

The concepts and Architecture of the Open Service Architecture are described within [2]. Within this stage 2 document several Service Capability Features are identified. However for OSA API Release 99, the set of addressed Service Capability Features are limited to the following:

- Framework SCF
  - Service Discovery interface

  - Trust and Security Management interfaces (Initial Contact interfaces and Authentication interfaces)
  - Integrity Management interfaces (Load Manager interfaces, Fault Manager interfaces, OAM interfaces, Heart Beat interfaces)
  - Registration interfaces
- Call Control SCF
- User Interaction SCFs
  - Generic User Interaction SCF
  - Call User Interaction SCF
- Network User Location SCF
- User Status SCF
  - Terminal Capabilities SCF
  - Data Session SCF

The Framework API contains interfaces between the Application Server and the Framework, and between Network Service Capability Server (SCS) and the Framework.

The User Profiles are limited to the Terminal Capabilities for OSA R'99. Therefore, only limited functionality is available for the security within OSA R'99. The Framework & Network SCSs provide the following security mechanisms for OSA R'99:

- Checking the subscriber's registration to the SCS feature
- Checking the subscriber's activation of the SCS feature
- Checking the subscriber's privacy settings of the SCS feature

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

For example, an application that has subscribed to the Network User Location SCF does not have to know whether the SCS provides location reports to the application based on information from the CSE or HLR. Similarly, the application does not have to know whether a message offered to the SCS for delivery to a terminal is actually sent by the SCS to the terminal via a WGP/WPP or SMS-C. It is the Service Capability Server that is capable of deciding how the message is to be sent. The OSA concept therefore leads to a shift of logic on dealing with the network from the applications to the Service Capability Servers.

5 Methodology

Following is a description of the methodology used for the establishment of stage 3 specification in the scope of 3GPP CN OSA.

5.1 Tools and Languages

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

5.2 Packaging

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

org.threegpp.osa

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

5.3 Colours

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

5.4 Naming scheme

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

packages

lowercase.

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

classes, structures and types. Start with T

TpCapitalizedWithInternalWordsAlsoCapitalized

Exception class:

TpClassNameEndsWithException

Interface. Start with Ip:

IpThisIsAnInterface

constants:

P_UPPER_CASE_WITH_UNDERSCORES_AND_START_WITH_P

methods:

firstWordLowerCaseButInternalWordsCapitalized()

method's parameters

firstWordLowerCaseButInternalWordsCapitalized

collections (set, array or list types)

TpCollectionEndsWithSet

class/structure members

FirstWordAndInternalWordsCapitalized

Spaces in between words are not allowed.

5.5 Error results

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

5.6 References

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

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

In IDL, however, the following rules apply:

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

This leads to:

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

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

5.7 Number of out parameters

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

5.8 Strings and Collections

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

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

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

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

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

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

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

```
} TpAddressSet;
```

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

5.9 Prefixes

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

5.10 Naming space across CORBA modules

The following shows the naming space used in this specification.

```
module org {
    module threegpp { // cannot use 3gpp, names need to start with letter
        module osa {
            // The fully qualified name of the following constant
            // is org::threegpp::osa::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 framework {
              // 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= framework::P_FW_CONST;
            };
        };
    };
};
```

6 Class diagrams

Class diagrams are specified in UML: interface classes are shown as interface names within shaded rectangular boxes; relationships and generalizations as lines connecting pairs of interface classes.

All OSA interface classes should be packaged into the org.threegpp.osa module. Further sub-packaging is an implementation decision, but this section proposes a way to do it. Using this recommended packaging, a top-down approach is followed in the subsequent sections. Note that UML packaging is only a logical packaging and does not necessarily reflects IDL packaging.

6.1 Class diagrams common across OSA

All application and framework interfaces inherit from IpOsa interface. Network Service Capability Features on the other hand inherit from the common IpService interface. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

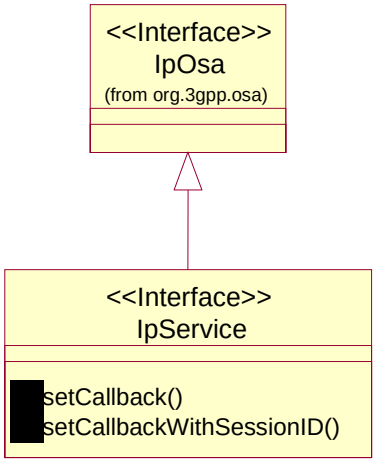
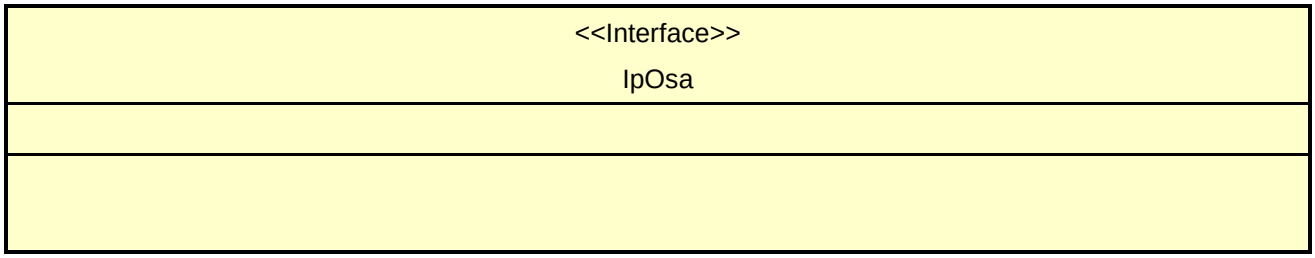


Figure 6-1: OSA base interfaces

6.1.1 Base OSA interface

All application and framework interfaces inherit from the following interface.



6.1.2 Generic Service Capability Feature interface

All Network SCF's interfaces inherit from the following interface.

<<Interface>> IpService
setCallback(appInterface : in IpOsaRef) : TpResult setCallbackWithSessionID(appInterface : in IpOsaRef, sessionID : in TpSessionID): TpResult

6.2 Class diagrams for the Framework

This section specifies the class diagrams that define the Framework, and proposes a way to package them.

6.2.1 Top level Framework packages

The top level view of the Framework consists of the following four packages:

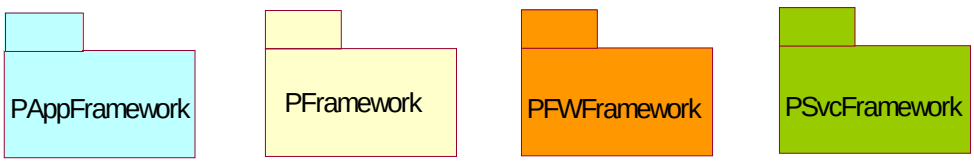
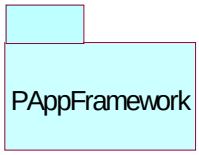
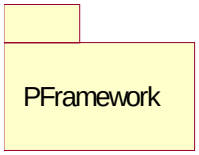


Figure 6-2: Framework top level packages

The first two packages are de-composed in the following way:



Consists of
- *PappTrustAndSecurityMgmt*
- *PAppIntegrityMgmt*



Consists of
- *PServiceDiscovery*
- *PTrustAndSecurityMgmt*
- *PIntegrityMgmt*

The latter two packages contain only one interface each:

- PFWFramework consists of the *Service Registration* Interface
- PSvcFramework consists of the *Service Factory* Interface

The top-level packages are de-composed as described above; between some of the resulting sub-packages there are dependencies, that reflect dependencies between any two classes in the sub-package. The following figure shows all this.

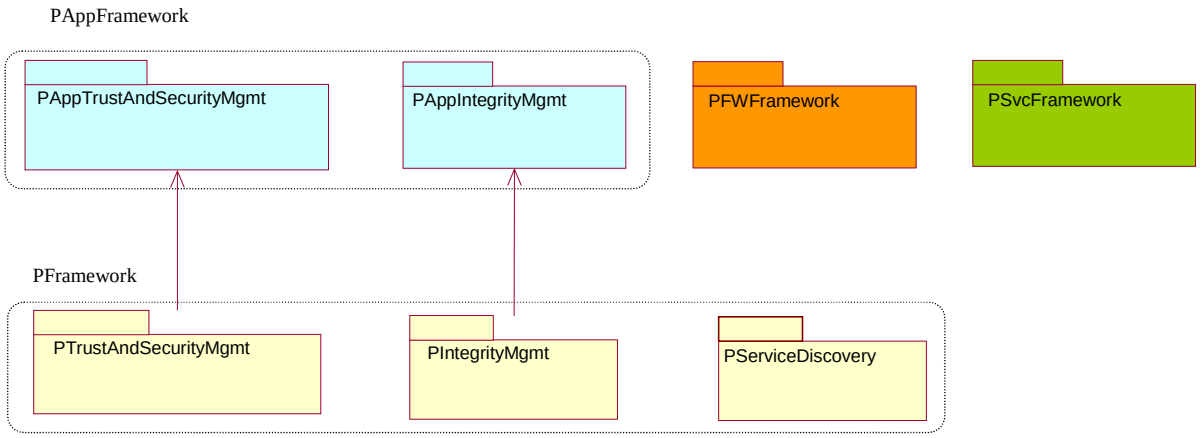


Figure 6-3: Framework sub-packages

6.2.2 Service Discovery

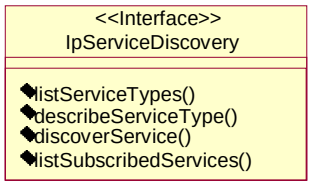


Figure 6-4: Service Discovery Class Diagrams

<<Interface>> IpServiceDiscovery
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

6.2.3 Trust and Security Management

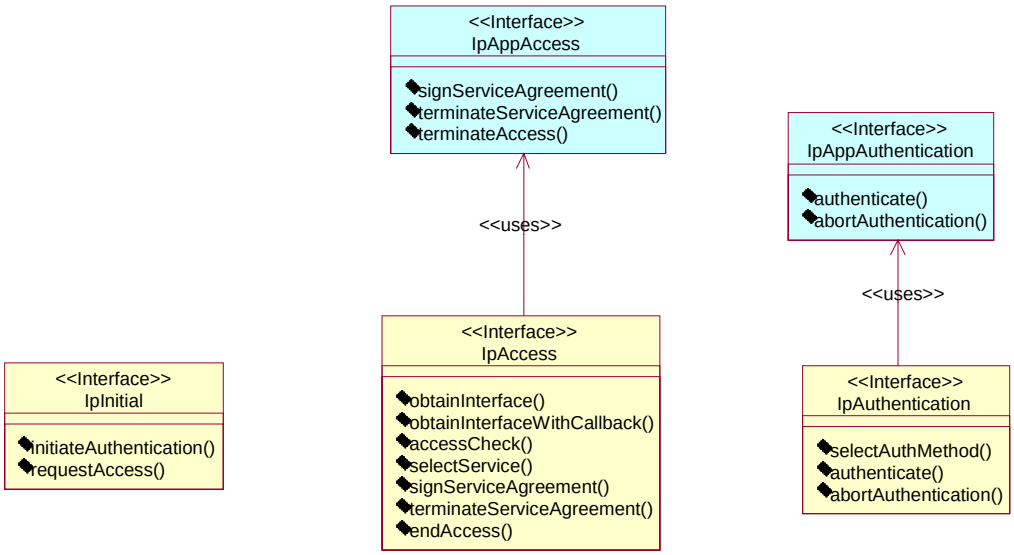


Figure 6-5: Trust and Security Management – Application and Framework Class Diagrams

6.2.3.1 IpInitial

<<Interface>> IpInitial
initiateAuthentication(appDomain: in TpAuthDomain , authType : in TpAuthType, fwDomain: out TpAuthDomainRef) : TpResult requestAccess(accessType: in TpAccessType, appAccessInterface; in IpOsaRef, fwAccessInterface: out IpOsaRefRef): TpResult

6.2.3.2 IpAppAuthentication

<<Interface>> IpAppAuthentication
authenticate(prescribedMethod: in TpAuthCapability, challenge: in TpString, response: out TpStringRef) : TpResult abortAuthentication() : TpResult

6.2.3.3 IpAuthentication

<<Interface>> IpAuthentication
selectAuthMethod (authCaps: in TpAuthCapabiltyList, prescribedMethod: out TpAuthCapabilityRef) : TpResult authenticate (prescribedMethod: in TpAuthCapability, challenge: in TpString, response: out TpStringRef) : TpResult abortAuthentication() : TpResult

6.2.3.4 IpAccess

<<Interface>> IpAccess
obtainInterface(interfaceName: in TpInterfaceName, fwInterface: out IpOsaRefRef): TpResult obtainInterfaceWithCallback(interfaceName: in TpInterfaceName, appInterface: in IpOsaRef, fwInterface: out IpOsaRefRef): TpResult accessCheck(serviceToken: in TpServiceToken,securityContext: in TpString, securityDomain: in TpString, group : in TpString, serviceAccessTypes: in TpString, 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

6.2.3.5 IpAppAccess

<<Interface>> IpAppAccess
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

6.2.4 Integrity Management

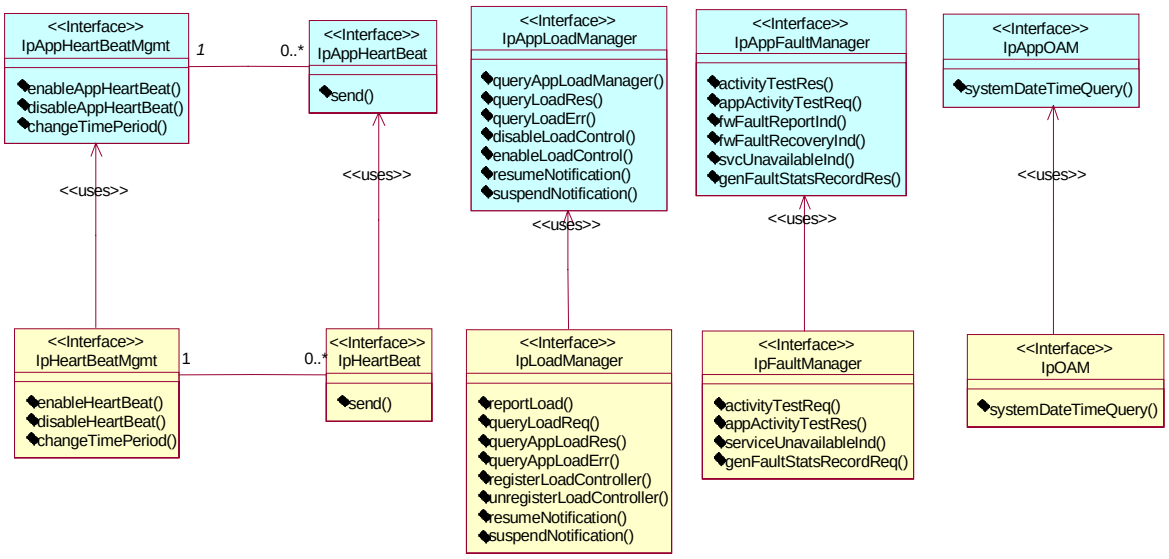


Figure 6-6: Integrity Management – Application and Framework Class Diagrams

6.2.4.1 IpHeartBeatMgmt

<<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

6.2.4.2 IpAppHeartBeatMgmt

<<Interface>> IpAppHeartBeatMgmt
enableAppHeartBeat(duration: in TpDuration, fwInterface: in IpHeartBeatRef, session: in TpSessionID) : TpResult disableAppHeartBeat(session: in TpSessionID) : TpResult changeTimePeriod(duration: TpDuration, session: in TpSessionID) : TpResult

6.2.4.3 IpHeartBeat

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

6.2.4.4 IpAppHeartBeat

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

6.2.4.5 IpLoadManager

<<Interface>> IpLoadManager
reportLoad(loadLevel : in TpLoadLevel) : TpResult queryLoadReq(serviceIDs: in TpServiceIDList, timeInterval : in TpTimeInterval) : TpResult queryAppLoadRes(loadStatistics : in TpLoadStatisticList) : TpResult queryAppLoadErr(loadStatisticsError : in TpLoadStatisticError) : TpResult registerLoadController(serviceIDs: in TpServiceIDList) : TpResult unregisterLoadController(serviceIDs: in TpServiceIDList) : TpResult resumeNotification(serviceIDs: in TpServiceIDList) : TpResult suspendNotification(serviceIDs: in TpServiceIDList) : TpResult

6.2.4.6 IpAppLoadManager

<<Interface>> IpAppLoadManager
queryAppLoadReq(serviceIDs: in TpServiceIdList, timeInterval : TpTimeInterval) : TpResult queryLoadRes(loadStatistics : in TpLoadStatisticList) : TpResult queryLoadErr(loadStatisticsError : in TpLoadStatisticError) : TpResult disableLoadControl(serviceIDs: in TpServiceIdList) : TpResult enableLoadControl(loadStatistics : in TpLoadStatisticList) : TpResult resumeNotification() : TpResult suspendNotification() : TpResult

6.2.4.7 IpFaultManager

<<Interface>> IpFaultManager
activityTestReq(activityTestID: in TpActivityTestID, svcID: in TpServiceID): TpResult appActivityTestRes(activityTestID: in TpActivityTestID, activityTestResult: in TpActivityTestRes): TpResult svcUnavailableInd(serviceID: in TpServiceID): TpResult genFaultStatsRecordReq(timePeriod: in TpTimeInterval, serviceIDs: in TpServiceIDList): TpResult

6.2.4.8 IpAppFaultManager

<<Interface>> IpAppFaultManager
activityTestRes(activityTestID: in TpActivityTestID, activityTestResult: in TpActivityTestRes): TpResult appActivityTestReq(activityTestID: in TpActivityTestID): TpResult fwFaultReportInd(fault: in TpInterfaceFault): TpResult fwFaultRecoveryInd(fault: in TpInterfaceFault): TpResult fwUnavailableInd(reason: in TpFwUnavailReason): TpResult svcUnavailableInd(serviceID: in TpServiceID, reason: in TpSvcUnavailReason): TpResult genFaultStatsRecordRes(faultStatistics: in TpFaultStatsRecord, serviceIDs: in TpServiceIDList): TpResult

6.2.4.9 IpOAM

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

6.2.4.10 IpAppOAM

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

6.2.5 Service Registration

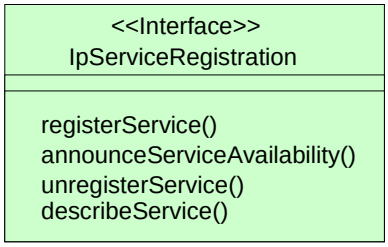


Figure 6-7: Service Registration Class Diagram

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

6.2.6 Service Factory

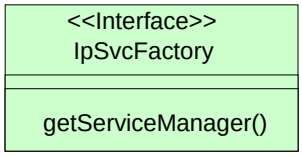


Figure 6-8: Service Factory Class Diagram

<<Interface>> IpSvcFactory
getServiceManager(application : in TpDomainID, serviceProperties : in TpServicePropertyList, serviceManager : out IpServiceRefRef) : TpResult

6.3 Generic Call Control

The Generic Call Control SCF provides the basic call control capabilities for the API. It allows calls to be instantiated from the network and routed through the network. The call model is based around a central call model that has zero to two call legs that are active (i.e., being routed or connected), each of which represents the logical relationship between the call and an address. However, the application does not have direct access to the call legs. Generic Call Control supports functionality to allow call routing and call management for Camel Phase 3 and earlier services.

Generic Call Control is represented by the IpCallManager and IpCall interfaces that interface to services provided by the network. Some methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppCallManager and IpAppCall.

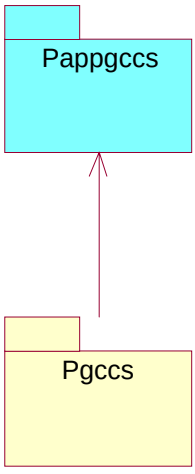


Figure 6-9: Generic Call Control Packages

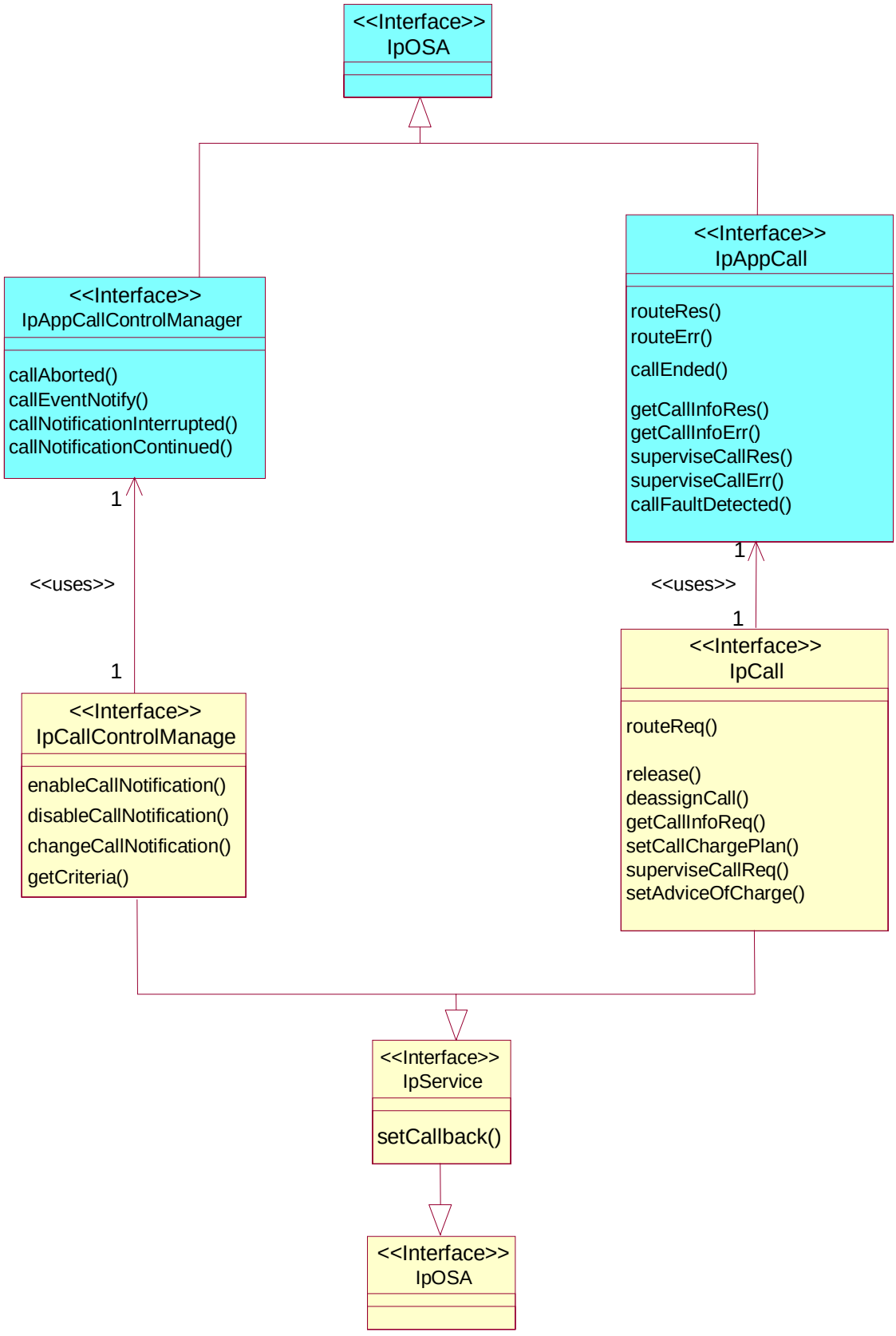


Figure 6-10: Generic Call Control Class diagram Interface Classes

This section contains the detailed interface specifications of the interfaces shown in the Generic Call Control Class diagram.

6.3.1 Interface Classes

6.3.1.1 IpAppCallControlManager

<<Interface>> IpAppCallControlManager
callAborted(callReference : in TpSessionID) : TpResult callEventNotify(callReference : in TpCallIdentifier , eventInfo : in TpCallEventInfo , assignmentID : in TpAssignmentID , appInterface : out IpAppCallRefRef) : TpResult callNotificationInterrupted() : TpResult callNotificationContinued(): TpResult

6.3.1.2 IpCallControlManager

<<Interface>> IpCallControlManager
enableCallNotification(appInterface : in IpAppCallControlManagerRef , eventCriteria : in TpCallEventCriteria , assignmentID : out TpAssignmentIDRef) : TpResult disableCallNotification(assignmentID : in TpAssignmentID) : TpResult changeCallNotification(assignmentID : in TpAssignmentID , eventCriteria : in TpCallEventCriteria) : TpResult getCriteria(eventCriteria : out TpCallEventCriteriaResultSet) : TpResult

6.3.1.3 IpAppCall

<<Interface>> IpAppCall
routeRes(callSessionID : in TpSessionID , eventReport : in TpCallReport, callLegSessionID : in TpSessionID) : TpResult routeErr(callSessionID : in TpSessionID , errorIndication : in TpCallError, callLegSessionID : in TpSessionID) : TpResult getCallInfoRes(callSessionID : in TpSessionID , callInfoReport : in TpCallInfoReport) : TpResult getCallInfoErr(callSessionID : in TpSessionID , errorIndication : in TpCallError) : TpResult superviseCallRes(callSessionID : in TpSessionID , report : in TpCallSuperviseReport , usedTime : in TpDuration) : TpResult superviseCallErr(callSessionID : in TpSessionID , errorIndication : in TpCallError) : TpResult callFaultDetected(callSessionID : in TpSessionID , fault : in TpCallFault) : TpResult callEnded(callSessionID : in TpSessionID , report : in TpCallEndedReport) : TpResult

6.3.1.4 IpCall

<<Interface>> IpCall
routeReq(callSessionID : in TpSessionID , responseRequested : in TpCallReportRequestSet , targetAddress : in TpAddress , originatingAddress : in TpAddress , originalDestinationAddress : in TpAddress , redirectingAddress : in TpAddress , appInfo : in TpCallAppInfoSet , callLegSessionID : out TpSessionIDRef) : TpResult release(callSessionID : in TpSessionID , cause : in TpCallReleaseCause) : TpResult deassignCall(callSessionID : in TpSessionID) : TpResult getCallInfoReq(callSessionID : in TpSessionID , callInfoRequested : in TpCallInfoType) : TpResult setCallChargePlan(callSessionID : in TpSessionID , callChargePlan : in TpCallChargePlan) : TpResult superviseCallReq(callSessionID : in TpSessionID , time : in TpDuration , treatment : in TpCallSuperviseTreatment) : TpResult setAdviceOfCharge(callSessionID : in TpSessionID , aOCInfo : in TpAoCInfo , tariffSwitch : in TpDuration) : TpResult

6.4 Generic User Interaction and Call User Interaction

The Generic User Interaction interface and Call User Interaction SCFs are used by applications to interact with end users.

The GUIS is represented by the `IpUIManager`, `IpUI and IpUICall` interfaces that interface to service capabilities provided by the network.

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.

To handle responses and reports, the developer must implement `IpAppUIManager, IpAppUI` and `IpAppUICall` interfaces to provide the callback mechanism.

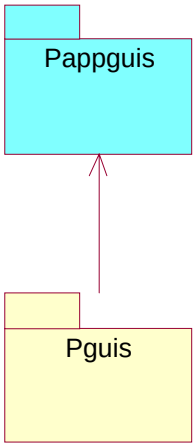


Figure 6-11: Generic User Interaction Packages

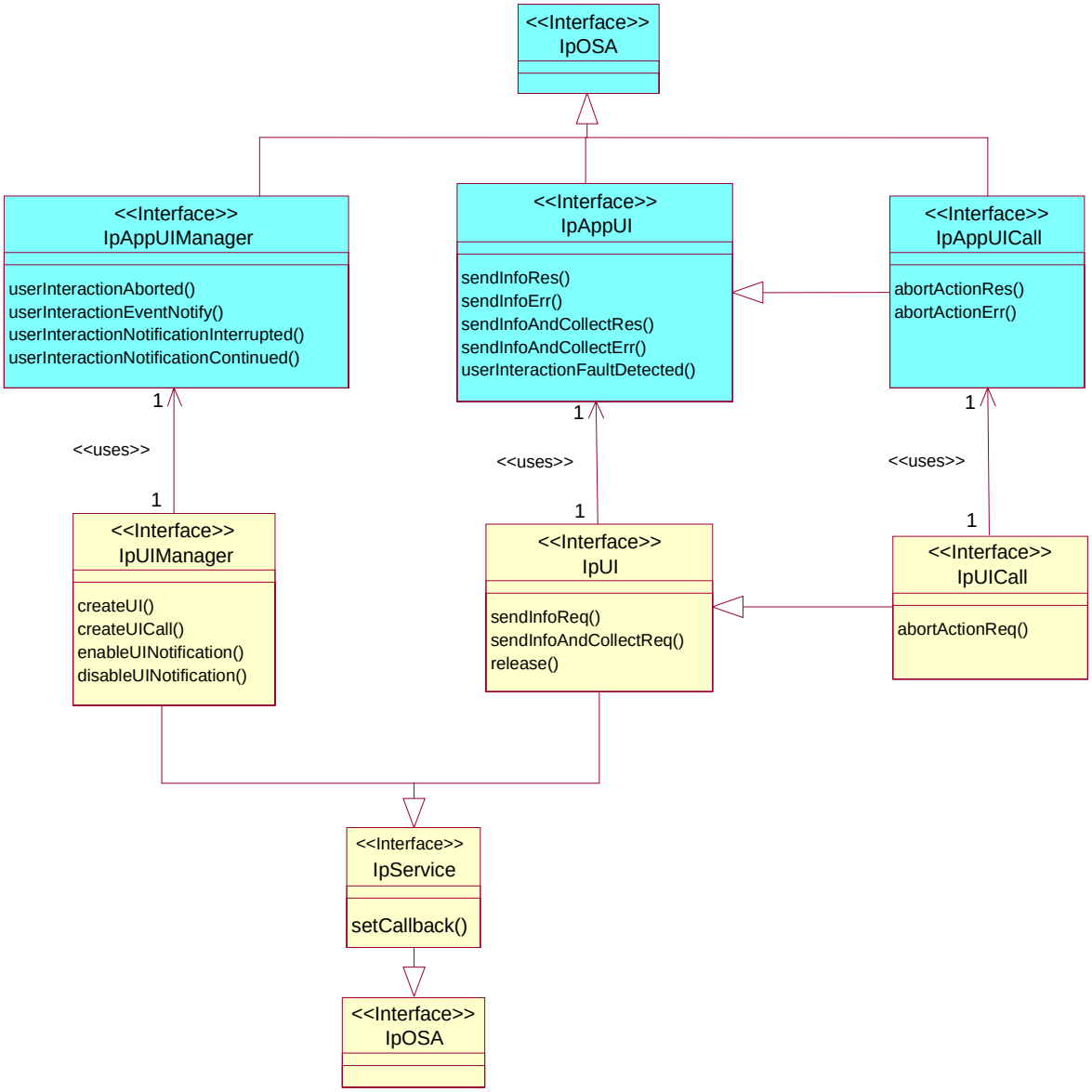


Figure 6-12: Generic User interaction Class diagram

6.4.1 Relation between IpCall and IpUICall during call related user interaction

For call related user interaction, the IpUICall Interface provides functions to send information to, or gather information from the user (or call party) attached to a call. This means that there is a relationship between a specific Call object and a UICall object. This is shown in the figure below.

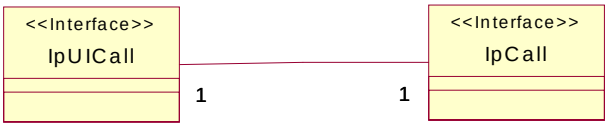


Figure 6-13: Relation between the UICall and the Call object.

In case a call requires user interaction, the application requests the UIManager to create the UICall object and provides a reference to the specific Call object. In this way the gateway is able to link the two objects together. It depends on the actual state of the call whether user interaction is really allowed.

6.4.2 Interface Classes

This section contains the detailed interface specifications of the interfaces shown in the Generic User Interaction Class diagram.

6.4.2.1 IpAppUIManager

<<Interface>> IpAppUIManager
userInteractionAborted(userInteraction : in TpUIIdentifier) : TpResult userInteractionEventNotify(ui : in TpUIIdentifier , eventInfo : in TpUIEventInfo , assignmentID : in TpAssignmentID , appInterface : out IpAppUIRefRef) : TpResult userInteractionNotificationInterrupted(): TpResult userInteractionNotificationContinued(): TpResult

6.4.2.2 IpUIManager

<<Interface>> IpUIManager
createUI(appUI : in IpAppUIRef , userAddress : in TpAddress , userInteraction : out TpUIIdentifierRef) : TpResult createUICall(appUI : in IpAppUICallRef , callIdentifier : in TpCallIdentifier , callLegIdentifier : in TpCallLegIdentifier , userInteraction : out TpUICallIdentifierRef) : TpResult enableUINotification(appInterface : in IpAppUIManagerRef , eventCriteria : in TpUIEventCriteria , assignmentID : out TpAssignmentIDRef) : TpResult disableUINotification(assignmentID : in TpAssignmentID) : TpResult

6.4.2.3 IpAppUI

<<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 , info : in TpString) : TpResult sendInfoAndCollectErr(userInteractionSessionID : in TpSessionID , assignmentID : in TpAssignmentID, error : in TpUIError) : TpResult userInteractionFaultDetected(userInteractionSessionID : in TpSessionID , fault : in TpUIFault) : TpResult

6.4.2.4 IpUI

<<Interface>> IpUI
sendInfoReq(userInteractionSessionID : in TpSessionID , info : in TpUIInfo , variableInfo : in TpUIVariableInfoSet , repeatIndicator : in TpInt32 , responseRequested : in TpUIResponseRequest , assignmentID : out TpAssignmentIDRef) : TpResult sendInfoAndCollectReq(userInteractionSessionID : in TpSessionID , info : in TpUIInfo , variableInfo : in TpUIVariableInfoSet , criteria : in TpUICollectCriteria , responseRequested: in TpUIResponseRequest , assignmentID : out TpAssignmentIDRef) : TpResult release(userInteractionSessionID : in TpSessionID) : TpResult

6.4.2.5 IpAppUICall

<<Interface>> IpAppUICall
abortActionRes(userInteractionSessionID : in TpSessionID , assignmentID : in TpAssignmentID) : TpResult abortActionErr(userInteractionSessionID : in TpSessionID , assignmentID : in TpAssignmentID , error : in TpUIError) : TpResult

6.4.2.6 IpUICall

<<Interface>> IpUICall
abortActionReq(userInteractionSessionID : in TpSessionID, assignmentID : in TpAssignmentID) : TpResult

6.5 Data Session Control

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.

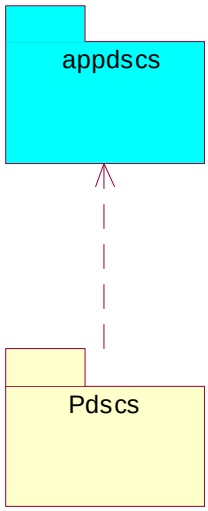


Figure 6-14: Data Session Control Packages

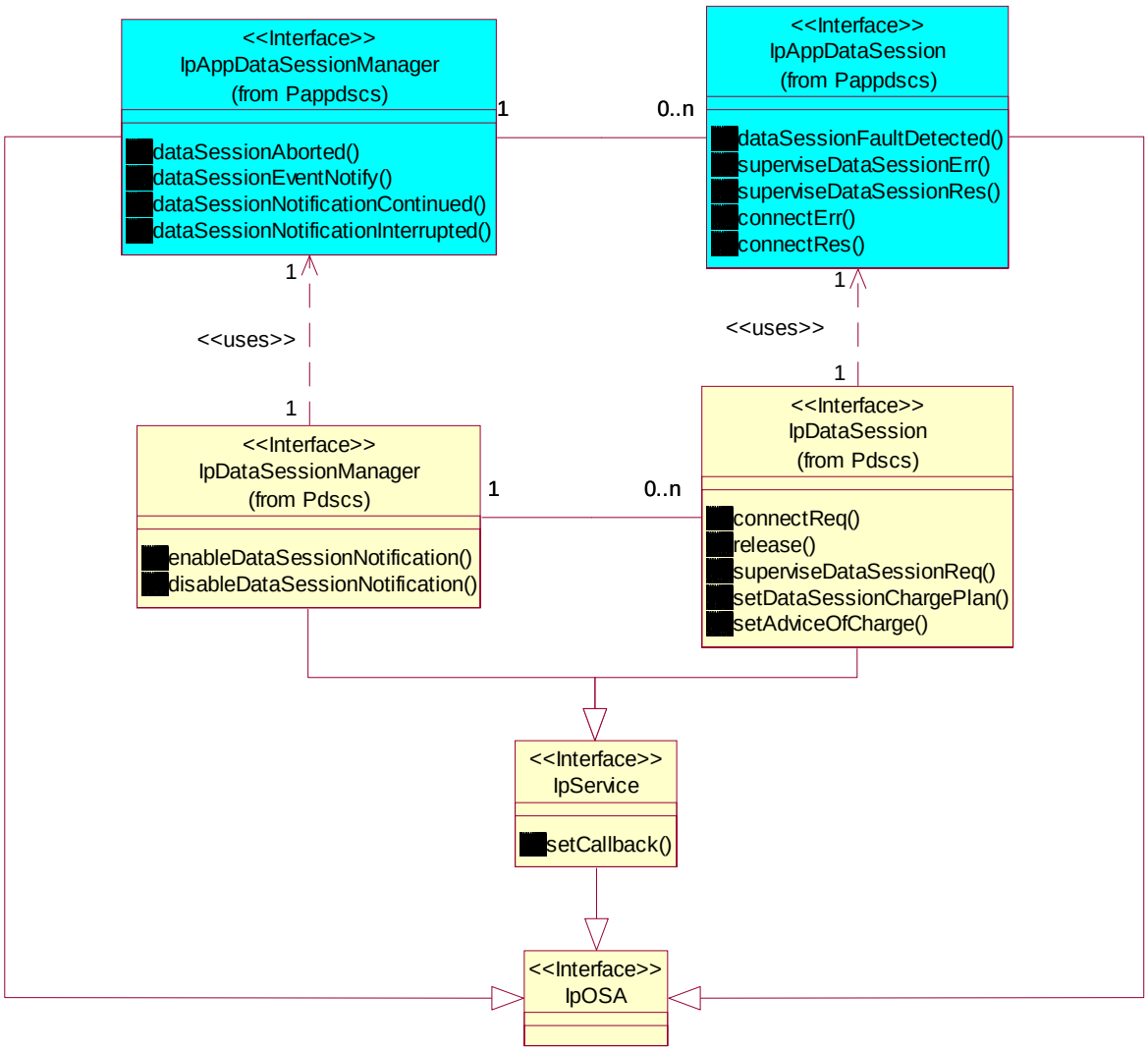


Figure 6-15: Data Session Control Class diagram Interface Classes

This section contains the detailed interface specifications of the interfaces shown in the Data Session Control Class diagram.

6.5.1 Interface Classes

6.5.1.1 IpAppDataSessionControlManager

<<Interface>> IpAppDataControlManager
dataSessionAborted(dataSessionID : in TpSessionID) : TpResult dataSessionEventNotify(dataSessionReference : in TpdataSessionIdentifier , eventInfo : in TpDataSessionEventInfo , assignmentID : in TpAssignmentID , appInterface : out IpAppdataSessionRefRef) : TpResult dataSessionNotificationContinued() : TpResult dataSessionNotificationInterrupted(): TpResult

6.5.1.2 IpDataSessionControlManager

<<Interface>> IpDataSessionControlManager
enableDataSessionNotification(appInterface : in IpAppDataSessionControlManagerRef , eventCriteria : in TpDataSessionEventCriteria , assignmentID : out TpAssignmentIDRef) : TpResult disableDataSessionNotification(assignmentID : in TpAssignmentID) : TpResult

6.5.1.3 IpAppDataSession

<<Interface>> IpAppDataSession
connectRes(dataSessionID : in TpSessionID , eventReport : in TpDataSessionEventReport, 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) : TpResult superviseDataSessionErr(dataSessionID : in TpSessionID , errorIndication : in TpDataSessionError) : TpResult dataSessionFaultDetected(dataSessionID : in TpSessionID , fault : in TpDataSessionFault) : TpResult

6.5.1.4 IpDataSession

<<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

6.6 Network User Location

The Network User Location (UL) SCF provides the `IpUserLocationCamel` interface, which provides methods for periodic and triggered location reporting. 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.

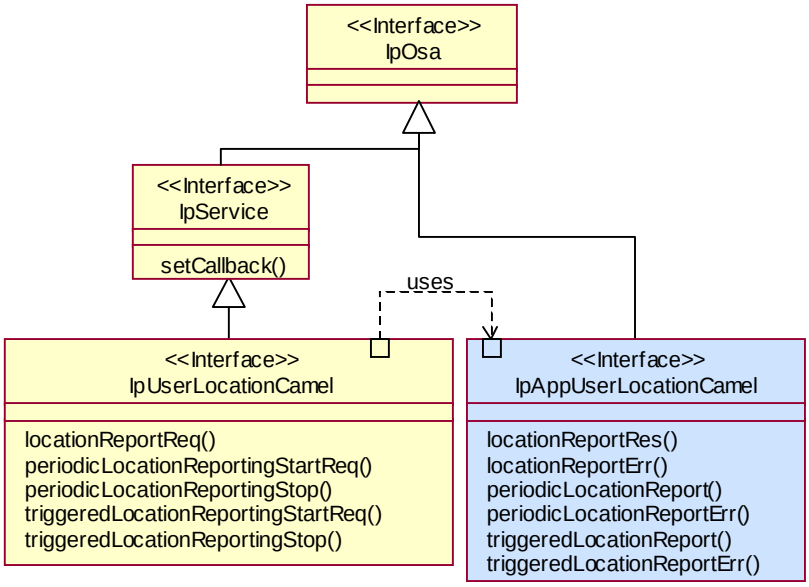


Figure 6-16: Network User Location class diagram.

6.6.1 Network User Location SCF interface

This interface is the 'SCF manager' interface for Network User Location.

<<Interface>> IpUserLocationCamel
locationReportReq(appLocationCamel : in IpAppUserLocationCamelRef, users : in TpAddressSet, assignmentId : out TpSessionIDRef) : TpResult periodicLocationReportingStartReq(appLocationCamel : in IpAppUserLocationCamelRef, users : in TpAddressSet, reportingInterval : in TpDuration, assignmentId : out TpSessionIDRef) : TpResult periodicLocationReportingStop(stopRequest : in TpMobilityStopAssignmentData) : TpResult triggeredLocationReportingStartReq(appLocationCamel : in IpAppUserLocationCamelRef, users : in TpAddressSet, trigger : in TpLocationTriggerCamel, assignmentId : out TpSessionIDRef) : TpResult triggeredLocationReportingStop(stopRequest : in TpMobilityStopAssignmentData) : TpResult

6.6.2 Network User Location application interface

The network user location application interface is implemented by the client application developer and is used to handle location reports that are specific for mobile telephony users.

<<Interface>> IpAppUserLocationCamel
locationReportRes(assignmentId : in TpSessionID, locations : in TpUserLocationCamelSet) : TpResult locationReportErr(assignmentId : in TpSessionID, cause : in TpMobilityError, diagnostic : in TpMobilityDiagnostic) periodicLocationReport(assignmentId : in TpSessionID, locations : in TpUserLocationCamelSet) : TpResult periodicLocationReportErr(assignmentId : in TpSessionID, cause : in TpMobilityError, diagnostic : in TpMobilityDiagnostic) triggeredLocationReport(assignmentId : in TpSessionID, location : in TpUserLocationCamel, criterion : in TpLocationTriggerCamel) : TpResult triggeredLocationReportErr(assignmentId : in TpSessionID, cause : in TpMobilityError, diagnostic : in TpMobilityDiagnostic)

6.7 User Status

The User Status (US) SCF 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.

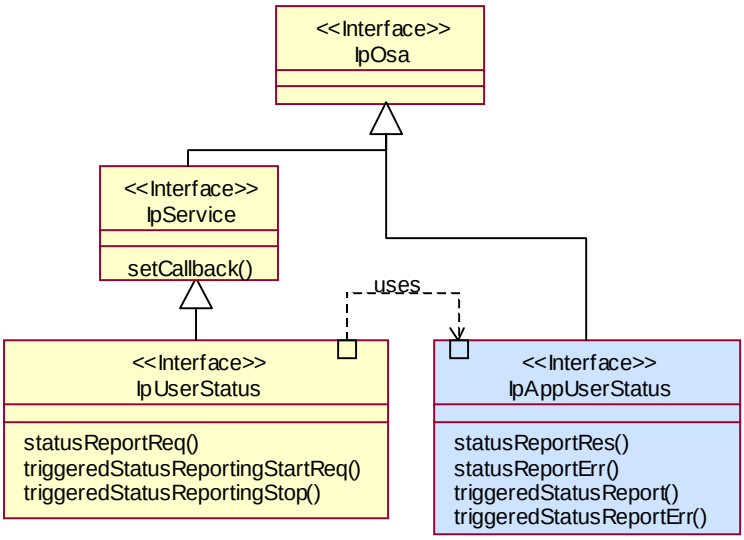


Figure 6-17: User Status class diagram.

6.7.1 User Status SCF interface

The user status interface represents the interface to the user status service capability feature.

<<Interface>> IpUserStatus
statusReportReq(appStatus : in IpAppUserStatusRef, users : in TpAddressSet, assignmentId : out TpSessionIDRef) : TpResult triggeredStatusReportingStartReq (appStatus : in IpAppUserStatusRef, users : in TpAddressSet, assignmentId : out TpSessionIDRef) : TpResult triggeredStatusReportingStop (stopRequest : in TpMobilityStopAssignmentData) : TpResult

6.7.2 User Status application interface

The user-status application interface is implemented by the client application developer and is used to handle user status reports.

<<Interface>> IpAppUserStatus
statusReportRes(assignmentId : in TpSessionID, status : in TpUserStatusSet) : TpResult statusReportErr(assignmentId : in TpSessionID, cause : in TpMobilityError, diagnostic : in TpMobilityDiagnostic) triggeredStatusReport(assignmentId : in TpSessionID, status : in TpUserStatus) : TpResult triggeredStatusReportErr(assignmentId : in TpSessionID, cause : in TpMobilityError, diagnostic : in TpMobilityDiagnostic)

6.8 Terminal Capabilities

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`.

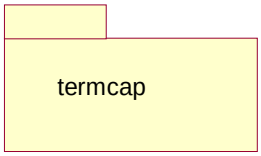


Figure 6-18: Terminal Capabilities package

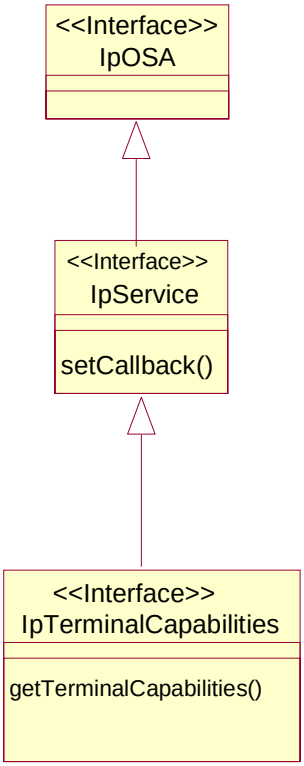
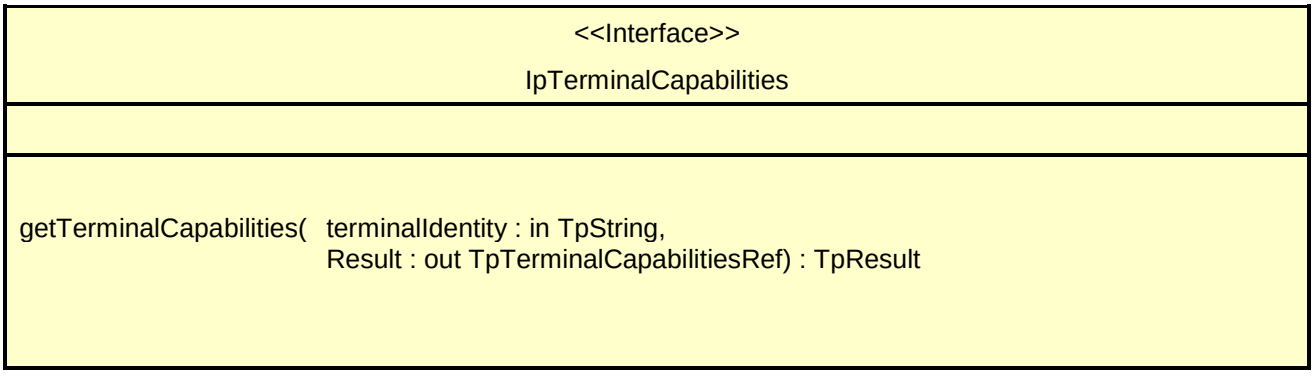


Figure 6-19: Terminal Capabilities class diagrams

6.8.1 Terminal Capabilities SCF interface

The Terminal Capabilities SCF interface `IpTerminalCapabilities` contains the synchronous method `getTerminalCapabilities`. The application has to provide the `terminalIdentity` is 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).



7 State Transition Diagrams

This section contains the State Transition Diagrams for the objects that implement the 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 the P_TASK_REFUSED 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.

7.1 Framework

7.1.1 IpAuthentication

.

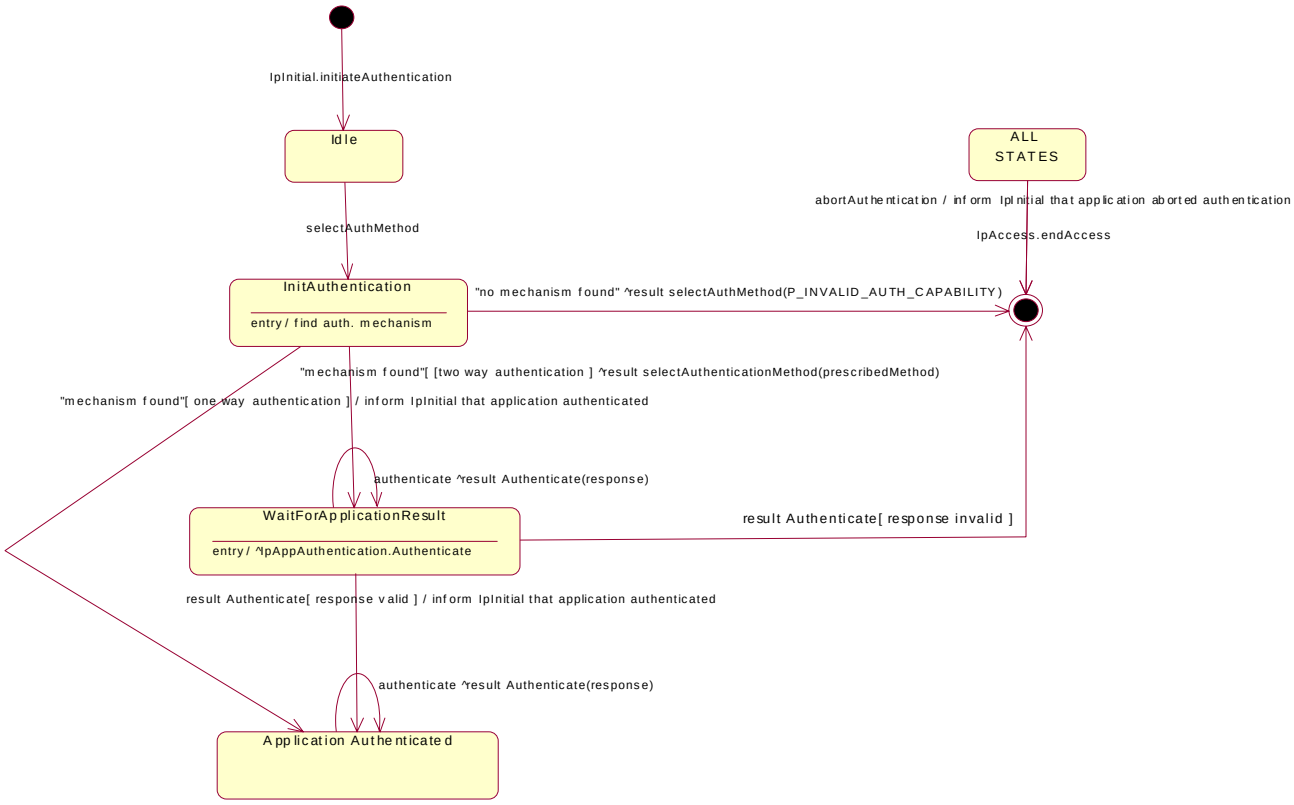


Figure 7-1: State Transition Diagram for Authentication

7.1.1.1 Idle state

When the application has requested the IpInitial interface for initiateAuthentication, an object implementing the IpAuthentication interface is created. The application now has to provide it's authentication capabilities by invoking the SelectAuthMethod method.

7.1.1.2 Init Authentication 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.

7.1.1.3 Wait For Application Result 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 IpAuthentication 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.

7.1.1.4 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 IpAuthentication interface, the Framework provides the correct response to the challenge of the application.

7.1.2 IpAccess

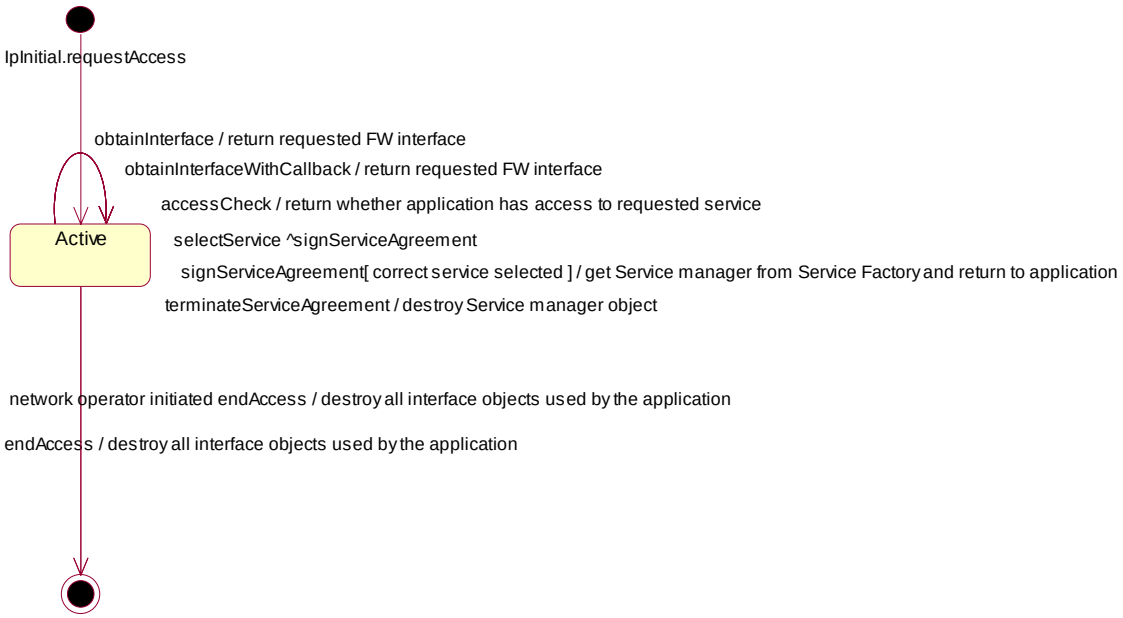


Figure 7-2: State Transition Diagram for Access

7.1.2.1 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.

7.1.3 IpServiceDiscovery

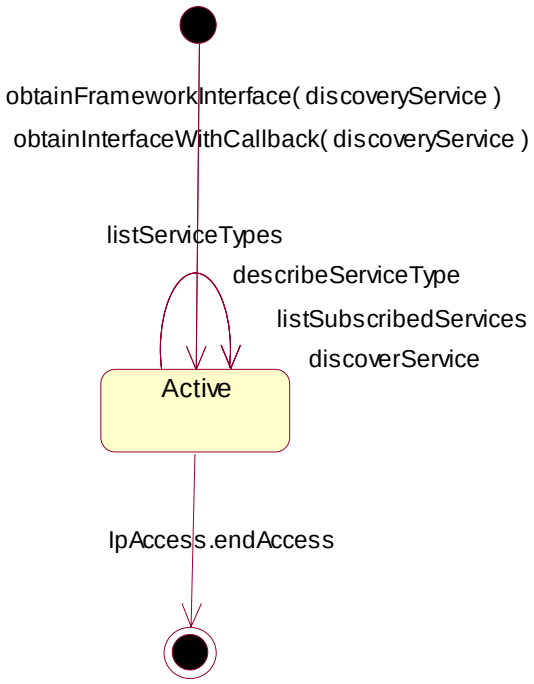


Figure 7-3: State Transition Diagram for Service Discovery

7.1.3.1 Active state

When the application requests for the Service Discovery SCF 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.

7.1.4 IpLoadManager

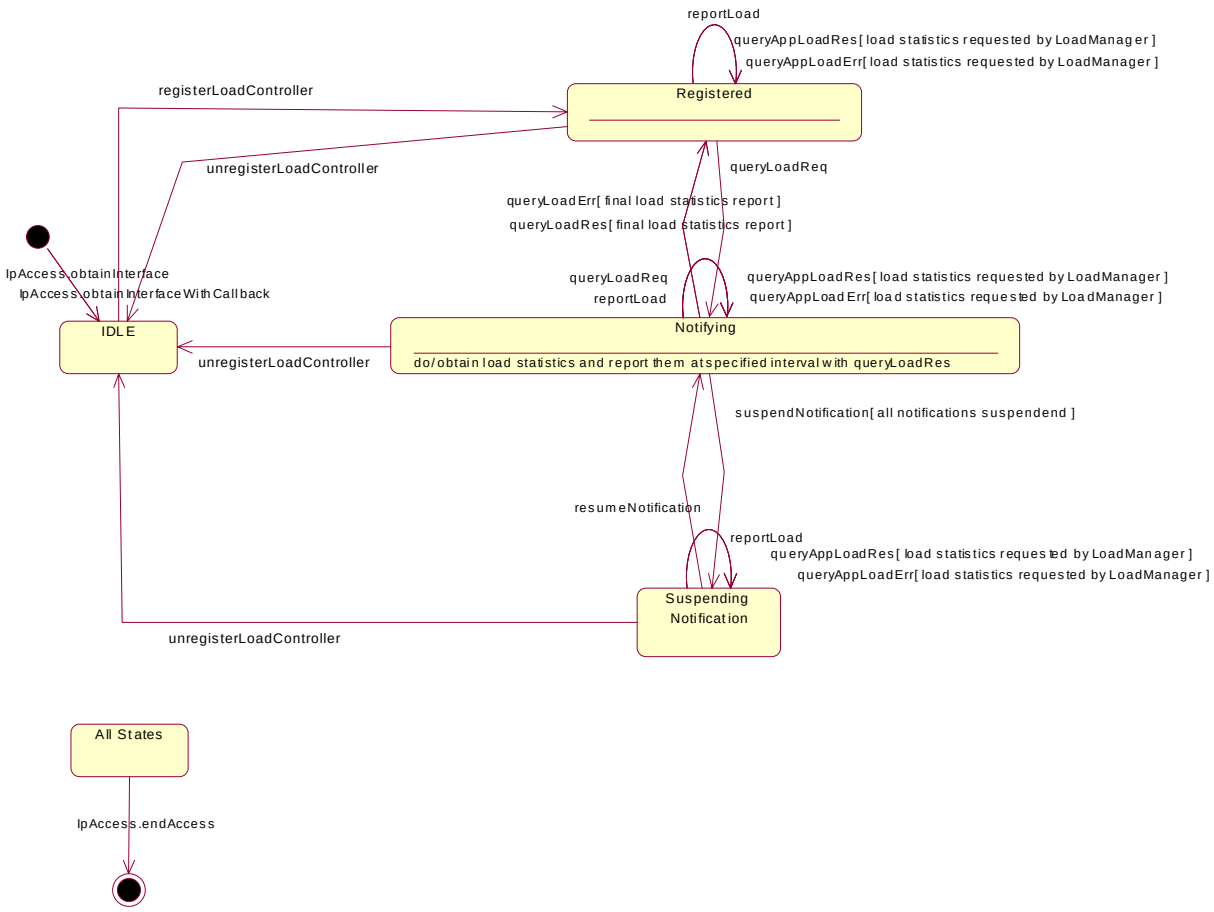


Figure 7-4: State Transition Diagram for LoadManager

7.1.4.1 Idle State

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

7.1.4.2 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 enableLoadControl() 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 .

7.1.4.3 Notifying

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

7.1.4.4 Suspending Notification

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

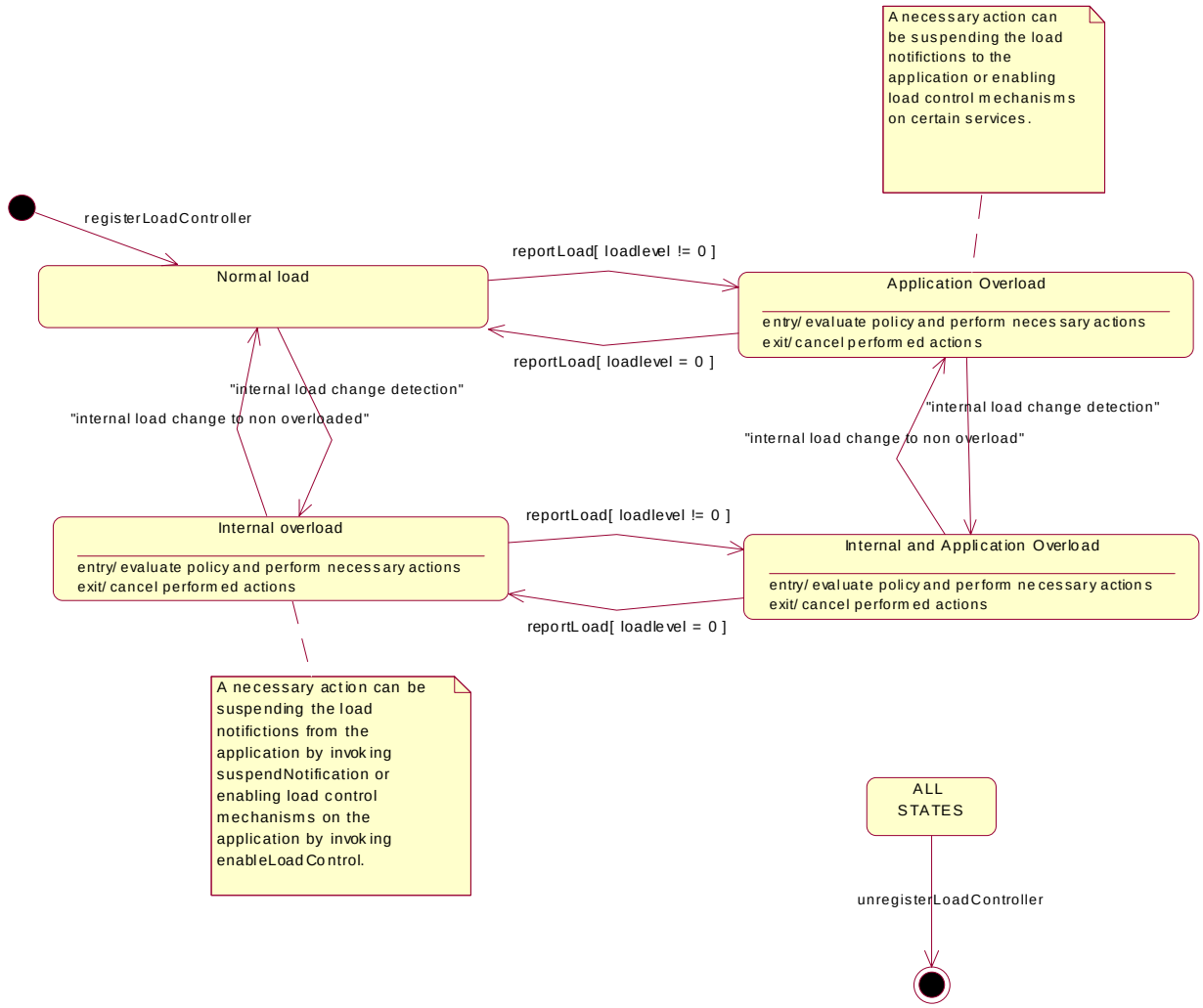


Figure 7-5: State Transition Diagram for the LoadManagerInternal

7.1.4.5 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.

7.1.4.6 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.

7.1.4.7 Internal overload

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.

7.1.4.8 Internal and application overload

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.

7.1.5 IPFaultManager

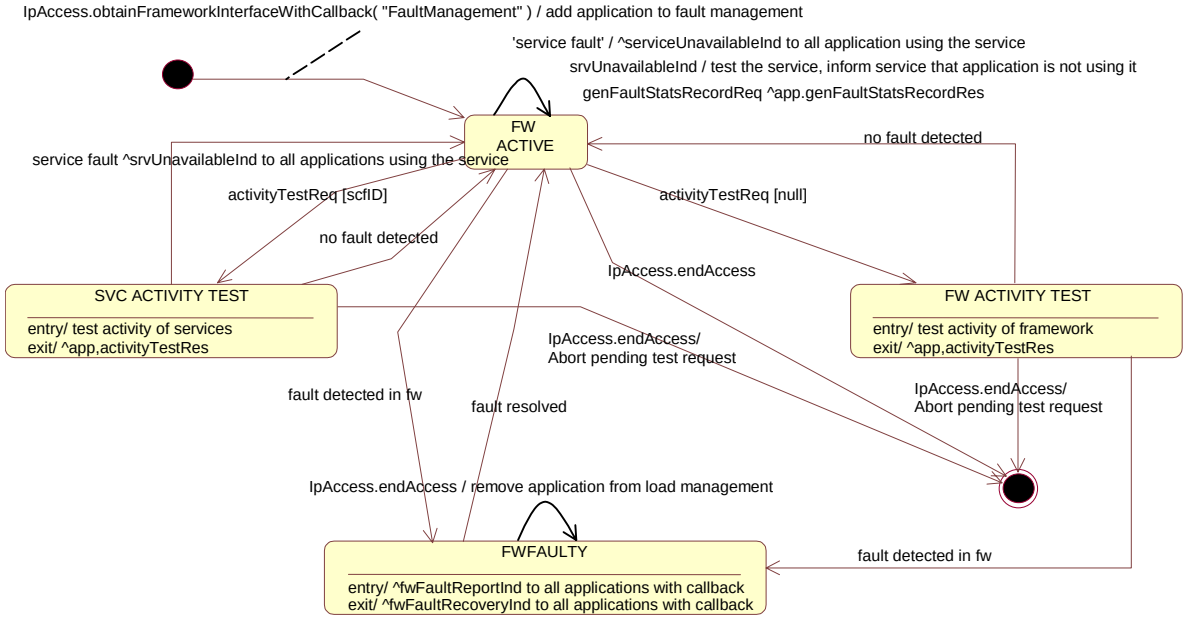


Figure 7-6: State Transition Diagram for Fault Manager

7.1.5.1 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.

7.1.5.2 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 belongs to any SCFs of the framework returns an error. If the framework ever recover, application with fault management callbacks will be notified via a fwFaultRecoveryInd message.

7.1.5.3 The 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.

7.1.5.4 The 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.

7.1.6 IpHeartbeatmgmt

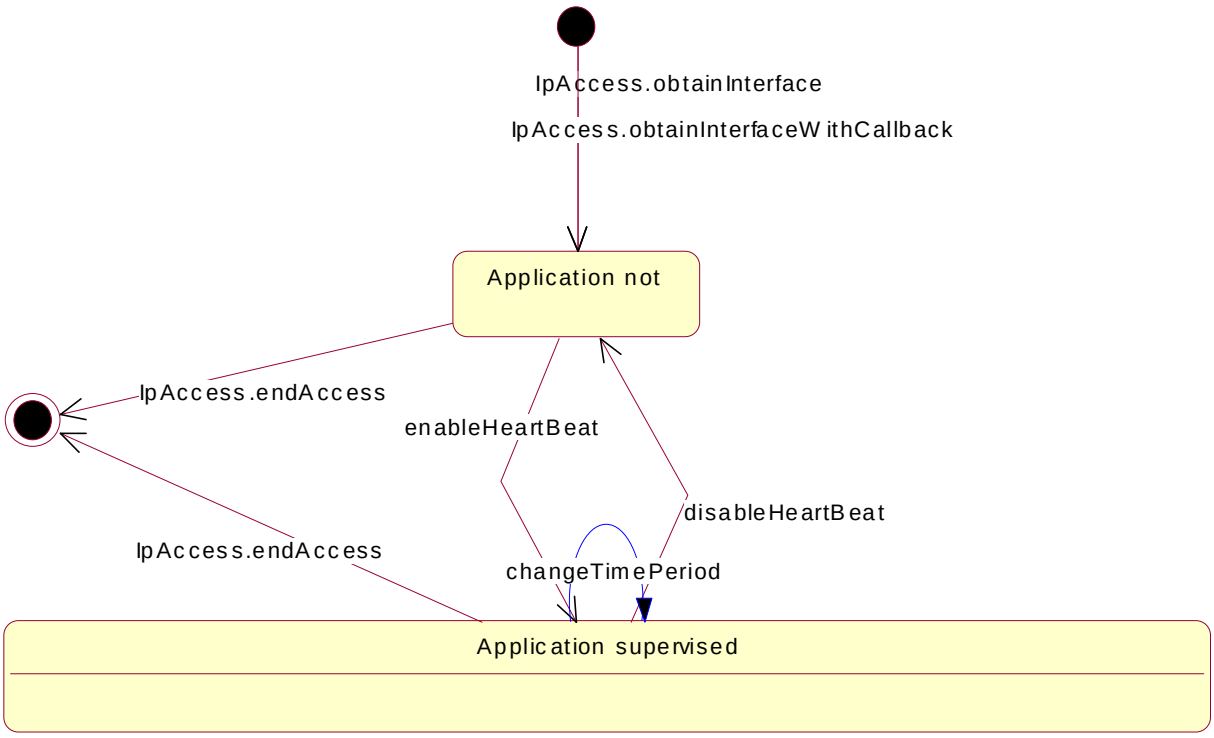


Figure 7-7: State Transition Diagram for the Heartbeat manager

7.1.6.1 Application not supervised

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

7.1.6.2 Application supervised

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.

7.1.7 IpHeartBeat

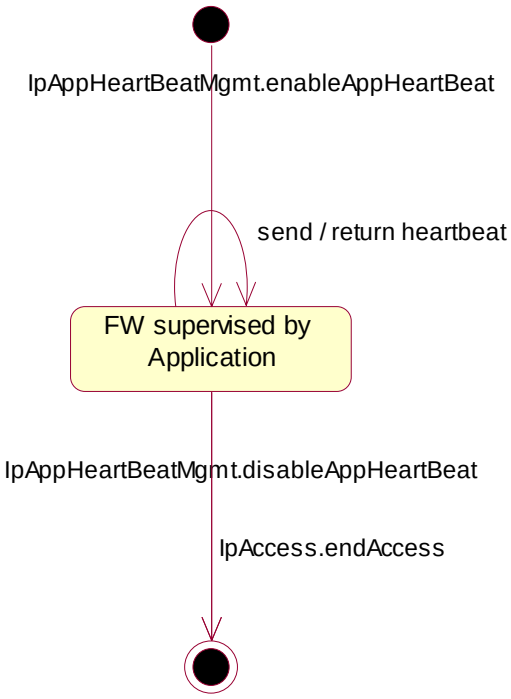


Figure 7-8: State Transition Diagram for HeartBeat

7.1.7.1 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.

7.1.8 IpOAM

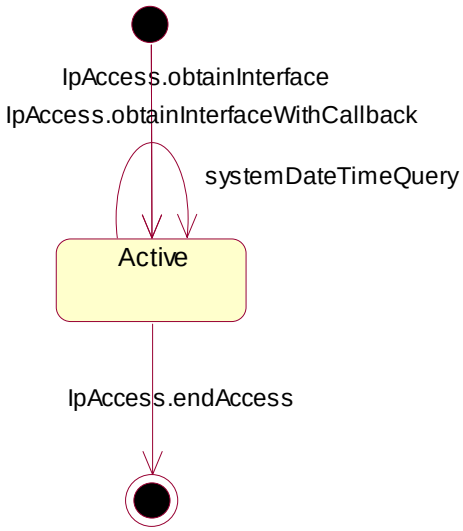


Figure 7-9: State Transition Diagram for OAM

7.1.8.1 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.

7.1.9. IpServiceRegistration

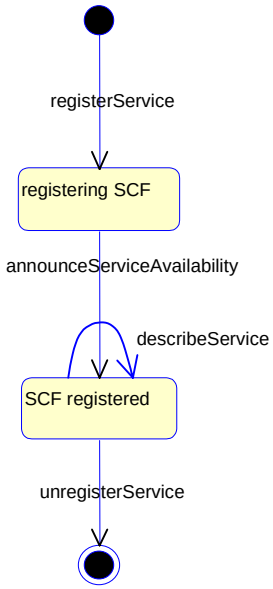


Figure 7-10: State Transition Diagram for Service Registration

7.1.9.1 Registering SCF

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

7.1.9.2 SCF Registered

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

7.2 Generic Call Control

7.2.1 Call Control Manager

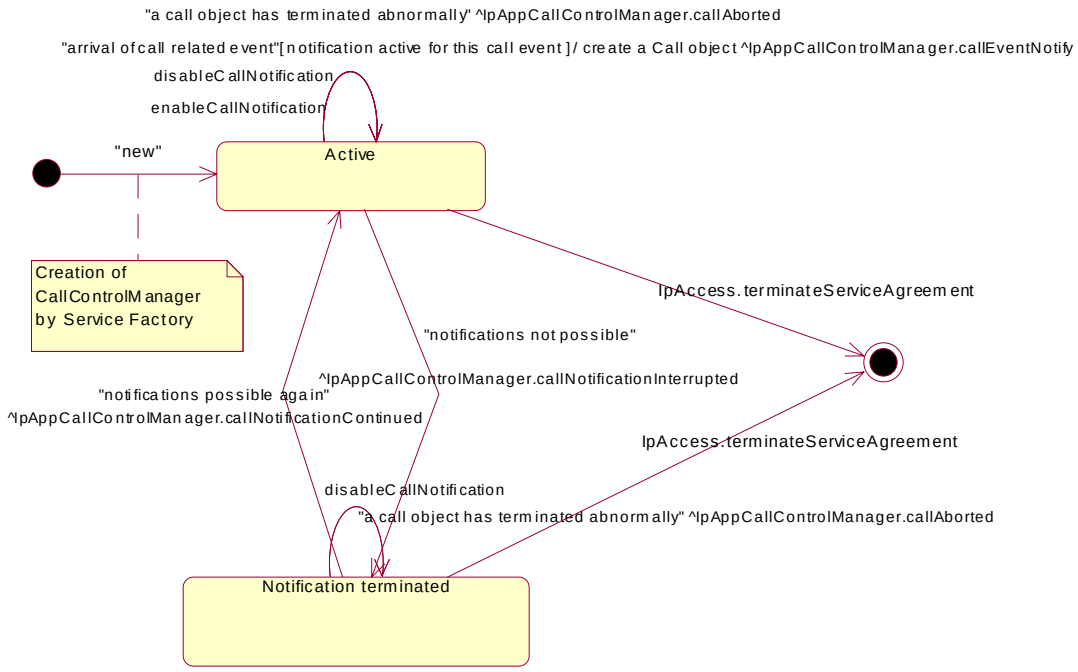


Figure 7-11: State Transition Diagram for the CallControlManager

7.2.1.1 Active state

In this state a relation between the Application and the Generic Call Control Service Capability Feature has been established. It allows the application to indicate that it is interested in call related events. In case such an event occurs, the Call Control Manager will create a Call object and inform the application by invoking the method callEventNotify() on the IpAppCallControlManager interface. The application can also indicate it is no longer interested in certain call related events by calling disableCallNotification().

7.2.1.2 Notification terminated state

When the Call Control manager is in the Notification terminated state, events requested with enableCallNotification() will not be forwarded to the application. There can be multiple reasons for this: for instance it might be that the application receives more notifications 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.

7.2.2 Call

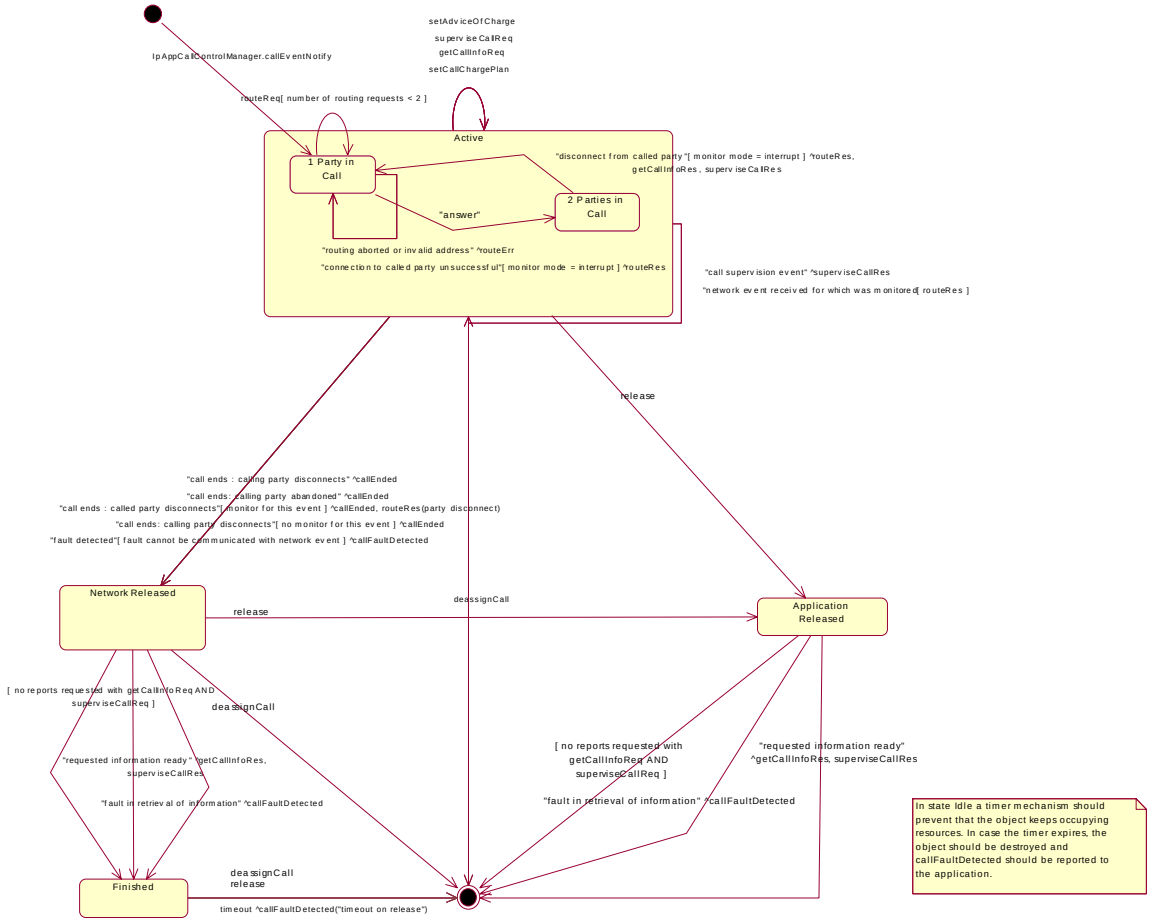


Figure 7-12: State Transition Diagram for Call

7.2.2.1 Active state

In this state a call between two parties is being setup or present. Refer to the substates for more details

The application can request the gateway for a certain type of charging of the call by calling setCallChargePlan(). The application can request for charging related information by calling getCallInfoReq(). Furthermore the application can request supervision of the call by calling superviseCallReq(). It is also allowed to send Advice Of Charge information by calling setAdviceOfCharge().

7.2.2.1.1 1 Party in Call state

When the Call is in this state a calling party is present. The application can now request that a connection to a called party be established by calling the method routeReq(). When the calling party abandons the call before the application has invoked the routeReq() operation, the gateway informs the application by invoking callFaultDetected() and also the operation callEnded() will be invoked. When the calling party abandons the call after the application has invoked routeReq() but before the call has actually been established, the gateway informs the application by invoking callEnded().

When the calling party answers the call, a transition will be made to the 2 Parties in Call state. In case the call can not be established because the application supplied an invalid address or the connection to the called party was unsuccessful while the application was monitoring for the latter in interrupt mode, the Call object will stay in this state

In this state user interaction is possible unless there is an outstanding routing request.

7.2.2.1.2 2 Parties in Call state

A connection between two parties has been established.

In case the calling party disconnects, the gateway informs the application by invoking callEnded().

When the called party disconnects different situations apply:

1. the application is monitoring for this event in interrupt mode: a transition is made to the 1 Party in Call state, the application is informed with routeRes with indication that the called party has disconnected and all requested reports are sent to the application. The application now again has control of the call.

2. the application is monitoring for this event but not in interrupt mode. In this case a transition is made to the Network Released state and the gateway informs the application by invoking the operation routeRes() and callEnded().

3. the application is not monitoring for this event. In this case the application is informed by the gateway invoking the callEnded() operation and a transition is made to the Network Released state.

7.2.2.3 Network released state

In this state the call has ended and the Gateway collects the possible call information requested with getCallInfoReq() and / or superviseCallReq(). The information will be returned to the application by invoking the methods getCallInfoRes() and / or superviseCallRes() on the application. Also when a call was unsuccessful these methods are used.In case the application has not requested additional call related information immediately a transition is made to state Idle.

7.2.2.4 Finished state

In this state the call has ended and no call related information is to be send to the application. The application can only release the Call object. Calling the deassingCall() method has the same effect. Note that the application has to release the object itself as good OO practice requires that when an object was created on behalf of a certain entity, this entity is also responsible for destroying it when the object is no longer needed.

7.2.2.5 Application released state.

In this state the application has requested to release the Call object and the Gateway collects the possible call information requested with getCallInfoReq(). In case the application has not requested additional call related information immediately the Call object is destroyed.

7.3 User Interaction

7.3.1 UI Manager

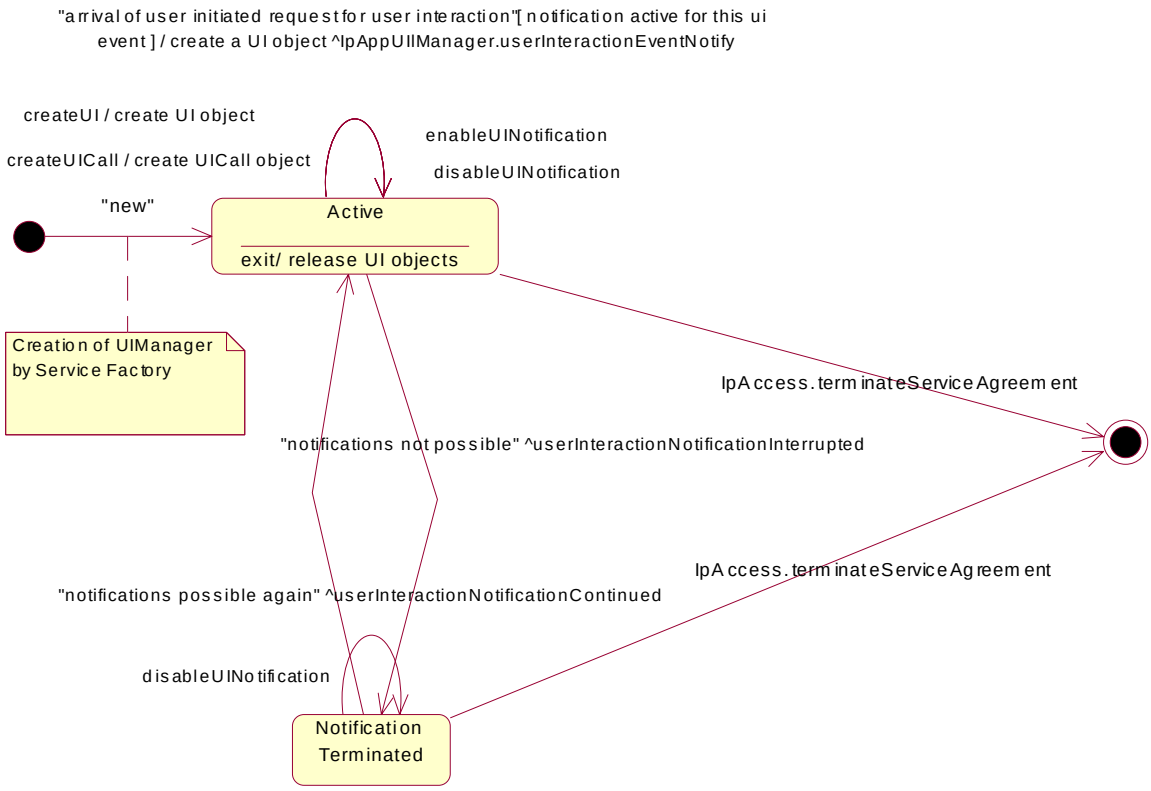


Figure 7-13: State Transition Diagram for the UIManager

7.3.1.1 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.

7.3.1.2. Notification Terminated state

When the UI manager is in the Notification terminated state, events requested with enableUINotification() 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.

7.3.2 UI

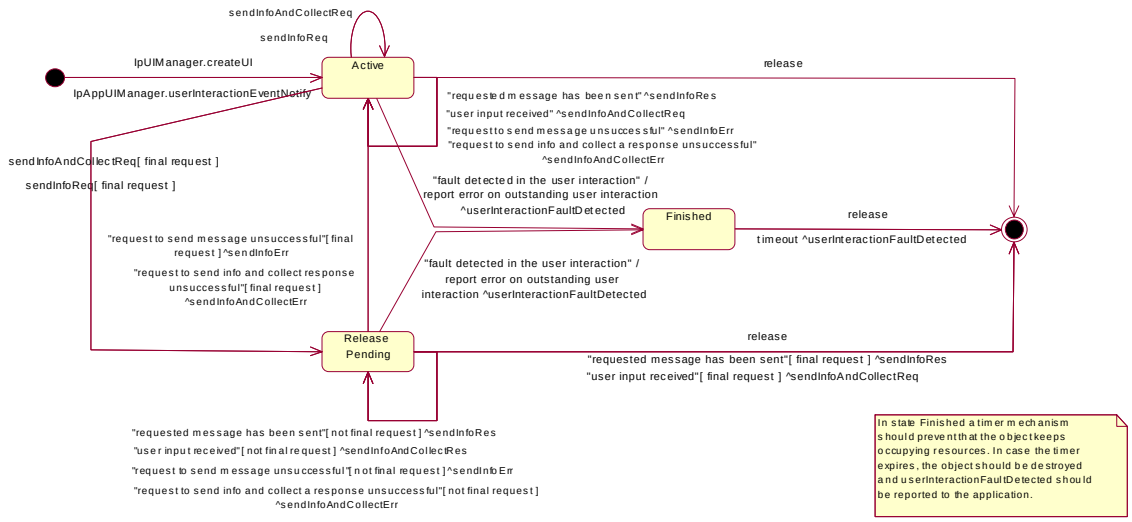


Figure 7-14: State Transition Diagram for UI

7.3.2.1 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.

7.3.2.2 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.

7.3.2.3 Finished

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

7.3.3 UI Call

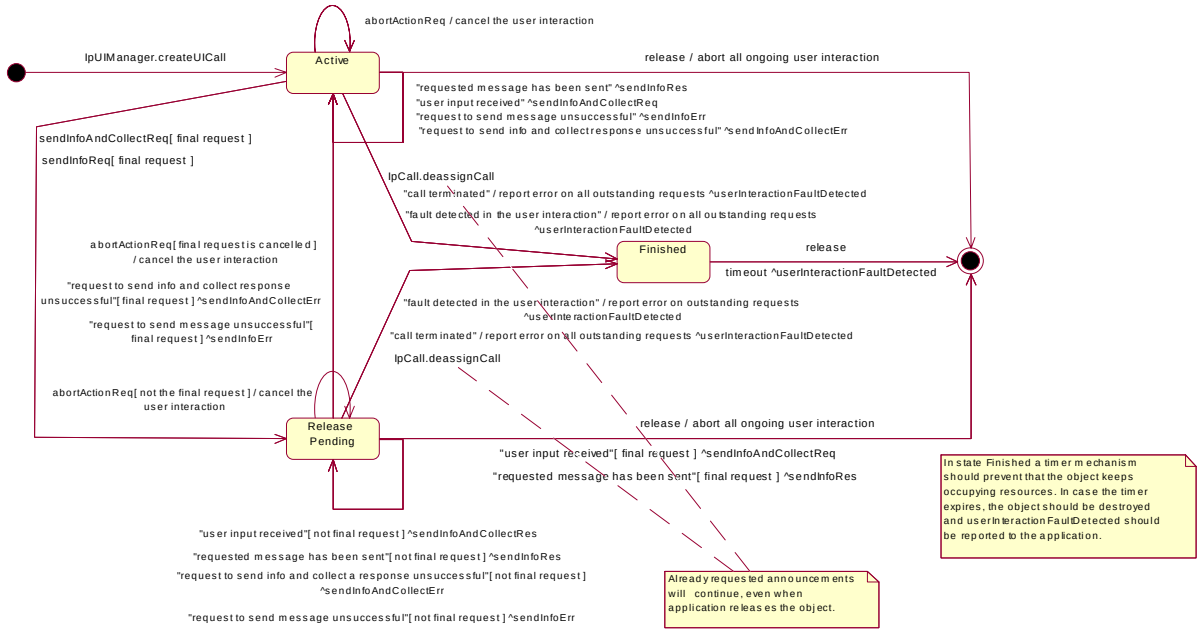


Figure 7-15: State Transition Diagram for UICall

7.3.3.1 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.

7.3.3.2 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.

7.3.3.3 Finished

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.

7.4 Data Session

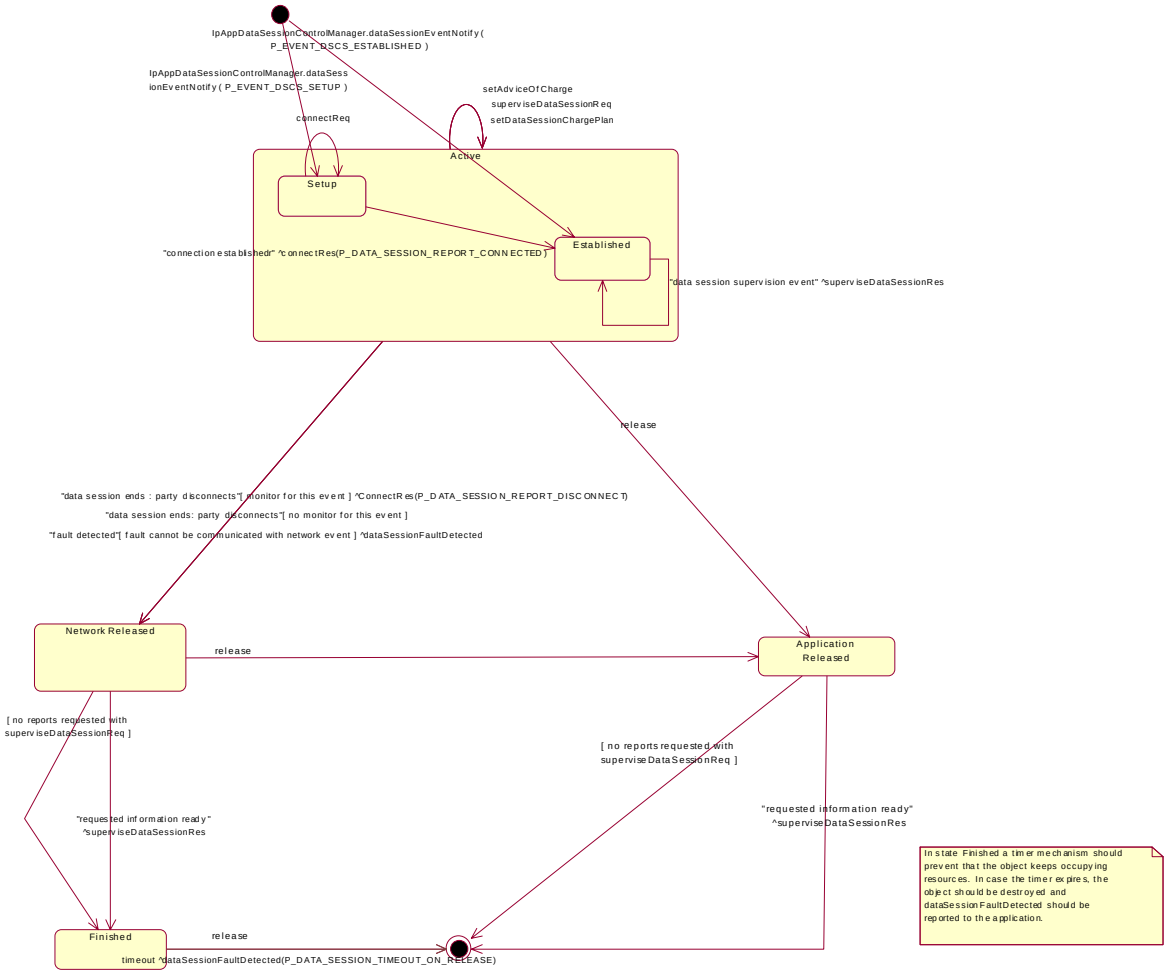


Figure 7-16: State Transition Diagram for Data Session

7.4.1 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().

7.4.1.1 Setup state

The Setup state is reached after a dataSessionEvebtNotify() 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.

7.4.1.2 Established state

In this state the data connection is established. If supervision has been requested the application expects the corresponding superviseDataSessionRes().

7.4.2 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().

7.4.3 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.

7.4.4 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.

7.5 Network User Location

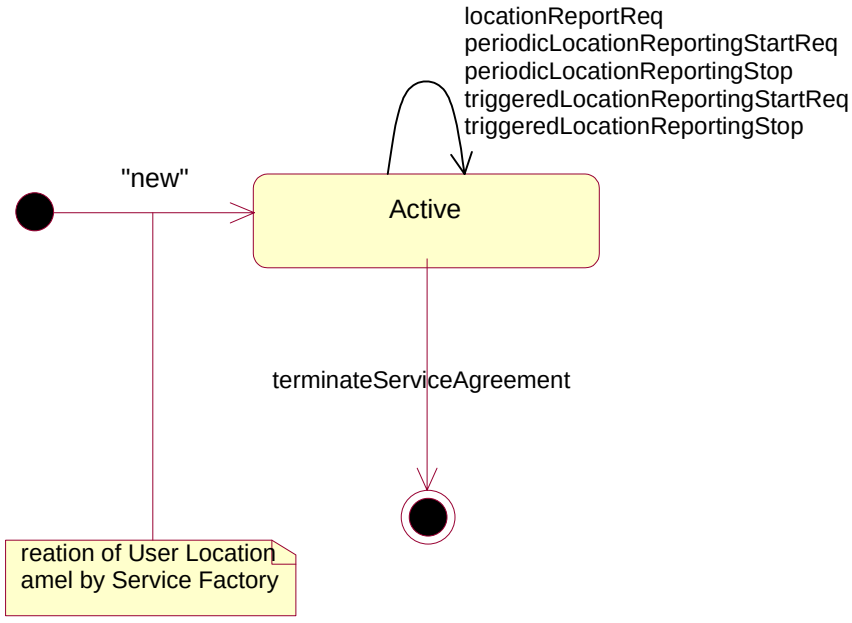


Figure 7-17: State Transition Diagram for Network User Location

During the signServiceAgreement a new user location interface reference is created, which is user as the initial point of contact for the application.

7.5.1 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.

7.6 User Status

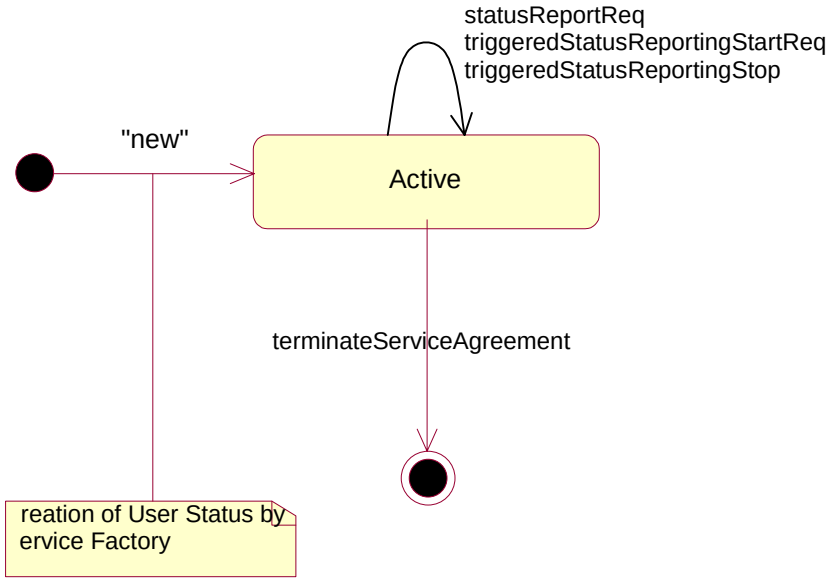


Figure 7-18: State Transition Diagram for User Status.

7.6.1 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.

8 Data Definitions

8.1 Common Data definitions

The constants and types defined in the following sections are defined in the *org.threegpp.osa* package.

8.1.1 Primitive Data Types

Type Name	Description
`TpBoolean`	Defines a Boolean data type.
`TpInt32`	Defines a signed 32 bit integer.
`TpFloat`	Defines a single precision float
`TpString`	Defines a string, comprising length and data.

8.1.2 Structured data types classification

Many different structured data types are used in OSA and a classification/clarification is required.

8.1.2.1 Structures made of data elements

This describes data types that can be considered as classes (in Java or C++) or structures (C++, IDL). The goal of these data types is to group pieces of information into a logical unit. *Example*: an TAddress data type may be defined in IDL as:

```
struct TpAddress {
```

```
        TpAddressPlan          Plan;
        TpString               AddrString;
        TpString               Name;
        TpAddressPresentation  Presentation;
        TpAddressScreening     Screening;
        TpString               SubAddressString;
    };
```

8.1.2.2 Tagged choice of data elements (i.e.: Free unions)

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

This data type can be implemented in IDL as a union with a switch statement for the *tag* part, and a set or case statements for the *union* part.

***Example*:** The TCallError data type may be defined in IDL as:

```
union TpCallError switch (TCallErrorType) {
  case CALL_ERROR_UNDEFINED:
    TpCallErrorInfoDefault                CallErrorUndefined;
  case CALL_ERROR_ROUTING_ABORTED:
    TpCallErrorInfoRoutingAborted         CallErrorRoutingAborted;
  case CALL_ERROR_CALL_ABANDONED:
    TpCallErrorInfoCallAbandoned          CallErrorCallAbandoned;
  case CALL_ERROR_INVALID_ADDRESS:
    TpCallErrorInfoInvalidAddress         CallErrorInvalidAddress;
  case CALL_ERROR_INVALID_STATE:
    TpCallErrorInfoDefault                CallErrorInvalidState;
  case CALL_ERROR_INVALID_CRITERIA:
    TpCallErrorInfoDefault                CallErrorInvalidCriteria;
};
```

8.1.2.3 Collection of data elements

This describes a data type, which comprises an ordered or unordered collection of data elements of the same type. The number of data elements in the collection is always know and can be implicit (IDL) or may appear as an integer inside a structure depending on the language used. This data type can be implemented in IDL as a sequence.

***Example*:**

```
typedef sequence<SessionID> SessionIDSet;
```

8.1.2.4 References

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

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

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

```
typedef TAddress *TAddressRef;
```

8.1.3 Interface Definitions

8.1.3.1 IpOsa

Defines the address of an IpOsa Interface.

8.1.3.2 IpOsaRef

Defines a Reference to type IpOsa

8.1.3.3 IpOsaRefRef

Defines a Reference to type IpOsaRef

8.1.3.4 IpService

Defines the address of an IpService Interface.

8.1.3.5 IpServiceRef

Defines a Reference to type IpService

8.1.3.6 IpServiceRefRef

Defines a Reference to type IpServiceRef

8.1.4 Non primitive and structured type types definition

8.1.4.1 TpAssignmentID

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

8.1.4.2 TpSessionID

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

The session ID is identical to a TpInt32 type.

8.1.4.3 TpSessionIDSet

Defines a collection of data elements of TpSessionID.

8.1.4.4 TpDuration

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

8.1.4.5 TpResult

Defines the structure of data elements that specifies the result of a method call.

Structure Member Name	Structure Member Type
ResultType	TpResultType
ResultFacility	TpResultFacility
ResultInfo	TpResultInfo

8.1.4.6 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

8.1.4.7 TpResultFacility

Defines the facility code of a result. In Release 99 of the OSA API, only P_RESULT_FACILITY_UNDEFINED must be used.

Name	Value	Description
P_RESULT_FACILITY_UNDEFINED	0	Undefined

8.1.4.8 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 capability feature ID
P_INVALID_EVENT_TYPE	0007h	Invalid event type
P_SERVICE_NOT_ENABLED	0008h	The service capability feature ID does not correspond to a SCF 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 required parameter has not been specified in the method call
P_RESOURCES_UNAVAILABLE	000Dh	The required resources in the network are not available
P_TASK_REFUSED	000Eh	The requested method has been refused
P_TASK_CANCELLED	000Fh	The requested method has been cancelled
P_INVALID_DATE_TIME_FORMAT	0010h	Invalid date and time format provided
P_NO_CALLBACK_ADDRESS_SET	0011h	The requested method has been refused because no callback address is set
P_INVALID_SIGNATURE	0012h	Invalid digital signature

P_INVALID_SERVICE_TOKEN	0013h	The service capability feature token does not correspond to a token that had been issued, or the issued token 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.
General security errors		
P_USER_NOT_SUBSCRIBED	0030h	A service (or application) is unauthorised to access information and request SCFs with regards to users that are **not** subscribed to it.
P_APPLICATION_NOT_ACTIVATED	0031h	A service (or application) is unauthorised to access information and request SCFs with regards to its subscribed users that have **deactivated** that particular service (or application).
P_USER_PRIVACY	0032h	A service (or application) is unauthorised to access information and request an SCF with regards to its subscribed users that have **set** their privacy flag regarding that particular SCF.
P_GCCS_SERVICE_INFORMATION_MISSING	0100h	Information relating to the Call Control SCF could not be found
P_GCCS_SERVICE_FAULT_ENCOUNTERED	0101h	Fault detected in the Call Control SCF
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 OSA 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_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 SCF
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 OSA 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.
P_DSCS_SERVICE_INFORMATION_MISSING	0400h	Information relating to the Data Session Control SCF could not be found
P_DSCS_SERVICE_FAULT_ENCOUNTERED	0401h	Fault detected in the Data Session Control SCF
P_DSCS_UNEXPECTED_SEQUENCE	0402h	Unexpected sequence of methods, i.e., the sequence does not match the specified state diagrams for the data session.
P_DSCS_INVALID_ADDDRESS	0403h	Invalid address specified
P_DSCS_INVALID_STATE	0404h	Invalid state specified

P_DSCS_INVALID_CRITERIA	0405h	Invalid criteria specified
P_DSCS_INVALID_NETWORK_STATE	0406h	Although the sequence of method calls is allowed by the OSA gateway, the underlying protocol can not support it.

8.1.4.9 TpDate

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

YYYY-MM-DD

where the date is specified as:

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

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

Example:

The 4 December 1998, is encoded as the string:

1998-12-04

8.1.4.10 TpTime

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

HH:MM:SS.mmm

or

HH:MM:SS.mmmZ

where the time is specified as:

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

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

Example

For local time, 10:30 and 15 seconds is encoded as the string:

10:30:15.000

or in UTC it would be:

10:30:15.000Z

8.1.4.11 TpDateAndTime

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

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)
```

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

Example

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
```

8.1.4.12 TpAddress

Defines the structure of data elements that specifies an address.

Structure Member Name	Structure Member Type
`Plan`	`TpAddressPlan`
`AddrString`	`TpString`
`Name`	`TpString`
`Presentation`	`TpAddressPresentation`
`Screening`	`TpAddressScreening`
`SubAddressString`	`TpString`

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

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

8.1.4.13 TpAddressSet

Defines a collection of TpAddress elements.

8.1.4.14 TpAddressPlan

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

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

8.1.4.15 TpAddressPresentation

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

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

8.1.4.16 TpAddressRange

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

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

Some examples for E164 addresses:

“123” matches specified number.

“123*” matches all numbers starting with 123 (including 123 itself).

“123??*” matches all numbers starting with 123 and at least 5 digits long.

“123???” matches all numbers starting with 123 and exactly 6 digits long

For e-mail style addresses, the wildcards can be used at the beginning of the addrString:

- *@3gpp.org matches all email addresses in the 3gpp.org domain.

The following address ranges are illegal:

- 1?3
- 1*3

- ?123*

8.1.4.17 TpAddressScreening

Defines whether an address has been screened by the application.

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

8.1.4.18 TpAddressError

Defines the reasons why an address is invalid.

Name	Value	Description
P_ADDRESS_INVALID_UNDEFINED	0	Undefined error
P_ADDRESS_INVALID_MISSING	1	Mandatory address not present
P_ADDRESS_INVALID_MISSING_ELEMENT	2	Mandatory address element not present
P_ADDRESS_INVALID_OUT_OF_RANGE	3	Address is outside of the valid range
P_ADDRESS_INVALID_INCOMPLETE	4	Address is incomplete
P_ADDRESS_INVALID_CANNOT_DECODE	5	Address cannot be decoded

8.1.4.19 TpURL

This data type is identical to a TpString and contains a URL address. The usage of this type is distinct of TpAddress, which can also hold an URL. The latter contains a user address which can be specified in many ways: IP, mail, URL, X.300, E164. On the other hand, the TpURL type does not hold the address of a user and always represents a URL. This type is used in user interaction and defines the URL of the text or stream to be sent to an end-user. It is therefore inappropriate to use a general address here.

8.1.4.20 TpPrice

This data type is identical to a TpString. It specifies price information, which is used in user interaction when an announcement is being played and additional information is given to the user. This is defined as the string of characters (digits) in the following format:

DDDDDD.DD

8.1.4.21 TpAoCInfo

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

Sequence Element Name	Sequence Element Type	Description
ChargeOrder	TpAoCOrder	Charge order

Currency	TpString	Currency unit according to ISO-4217:1995

8.1.4.22 TpAoCOrder

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

	Tag Element Type	
	TpAoCOrderCategory	

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

8.1.4.23 TpCallAoCOrderCategory

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

8.1.4.24 TpChargeAdviceInfo

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

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

8.1.4.25 TpCAIElements

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

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

8.1.4.26 TpChargePerTime

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

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

8.2 Framework Data Definitions

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

This document is written using Hypertext link, to aid navigation through the data structures. Underlined text represents Hypertext links.

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

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

8.2.1 Common Framework Data Definitions

8.2.1.1 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.

8.2.1.2 TpClientAppIDList

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

8.2.1.3 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

8.2.1.4 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

8.2.1.5 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.

8.2.1.6 TpPropertyName

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

8.2.1.7 TpPropertyValue

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

8.2.1.8 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

8.2.1.9 TpPropertyList

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

8.2.1.10 TpEntOpIDList

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

8.2.1.11 TpFwID

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

8.2.1.12 TpService

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

Sequence Element Name	Sequence Element Type	Documentation
ServiceID	TpServiceID	
ServicePropertyList	TpServicePropertyList	

8.2.1.13 TpServiceList

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

8.2.1.14 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	

8.2.1.15 TpServiceID

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

8.2.1.16 TpServiceIDList

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

8.2.1.17 TpServiceIDRef

Defines a Reference to type TpServiceId.

8.2.1.18 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

8.2.1.19 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

8.2.1.20 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).

8.2.1.21 TpServiceTypeProperty

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

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

8.2.1.22 TpServiceTypePropertyList

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

8.2.1.23 TpServicePropertyMode

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

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

8.2.1.24 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.

8.2.1.25 TpServicePropertyName

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

8.2.1.26 TpServicePropertyNameList

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

8.2.1.27 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.

8.2.1.28 TpServicePropertyValueList

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

8.2.1.29 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	

8.2.1.30 TpServicePropertyList

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

8.2.1.31 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.

8.2.1.32 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 (true) or disabled (false)

8.2.1.33 TpServiceTypeName

This data type is identical to TpString and describes a valid SCF type name.

8.2.1.34 TpServiceTypeNameList

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

8.2.2 Trust and Security Management Data Definitions

8.2.2.1 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_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 for OSA release 99:

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

8.2.2.2 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 Authentication interface, e.g. CORBA Security. OSA 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 value is defined for OSA release 99:

String Value	Description
P_AUTHENTICATION	Indicates the default authentication method, i.e. the IpAuthentication and IpAppAuthentication interfaces.

8.2.2.3 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 for OSA release 99.

String Value	Description
NULL	An empty (NULL) string indicates no client capabilities.
P_DES_56	A simple transfer of secret information that is shared between the client application and the framework with protection against interception on the link provided by the DES algorithm with a 56bit shared secret key
_128	A simple transfer of secret information that is shared between the client entity and the framework with protection against interception on the link provided by the DES algorithm with a 128bit shared secret key
P_RSA_512	A public-key cryptography system providing authentication without prior exchange of secrets using 512 bit keys
P_RSA_1024	A public-key cryptography system providing authentication without prior exchange of secrets using 1024bit keys

8.2.2.4 TpAuthCapabilityList

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

8.2.2.5 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).

8.2.2.6 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	IpOSARef	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.

8.2.2.7 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 for OSA release 99.

Character String Value	Description
P_DISCOVERY	The name for the Discovery 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.

8.2.2.8 TpServiceAccessControl

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

Sequence Element Name	Sequence Element Type
Policy	TpString
TrustLevel	TpString

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

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

8.2.2.9 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.

8.2.2.10 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.

8.2.2.11 TpSigningAlgorithm

This data type is identical to a TpString, and is defined as a string of characters that identify the signing algorithm that must be used. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined for OSA release 99.

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

8.2.3 Integrity Management Data Definitions

8.2.3.1 TpActivityTestRes

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

8.2.3.2 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
FaultRecords	TpFaultStatsSet

8.2.3.3 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.

8.2.3.4 TpFaultStatsSet

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

8.2.3.5 TpActivityTestID

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

8.2.3.6 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

8.2.3.7 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)

8.2.3.8 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

8.2.3.9 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

8.2.3.10 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

8.2.3.11 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

8.2.3.12 TpTimeInterval

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

Sequence Element Name	Sequence Element Type
StartTime	TpDateAndTime
StopTime	TpDateAndTime

8.2.3.13 TpLoadPolicy

Defines the load balancing policy.

Sequence Element Name	Sequence Element Type
LoadPolicy	TpString

8.2.3.14 TpLoadStatistic

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

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

8.2.3.15 TpLoadStatisticList

Defines a Numbered List of Data Elements of type TpLoadStatistic.

8.2.3.16 TpLoadStatisticData

Defines the Sequence of Data Elements that represents load statistic information

Sequence Element Name	Sequence Element Type
LoadValue	TpFloat
LoadLevel	TpLoadLevel

Note: LoadValue is expressed as a percentage.

8.2.3.17 TpLoadStatisticEntityID

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

	Tag Element Type	
	TpLoadStatisticEntityType	

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

8.2.3.18 TpLoadStatisticEntityType

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

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

8.2.3.19 TpLoadStatisticInfo

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

	Tag Element Type	
	TpLoadStatisticInfoType	

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

8.2.3.20 TpLoadStatisticInfoType

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

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

8.2.3.21 TpLoadStatisticError

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

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

8.3 Generic Call Control Data Definitions

The constants and types defined in the following sections are defined in the *org.threegpp.osa.gccs* package.

8.3.1 Interface definitions

8.3.1.1 IpAppCall

Defines the address of an `IAppCall` Interface.

8.3.1.2 IpAppCallRef

Defines a Reference to type `IAppCall`

8.3.1.3 IpAppCallRefRef

Defines a Reference to type `IAppCallRef`.

8.3.1.4 IpAppCallControlManager

Defines the address of an `IAppCallControlManager` Interface.

8.3.1.5 IpAppCallControlManagerRef

Defines a Reference to type `IAppCallControlManager`.

8.3.1.6 IpCall

Defines the address of an `ICall` Interface.

8.3.1.7 IpCallRef

Defines a Reference to type `ICall`.

8.3.1.8 IpCallRefRef

Defines a Reference to type `ICallRef`.

8.3.1.9 IpCallControlManager

Defines the address of an `ICallControlManager` Interface.'

8.3.1.10 IpCallControlManagerRef

Defines a Reference to type `ICallControlManager`.

8.3.2 Event Notification data definitions

8.3.2.1 TpCallEventName

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 TpCallReportType data-type.

Name	Value	Description
P_EVENT_NAME_UNDEFINED	0	Undefined
P_EVENT_GCCS_OFFHOOK_EVENT	1	GCCS – Offhook event. This can be used for hot-line features. In case this event is set in the TpCallEventCriteria, only the originating address(es) may be specified in the criteria.
P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT	2	GCCS – Address information collected The network has collected the information from the calling party, but not yet analysed the information. The number can still be incomplete. Applications might set notification for this event when part of the number analysis needs to be done in the application.
P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT	4	GCCS – Address information is analysed. The dialled number is a valid and complete number in the network.
P_EVENT_GCCS_CALLED_PARTY_BUSY	8	GCCS – Called party is busy
P_EVENT_GCCS_CALLED_PARTY_UNREACHABLE	16	GCCS – Called party is unreachable This can happen when the called party has a mobile phone that is switched off.
P_EVENT_GCCS_NO_ANSWER_FROM_CALLED_PARTY	32	GCCS – No answer from called

		party
P_EVENT_GCCS_ROUTE_SELECT_FAILURE	64	GCCS – Failure in routing the call
P_EVENT_GCCS_ANSWER_FROM_CALL_PARTY	128	GCCS – Party answered call.

8.3.2.2 TpCallEventCriteria

Defines the Sequence of Data Elements that specify the criteria for an event notification.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddressRange	Defines the destination address or address range for which the notification is requested
OriginationAddress	TpAddressRange	Defines the origination address or address range for which the notification is requested
CallEventName	TpCallEventName	Name of the event(s)
CallNotificationType	TpCallNotificationType	Indicates whether it is related to the originating or the terminating user in the call.
MonitorMode	TpCallMonitorMode	Defines the mode that the call is in following the notification. Monitor mode P_CALL_MONITOR_MODE_DO_NOT_MONITOR is not a legal value here.

8.3.2.3 TpCallEventCriteriaResult

Defines a sequence of data elements that specify a requested call event notification criteria with the associated assignmentID.

Sequence Element Name	Sequence Element Type	Sequence Element Description
EventCriteria	TpCallEventCriteria	The event criteria that were specified by the application.
AssignmentID	TpInt32	The associated assignementID. This can be used to disable the notification.

8.3.2.4 TpCallEventCriteriaResultSet

Defines a set of TpCallEventCriteriaResult.

8.3.2.5 TpCallNotificationType

Defines the type of notification. Indicates whether it is related to the originating or the terminating user in the call.

Name	Value	Description
P_ORIGINATING	1	Indicates that the notification is related to the originating user in the call.
P_TERMINATING	2	Indicates that the notification is related to the terminating user in the call.

8.3.2.6 TpCallEventInfo

Defines the Sequence of Data Elements that specify the information returned to the application in a New Call event notification.

Sequence Element Name	Sequence Element Type
DestinationAddress	TpAddress
OriginatingAddress	TpAddress
OriginalDestinationAddress	TpAddress
RedirectingAddress	TpAddress
CallAppInfo	TpCallAppInfoSet
CallEventName	TpCallEventName
CallNotificationType	TpCallNotificationType
MonitorMode	TpCallMonitorMode

8.3.3 Generic Call Control Type definitions

8.3.3.1 TpCallAlertingMechanism

This data type is identical to a TpInt32, and defines the mechanism that will be used to alert a called party. The values of this data type are operator specific.

8.3.3.2 TpCallAppInfo

Defines the Tagged Choice of Data Elements that specify application-related call information.

	Tag Element Type	
	TpCallAppInfoType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_APP_ALERTING_MECHANISM	TpCallAlertingMechanism	CallAppAlertingMechanism
P_CALL_APP_NETWORK_ACCESS_TYPE	TpCallNetworkAccessType	CallAppNetworkAccessType
P_CALL_APP_TELE_SERVICE	TpCallTeleService	CallAppTeleService
P_CALL_APP_BEARER_SERVICE	TpCallBearerService	CallAppBearerService
P_CALL_APP_PARTY_CATEGORY	TpCallPartyCategory	CallAppPartyCategory
P_CALL_APP_PRESENTATION_ADDRESS	TpAddress	CallAppPresentationAddress
P_CALL_APP_GENERIC_INFO	TpString	CallAppGenericInfo
P_CALL_APP_ADDITIONAL_ADDRESS	TpAddress	CallAppAdditionalAddress

CallAppPresentationAddress contains presentation address.

CallAppGenericInfo contains operator specific information.

CallAppAdditionalAddress contains additional address.

8.3.3.3 TpCallAppInfoType

Defines the type of application related call information.

Name	Value	Description
P_CALL_APP_UNDEFINED	0	Undefined
P_CALL_APP_ALERTING_MECHANISM	1	The alerting mechanism or pattern to use
P_CALL_APP_NETWORK_ACCESS_TYPE	2	The network access type (e.g. ISDN)

P_CALL_APP_TELE_SERVICE	3	Indicates the tele-service (e.g. speech) and related info such as clearing programme
P_CALL_APP_BEARER_SERVICE	4	Indicates the bearer service (e.g. 64kb/s unrestricted data).
P_CALL_APP_PARTY_CATEGORY	5	The category of the calling or called party
P_CALL_APP_PRESENTATION_ADDRESS	6	The address to be presented to other call parties
P_CALL_APP_GENERIC_INFO	7	Carries unspecified application-Service Capability Feature information
P_CALL_APP_ADDITIONAL_ADDRESS	8	Indicates an additional address

8.3.3.4 TpCallAppInfoSet

Defines a Numbered Set of Data Elements of TpCallAppInfo.

8.3.3.5 TpCallBearerService

This data type defines the type of call application-related specific information (Q.931: Information Transfer Capability, and 3GPP TS 22.002)

Name	Value	Description
P_CALL_BEARER_SERVICE_UNKNOWN	0	Bearer capability information unknown at this time
P_CALL_BEARER_SERVICE_SPEECH	1	Speech
P_CALL_BEARER_SERVICE_DIGITALUNRESTRICTED	2	Unrestricted digital information
P_CALL_BEARER_SERVICE_ DIGITALRESTRICTED	3	Restricted digital information
P_CALL_BEARER_SERVICE_AUDIO	4	3.1 kHz audio
P_CALL_BEARER_SERVICE_ DIGITALUNRESTRICTEDTONES	5	Unrestricted digital information with tones/announcements
P_CALL_BEARER_SERVICE_VIDEO	6	Video

8.3.3.6 TpCallChargePlan

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

Sequence Element Name	Sequence Element Type	Description
ChargeOrderType	TpCallChargeOrder	Charge order
Currency	TpString	Currency unit according to ISO-4217:1995
AdditionalInfo	TpString	Descriptive string which is sent to the billing system without prior evaluation. Could be included in the ticket.

Valid Currencies are:

ADP, AED, AFA, ALL, AMD, ANG, AON, AOR, ARS, ATS, AUD, AWG, AZM, BAM, BBD, BDT, BEF, BGL, BGN, BHD, BIF, BMD, BND, BOB, BOV, BRL, BSD, BTN,

BWP, BYB, BZD, CAD, CDF, CHF, CLF, CLP, CNY, COP, CRC, CUP, CVE, CYP, CZK, DEM, DJF, DKK, DOP, DZD, ECS, ECV, EEK, EGP, ERN, ESP, ETB, EUR, FIM, FJD, FKP, FRF, GBP, GEL, GHC, GIP, GMD, GNF, GRD, GTQ, GWP, GYD, HKD, HNL, HRK, HTG, HUF, IDR, IEP, ILS, INR, IQD, IRR, ISK, ITL, JMD, JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KYD, KZT, LAK, LBP, LKR, LRD, LSL, LTL, LUF, LVL, LYD, MAD, MDL, MGF, MKD, MMK, MNT, MOP, MRO, MTL, MUR, MVR, MWK, MXN, MXV, MYR, MZM, NAD, NGN, NIO, NLG, NOK, NPR, NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PTE, PYG, QAR, ROL, RUB, RUR, RWF, SAR, SBD, SCR, SDD, SEK, SGD, SHP, SIT, SKK, SLL, SOS, SRG, STD, SVC, SYP, SZL, THB, TJR, TMM, TND, TOP, TPE, TRL, TTD, TWD, TZS, UAH, UGX, USD, USN, USS, UYU, UZS, VEB, VND, VUV, WST, XAF, XAG, XAU, XBA, XBB, XBC, XBD, XCD, XDR, XFO, XFU, XOF, XPD, XPF, XPT, XTS, XXX, YER, YUM, ZAL, ZAR, ZMK, ZRN, ZWD.

XXX is used for transactions where no currency is involved.

8.3.3.7 TpCallChargeOrder

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

	Tag Element Type	
	TpCallChargeOrderCategory	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_CHARGE_PER_TIME	TpChargePerTime	ChargePerTime
P_CALL_CHARGE_NETWORK	TpString	NetworkCharge

8.3.3.8 TpCallChargeOrderCategory

Name	Value	Description
P_CALL_CHARGE_PER_TIME	0	Charge per time
P_CALL_CHARGE_NETWORK	1	Operator specific charge plan specification, e.g. charging table name / charging table entry

8.3.3.9 TpCallEndedReport

Defines the Sequence of Data Elements that specify the reason for the call ending.

Sequence Element Name	Sequence Element Type	
CallLegSessionID	TpSessionID	The leg that initiated the release of the call. If the call release was not initiated by the leg, then this value is set to –1.
Cause	TpCallReleaseCause	The cause of the call ending.

8.3.3.10 TpCallError

Defines the Sequence of Data Elements that specify the additional information relating to an undefined call error.

Sequence Element Name	Sequence Element Type
ErrorTime	TpDateAndTime
ErrorType	TpCallErrorType
AdditionalErrorInfo	TpCallAdditionalErrorInfo

8.3.3.11 TpCallAdditionalErrorInfo

Defines the Tagged Choice of Data Elements that specify additional call error and call error specific information. This is also used to specify call leg errors and call information errors.

	Tag Element Type	
	TpCallErrorType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_ERROR_UNDEFINED	NULL	Undefined
P_CALL_ERROR_INVALID_ADDRESS	TpAddressError	CallErrorInvalidAddress
P_CALL_ERROR_INVALID_STATE	NULL	Undefined

8.3.3.12 TpCallErrorType

Defines a specific call error.

Name	Value	Description
P_CALL_ERROR_UNDEFINED	0	Undefined; the method failed or was refused, but no specific reason can be given.
P_CALL_ERROR_INVALID_ADDRESS	1	The operation failed because an invalid address was given
P_CALL_ERROR_INVALID_STATE	2	The call was not in a valid state for the requested operation

8.3.3.13 TpCallFault

Defines the cause of the call fault detected.

Name	Value	Description
P_CALL_FAULT_UNDEFINED	0	Undefined
P_TIMEOUT_ON_RELEASE	1	This fault occurs when the final report has been sent to the application, but the application did not explicitly release or deassign the call object, within a specified time. The timer value is operator specific.
P_TIMEOUT_ON_INTERRUPT	2	This fault occurs when the application did not instruct the gateway how to handle the call within a specified time, after the gateway reported an event that was requested by the application in interrupt mode. The timer value is operator specific.

8.3.3.14 TpCallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Generic Call object

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallReference	IpCallRef	This element specifies the interface reference for the call object.
CallSessionID	TpSessionID	This element specifies the call session ID of the call.

8.3.3.15 TpCallIdentifierRef

Defines a Reference to type TpCallIdentifier.

8.3.3.16 TpCallInfoReport

Defines the Sequence of Data Elements that specify the call information requested. Information that was not requested is invalid.

Sequence Element Name	Sequence Element Type	Description
CallInfoType	TpCallInfoType	The type of call report.
CallInitiationStartTime	TpDateAndTime	The time and date when the call, or follow-on call, was started.
CallConnectedToResourceTime	TpDateAndTime	The date and time when the call was connected to the resource. This data element is only valid where information on user interaction is reported.
CallConnectedToDestinationTime	TpDateAndTime	The date and time when the call was connected to the destination (i.e. when the destination answered the call). If the destination did not answer the time is set to an empty string. This data element is invalid where information on user interaction is reported with an intermediate report.
CallEndTime	TpDateAndTime	The date and time when the call, follow-on call or user-interaction was terminated.
Cause	TpCallReleaseCause	The cause of call termination.

8.3.3.17 TpCallInfoType

Defines the type of call information requested and reported. The values may be combined by a logical 'OR' function.

Name	Value	Description
P_CALL_INFO_UNDEFINED	00h	Undefined
P_CALL_INFO_TIMES	01h	Relevant call times
P_CALL_INFO_RELEASE_CAUSE	02h	Call release cause
P_CALL_INFO_INTERMEDIATE	04h	Send only intermediate reports. When this is not specified the information report will only be sent when the call has ended. When intermediate reports are requested a report will be generated between follow-on calls, i.e. when a party leaves the call.

8.3.3.18 TpCallMonitorMode

Defines the mode that the call will monitor for events, or the mode that the call is in following a detected event.

Name	Value	Description
P_CALL_MONITOR_MODE_INTERRUPT	0	The call event is intercepted by the call control SCF and call processing is interrupted. The application is notified of the event and call processing resumes following an appropriate API call or network event (such as a call release)
P_CALL_MONITOR_MODE_NOTIFY	1	The call event is detected by the call control SCF but not intercepted. The application is notified of the event and call processing continues
P_CALL_MONITOR_MODE_DO_NOT_MONITOR	2	Do not monitor for the event

8.3.3.19 TpCallNetworkAccessType

This data defines the bearer capabilities associated with the call. (3GPP TS 24.002) This information is network operator specific and may not always be available because there is no standard protocol to retrieve the information.

Name	Value	Description
P_CALL_NETWORK_ACCESS_TYPE_UNKNOWN	0	Network type information unknown at this time
P_CALL_NETWORK_ACCESS_TYPE_POT	1	POTS
P_CALL_NETWORK_ACCESS_TYPE_ISDN	2	ISDN
P_CALL_NETWORK_ACCESS_TYPE_DIALUPINTERNET	3	Dial-up Internet
P_CALL_NETWORK_ACCESS_TYPE_XDSL	4	xDSL
P_CALL_NETWORK_ACCESS_TYPE_WIRELESS	5	Wireless

8.3.3.20 TpCallOverloadType

Defines the type of call overload that has been detected (and possibly acted upon) by the network.

Name	Value	Description
P_CALL_OVERLOAD_TYPE_UNDEFINED	0	Infinite interval (do not admit any calls)
P_CALL_OVERLOAD_TYPE_NEW_CALLS	1	New calls to the application are causing overload (i.e. inbound overload)
P_CALL_OVERLOAD_TYPE_ROUTED_CALLS	2	Calls being routed to destination or origination addresses by the application are causing overload (i.e. outbound overload)

8.3.3.21 TpCallServiceCode

Defines the Sequence of Data Elements that specify the service code and type of service code received during a call. The service code type defines how the value string should be interpreted.

Sequence Element Name	Sequence Element Type
CallServiceCodeType	TpCallServiceCodeType
ServiceCodeValue	TpString

8.3.3.22 TpCallServiceCodeType

Defines the different types of service codes that can be received during the call.

Name	Value	Description
P_CALL_SERVICE_CODE_UNDEFINED	0	The type of service code is unknown. The corresponding string is operator specific.
P_CALL_SERVICE_CODE_DIGITS	1	The user entered a digit sequence during the call. The corresponding string is an ascii representation of the received digits.
P_CALL_SERVICE_CODE_FACILITY	2	A facility information element is received. The corresponding string contains the facility information element as defined in ITU Q.932
P_CALL_SERVICE_CODE_U2U	3	A user-to-user message was received. The associated string contains the content of the user-to-user information element.
P_CALL_SERVICE_CODE_HOOKFLASH	4	The user performed a hookflash, optionally followed by some digits. The corresponding string is an ascii representation of the entered digits.
P_CALL_SERVICE_CODE_RECALL	5	The user pressed the register recall button, optionally followed by some digits. The corresponding string is an ascii representation of the entered digits.

8.3.3.23 TpCallPartyCategory

This data type defines the category of a calling party. (Q.763: Calling Party Category / Called Party Category)

Name	Value	Description
P_CALL_PARTY_CATEGORY_UNKNOWN	0	calling party's category unknown at this time
P_CALL_PARTY_CATEGORY_OPERATOR_F	1	operator, language French
P_CALL_PARTY_CATEGORY_OPERATOR_E	2	operator, language English
P_CALL_PARTY_CATEGORY_OPERATOR_G	3	operator, language German
P_CALL_PARTY_CATEGORY_OPERATOR_R	4	operator, language Russian
P_CALL_PARTY_CATEGORY_OPERATOR_S	5	operator, language Spanish
P_CALL_PARTY_CATEGORY_ORDINARY_SUB	6	ordinary calling subscriber
P_CALL_PARTY_CATEGORY_PRIORITY_SUB	7	calling subscriber with priority
P_CALL_PARTY_CATEGORY_DATA_CALL	8	data call (voice band data)
P_CALL_PARTY_CATEGORY_TEST_CALL	9	test call
P_CALL_PARTY_CATEGORY_PAYPHONE	10	payphone

8.3.3.24 TpCallReleaseCause

Defines the Sequence of Data Elements that specify the cause of the release of a call.

Sequence Element Name	Sequence Element Type
Value	TpInt32
Location	TpInt32

Note: the Value and Location are specified as in ITU-T recommendation Q.850.

8.3.3.25 TpCallReport

Defines the Sequence of Data Elements that specify the call report and call leg report specific information.

Sequence Element Name	Sequence Element Type
MonitorMode	TpCallMonitorMode
CallEventTime	TpDateAndTime
CallReportType	TpCallReportType
AdditionalReportInfo	TpCallAdditionalReportInfo

8.3.3.26 TpCallAdditionalReportInfo

Defines the Tagged Choice of Data Elements that specify additional call report information for certain types of reports.

	Tag Element Type	
	TpCallReportType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_REPORT_UNDEFINED	NULL	Undefined
P_CALL_REPORT_PROGRESS	NULL	Undefined
P_CALL_REPORT_ALERTING	NULL	Undefined
P_CALL_REPORT_ANSWER	NULL	Undefined
P_CALL_REPORT_BUSY	TpCallReleaseCause	Busy
P_CALL_REPORT_NO_ANSWER	NULL	Undefined
P_CALL_REPORT_DISCONNECT	TpCallReleaseCause	CallDisconnect
P_CALL_REPORT_REDIRECTED	TpAddress	ForwardAddress
P_CALL_REPORT_SERVICE_CODE	TpCallServiceCode	ServiceCode
P_CALL_REPORT_ROUTING_FAILURE	TpCallReleaseCause	RoutingFailure

8.3.3.27 TpCallReportRequest

Defines the Sequence of Data Elements that specify the criteria relating to call report requests.

Sequence Element Name	Sequence Element Type
MonitorMode	TpCallMonitorMode
CallReportType	TpCallReportType
AdditionalReportcriteria	TpCallAdditionalReportCriteria

8.3.3.28 TpCallAdditionalReportCriteria

Defines the Tagged Choice of Data Elements that specify specific criteria.

	Tag Element Type	
	TpCallReportType	

Tag Element Value	Choice Element Type	Choice Element Name

P_CALL_REPORT_UNDEFINED	NULL	Undefined
P_CALL_REPORT_PROGRESS	NULL	Undefined
P_CALL_REPORT_ALERTING	NULL	Undefined
P_CALL_REPORT_ANSWER	NULL	Undefined
P_CALL_REPORT_BUSY	NULL	Undefined
P_CALL_REPORT_NO_ANSWER	TpDuration	NoAnswerDuration
P_CALL_REPORT_DISCONNECT	NULL	Undefined
P_CALL_REPORT_REDIRECTED	NULL	Undefined
P_CALL_REPORT_SERVICE_CODE	TpCallServiceCode	ServiceCode
P_CALL_REPORT_ROUTING_FAILURE	NULL	Undefined

8.3.3.29 TpCallReportRequestSet

Defines a Numbered Set of Data Elements of TpCallReportRequest.

8.3.3.30 TpCallReportType

Defines a specific call event report type.

Name	Value	Description
P_CALL_REPORT_UNDEFINED	0	Undefined
P_CALL_REPORT_PROGRESS	1	Call routing progress event: an indication from the network that progress has been made in routing the call to the requested called party.
P_CALL_REPORT_ALERTING	2	Call is alerting at the called party.
P_CALL_REPORT_ANSWER	3	Call answered at address
P_CALL_REPORT_BUSY	4	Called address refused call due to busy
P_CALL_REPORT_NO_ANSWER	5	No answer at called address
P_CALL_REPORT_DISCONNECT	6	The called party has disconnected.
P_CALL_REPORT_REDIRECTED	7	Call redirected to new address: an indication from the network that the call has been redirected to a new address.
P_CALL_REPORT_SERVICE_CODE	8	Mid-call service code received
P_CALL_REPORT_ROUTING_FAILURE	9	Call routing failed - re-routing is possible

8.3.3.31 TpCallTeleService

This data type defines the tele-service associated with the call. (Q.763: User Teleservice Information, Q.931: High Layer Compatitibility Information, and 3GPP TS 22.003)

Name	Value	Description
P_CALL_TELE_SERVICE_UNKNOWN	0	Teleservice information unknown at this time
P_CALL_TELE_SERVICE_TELEPHONY	1	Telephony
P_CALL_TELE_SERVICE_FAX_2_3	2	Facsimile Group 2/3
P_CALL_TELE_SERVICE_FAX_4_I	3	Facsimile Group 4, Class I
P_CALL_TELE_SERVICE_FAX_4_II_III	4	Facsimile Group 4, Classes II and III
P_CALL_TELE_SERVICE_VIDEOTEX_SYN	5	Syntax based Videotex
P_CALL_TELE_SERVICE_VIDEOTEX_INT	6	International Videotex interworking via gateways or interworking units
P_CALL_TELE_SERVICE_TELEX	7	Telex service
P_CALL_TELE_SERVICE_MHS	8	Message Handling Systems
P_CALL_TELE_SERVICE_OSI	9	OSI application
P_CALL_TELE_SERVICE_FTAM	10	FTAM application
P_CALL_TELE_SERVICE_VIDEO	11	Videotelephony
P_CALL_TELE_SERVICE_VIDEO_CONF	12	Videoconferencing
P_CALL_TELE_SERVICE_AUDIOGRAPH_CONF	13	Audiographic conferencing
P_CALL_TELE_SERVICE_MULTIMEDIA	14	Multimedia services
P_CALL_TELE_SERVICE_CS_INI_H221	15	Capability set of initial channel of H.221
P_CALL_TELE_SERVICE_CS_SUB_H221	16	Capability set of subsequent channel of H.221
P_CALL_TELE_SERVICE_CS_INI_CALL	17	Capability set of initial channel associated with an active 3.1 kHz audio or speech call.
P_CALL_TELE_SERVICE_DATATRAFFIC	18	Data traffic.
P_CALL_TELE_SERVICE_EMERGENCY_CALLS	19	Emergency Calls
P_CALL_TELE_SERVICE_SMS_MT_PP	20	Short message MT/PP
P_CALL_TELE_SERVICE_SMS_MO_PP	21	Short message MO/PP
P_CALL_TELE_SERVICE_CELL_BROADCAST	22	Cell Broadcast Service
P_CALL_TELE_SERVICE_ALT_SPEECH_FAX_3	23	Alternate speech and facsimile group 3
P_CALL_TELE_SERVICE_AUTOMATIC_FAX_3	24	Automatic Facsimile group 3
P_CALL_TELE_SERVICE_VOICE_GROUP_CALL	25	Voice Group Call Service
P_CALL_TELE_SERVICE_VOICE_BROADCAST	26	Voice Broadcast Service

8.3.3.32 TpCallSuperviseReport

Defines the responses from the call control SCF for calls that are supervised. The values may be combined by a logical 'OR' function.

Name	Value	Description
P_CALL_SUPERVISE_TIMEOUT	01h	The call supervision timer has expired
P_CALL_SUPERVISE_CALL_ENDED	02h	The call has ended, either due to timer expiry or call party release. In case the called party disconnects but a follow-on call can still be made also this indication is used.
P_CALL_SUPERVISE_TONE_APPLIED	04h	A warning tone has been applied This is only sent in combination with P_CALL_SUPERVISE_TIMEOUT.
P_CALL_SUPERVISE_UI_FINISHED	08h	The user interaction has finished.

8.3.3.33 TpCallSuperviseTreatment

Defines the treatment of the call by the call control SCF when the call supervision timer expires. The values may be combined by a logical 'OR' function.

Name	Value	Description
P_CALL_SUPERVISE_RELEASE	01h	Release the call when the call supervision timer expires
P_CALL_SUPERVISE_RESPOND	02h	Notify the application when the call supervision timer expires
P_CALL_SUPERVISE_APPLY_TONE	04h	Send a warning tone to the controlling party when the call supervision timer expires. If call release is requested, then the call will be released following the tone after an administered time period.

8.4 User Interaction Data Definitions

The constants and types defined in the following sections are defined in the *org.threegpp.osa.guis* package.

8.4.1 Interface definitions

8.4.1.1 IpUI

Defines the address of an `IUI` Interface.

8.4.1.2 IpUIRef

Defines a Reference to type `IUI`.

8.4.1.3 IpUIRefRef

Defines a Reference to type `IUIRef`.

8.4.1.4 IpUIManager

Defines the address of an `IUIManager` Interface.

8.4.1.5 IpUIManagerRef

Defines a Reference to type `IUIManager`.

8.4.1.6 IpAppUI

Defines the address of an `IAppUI` Interface.

8.4.1.7 IpAppUIRef

Defines a Reference to type `IAppUI`.

8.4.1.8 IpAppUIRefRef

Defines a Reference to type `IAppUIRef`.

8.4.1.9 IpAppUIManager

Defines the address of an `IAppUIManager` Interface.

8.4.1.10 IpAppUIManagerRef

Defines a Reference to type `IAppUIManager`.

8.4.2 Type definitions

8.4.2.1 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.`

8.4.2.2 TpUICallIdentifierRef

Defines a reference to type TpUICallIdentifier.

8.4.2.3 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 hs been successful.

8.4.2.4 TpUIError

Defines the UI call error codes.

Name	Value	Description
P_UI_ERROR_UNDEFINED	0	Undefined error
P_UI_ERROR_ILLEGAL_ID	1	The information id specified is invalid
P_UI_ERROR_ID_NOT_FOUND	2	A legal information id is not known to the the User Interaction SCF
P_UI_ERROR_RESOURCE_UNAVAILABLE	3	The information resources used by the User Interaction SCF 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_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

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.

8.4.2.5 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.

8.4.2.6 TpUIEventInfo

Defines the Sequence of Data Elements that specify a UI notification

Structure Element Name	Structure Element Type	
OriginatingAddress	TpAddress	Defines the originating address.
DestinationAddress	TpAddress	Defines the destination address.
ServiceCode	TpString	Defines a 2 digit code indicating the UI to be triggered. The value is operator specific.
DataTypeIndication	TpUIEventInfoDataType	Identifies the type of contents in the dataString.
DataString	TpString	Freely defined data string with a limited length e.g. 160 bytes according to the network policy.

8.4.2.7 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

8.4.2.8 TpUIFault

Defines the cause of the UI fault detected.

Name	Value	Description
P_UI_FAULT_UNDEFINED	0	Undefined
P_UI_CALL_DEASSIGNED	1	The related Call object has been deassigned. No further interaction is possible. Already requested announcements will continue but no requested reports will be send to the application.

8.4.2.9 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.

8.4.2.10 TpUIIdentifierRef

Defines a reference to type TpUIIdentifier.

8.4.2.11 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.

8.4.2.12 TpUIInfoType

Defines the type of the information to be sent 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.

8.4.2.13 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	1	Confirmation that the announcement has ended
P_UI_REPORT_LEGAL_INPUT	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

8.4.2.14 TpUIResponseRequest

Defines the situations for which a response is expected following the user interaction.

Name	Value	Description
P_UI_RESPONSE_REQUIRED	1	The User Interaction Call must send a response when the request has completed.
P_UI_LAST_ANNOUNCEMENT_IN_A_ROW	2	This is the final announcement within a sequence. It might, however, be that additional announcements will be requested at a later moment. The Call User Interaction Call SCF 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 bitmask, i.e. the values can be added to derived the final meaning.

8.4.2.15 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

8.4.2.16 TpUIVariableInfoSet

Defines a Numbered Set of Data Elements of TpUIVariableInfo.

8.4.2.17 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_VARIALBE_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

8.5 Data Session Control Data Definitions

The constants and types defined in the following sections are defined in the *org.threegpp.osa.dscs* package.

8.5.1 Interface definitions

8.5.1.1 IpAppDataSession

Defines the address of an `IpAppDataSession` Interface.

8.5.1.2 IpAppDataSessionRef

Defines a Reference to type `IpAppDataSession`

8.5.1.3 IpAppDataSessionRefRef

Defines a Reference to type `IpAppDataSessionRef`.

8.5.1.4 IpAppDataSessionControlManager

Defines the address of an `IpAppDataSessionControlManager` Interface.

8.5.1.5 IpAppDataSessionControlManagerRef

Defines a Reference to type `IpAppDataSessionControlManager`.

8.5.1.6 IpDataSession

Defines the address of an `IpDataSession` Interface.

8.5.1.7 IpDataSessionRef

Defines a Reference to type `IpDataSession`.

8.5.1.8 IpDataSessionRefRef

Defines a Reference to type `IpDataSessionRef`.

8.5.1.9 IpDataSessionControlManager

Defines the address of an `IpDataSessionManager` Interface.

8.5.1.10 IpDataSessionManagerRef

Defines a Reference to type `IpDataSessionControlManager`.

8.5.2 Event Notification data definitions

8.5.2.1 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.

8.5.2.2 TpDataSessionMonitorMode

Defines the mode that the call will monitor for events, or the mode that the call is in following a detected event.

Name	Value	Description
P_DATA_SESSION_MONITOR_MODE_INTERRUPT	0	The data session event is intercepted by the data session control service and data session establishment is interrupted. The application is notified of the event and data session establishement resumes following an appropriate API call or network event (such as a data session release)
P_DATA_SESSION_MONITOR_MODE_NOTIFY	1	The data session event is detected by the data session control service but not intercepted. The application is notified of the event and data session establishment continues
P_DATA_SESSION_MONITOR_MODE_DO_NOT_MONITOR	2	Do not monitor for the event

8.5.2.3 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.

8.5.2.4 TpDataSessionEventInfo

Defines the Sequence of Data Elements that specify the information returned to the application in a `Data Session` event notification.

Sequence Element Name	Sequence Element Type
DestinationAddress	TpAddress
OriginatingAddress	TpAddress
DataSessionEventName	TpDataSessionEventName
MonitorMode	TpDataSessionMonitorMode

8.5.2.5 TpDataSessionChargePlan

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

Sequence Element Name	Sequence Element Type	Description
ChargeOrderType	TpDataSessionChargeOrder	Charge order
Currency	TpString	Currency unit according to ISO-4217:1995
AdditionalInfo	TpString	Descriptive string which is sent to the billing system without prior evaluation. Could be included in the ticket.

Valid Currencies are:

ADP, AED, AFA, ALL, AMD, ANG, AON, AOR, ARS, ATS, AUD, AWG, AZM, BAM,
BBD, BDT, BEF, BGL, BGN, BHD, BIF, BMD, BND, BOB, BOV, BRL, BSD, BTN,
BWP, BYB, BZD, CAD, CDF, CHF, CLF, CLP, CNY, COP, CRC, CUP, CVE, CYP,
CZK, DEM, DJF, DKK, DOP, DZD, ECS, ECV, EEK, EGP, ERN, ESP, ETB, EUR,
FIM, FJD, FKP, FRF, GBP, GEL, GHC, GIP, GMD, GNF, GRD, GTQ, GWP, GYD,
HKD, HNL, HRK, HTG, HUF, IDR, IEP, ILS, INR, IQD, IRR, ISK, ITL, JMD,
JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KYD, KZT, LAK, LBP, LKR,
LRD, LSL, LTL, LUF, LVL, LYD, MAD, MDL, MGF, MKD, MMK, MNT, MOP, MRO,
MTL, MUR, MVR, MWK, MXN, MXV, MYR, MZM, NAD, NGN, NIO, NLG, NOK, NPR,
NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PTE, PYG, QAR, ROL, RUB, RUR,
RWF, SAR, SBD, SCR, SDD, SEK, SGD, SHP, SIT, SKK, SLL, SOS, SRG, STD,
SVC, SYP, SZL, THB, TJR, TMM, TND, TOP, TPE, TRL, TTD, TWD, TZS, UAH,
UGX, USD, USN, USS, UYU, UZS, VEB, VND, VUV, WST, XAF, XAG, XAU, XBA,
XBB, XBC, XBD, XCD, XDR, XFO, XFU, XOF, XPD, XPF, XPT, XTS, XXX, YER,
YUM, ZAL, ZAR, ZMK, ZRN, ZWD.

XXX is used for transactions where no currency is involved.

8.5.2.6 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

8.5.2.7 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

8.5.2.8 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()

8.5.2.9 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.

8.5.2.10 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

8.5.2.11 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

8.5.2.12 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

8.5.2.13 TpDataSessionFault

Defines the cause of the data session fault detected.

Name	Value	Description
P_DATA_SESSION_FAULT_UNDEFINED	0	Undefined
P_DATA_SESION_USER_ABORTED	1	User has finalised the data session before any message could be sent by the application
P_DATA_SESSION_TIMEOUT_ON_RELEASE	2	This fault occurs when the final report has been sent to the application, but the application did not explicitly release data session object, within a specified time. The timer value is operator specific.
P_DATA_SESSION_TIMEOUT_ON_INTERRUPT	3	This fault occurs when the application did not instruct the gateway how to handle the call within a specified time, after the gateway reported an event that was requested by the application in interrupt mode. The timer value is operator specific.

8.5.2.14 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.

8.5.2.15 TpDataSessionSuperviseVolume

Defines the Sequence of Data Elements that specify the amount of volume that is allowed to be transmitted for the specific connection.

Sequence Element Name	Sequence Element Type	Sequence Element Description
VolumeQuantity	TpInt32	This data type is identical to a TpInt32, and defines the quantity of the granted volume that can be transmitted for the specific connection. The volume specifies the sum of uplink and downlink transfer data volumes.
VolumeUnit	TpInt32	In Order to enlarge the range of the volume quantity value the exponent of a scaling factor (10^VolumeUnit) is provided. When the unit is for example in kilobytes, VolumeUnit must be set to 3.

8.5.2.16 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.

8.5.2.17 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

8.5.2.18 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

8.5.2.19 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

8.5.2.20 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

8.5.2.21 TpDataSessionReportRequestSet

Defines a Numbered Set of Data Elements of TpDataSessionReportRequest.

8.5.2.22 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

8.5.2.23 TpDataSessionEventCriteriaResultSetRef

Defines a refernce to TpDataSessionEventCriteriaResultSet.

8.5.2.24 TpDataSessionEventCriteriaResultSet

Defines a set of TpDataSessionEventCriteriaResult.

8.5.2.25 TpDataSessionEventCriteriaResult

Defines a sequence of data elements that specify a requested call event notification criteria with the associated assignmentID.

Sequence Element Name	Sequence Element Type	Sequence Element Description
EventCriteria	TpDataSessionEventCriteria	The event criteria that were specified by the application.
AssignmentID	TpInt32	The associated assignementID. This can be used to disable the notification.

8.6 Network User Location and User Status Data definitions

8.6.1 Interface Definitions

8.6.1.1 IpAppUserStatus

Defines the address of an IpAppUserStatus Interface.

8.6.1.2 IpAppUserStatusRef

Defines a reference to type IpAppUserStatus.

8.6.1.3 IpUserStatus

Defines the address of an IpUserStatus Interface.

8.6.1.4 IpAppUserLocationCamel

Defines the address of an IpAppUserLocationCamel Interface.

8.6.1.5 IpAppUserLocationCamelRef

Defines a reference to type IpAppUserLocationCamelRef.

8.6.1.6 IpUserLocationCamel

Defines the address of an IpUserLocationCamel Interface.

8.6.2 Common Data Definitions for Network User Location and User Status

The constants and types defined in the following sections are defined in the *org.threegpp.osa.mm* package.

8.6.2.1 TpGeographicalPosition

Defines the structure of data elements that specify a geographical position.

An “ellipsoid point with uncertainty shape” defines the horizontal location. 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

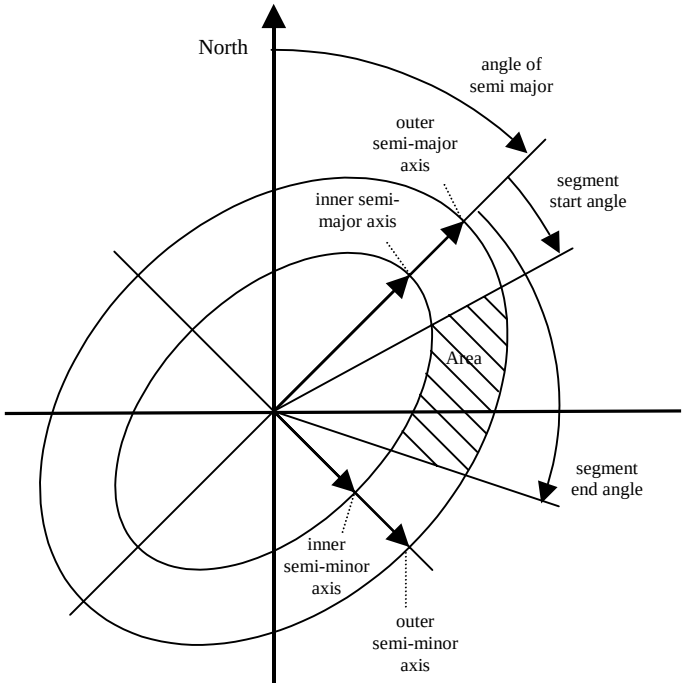


Figure 8-1: Description of an Ellipse Arc

Structured Member Name	Structured Member Type
Longitude	TpFloat
Latitude	TpFloat
TypeOfUncertaintyShape	TpLocationUncertaintyShape
UncertaintyInnerSemiMajor	TpFloat
UncertaintyOuterSemiMajor	TpFloat
UncertaintyInnerSemiMinor	TpFloat
UncertaintyOuterSemiMinor	TpFloat
AngleOfSemiMajor	TpInt32
SegmentStartAngle	TpInt32
SegmentEndAngle	TpInt32

8.6.2.2 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.

8.6.2.3 TpMobilityDiagnostic

Defines a diagnostic value that is reported in addition to an error by the Network User Location or User Status service capability feature.

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_SUPBY_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.

8.6.2.4 TpMobilityError

Defines an error that is reported by the Network User Location or User Status service capability feature. A fatal error occurring during the life of periodic or triggered user location/status requests (triggeredStatusReportErr,

triggeredLocationReportErr or periodicLocationReportErr) will terminate the request such that any particular request is allowed to generate at most one fatal error but possibly several non fatal errors.

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 Network User Location or User Status SCF 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 Network User Location SCF failed to obtain the user's position.	No

8.6.2.5 TpMobilityStopAssignmentData

Defines the structure of data elements that specifies a request to stop whole or parts of an assignment. Assignments are used for periodic or triggered reporting of a user locations or statuses.

Observe that the parameter "users" is optional. If the parameter "stopScope" is set to P_M_ALL_IN_ASSIGNMENT, the parameter "stopScope" is undefined. If the parameter "stopScope" is set to P_M_SPECIFIED_USERS, then the assignment shall be stopped only for the users specified in the "users" collection.

Structure Element Name	Structure 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 whole 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.

8.6.2.6 TpMobilityStopScope

This enumeration is used in requests to stop mobility reports that are sent from the Network User Location service capability feature 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 collection.

8.6.3 Network User Location Data Definitions

The constants and types defined in the following sections are defined in the *org.threegpp.osa.mm.ul* package.

8.6.3.1 TpLocationCellIDOrLAI

This data type is identical to a TString. It specifies the Cell Global Identification or the Location Area Identification (LAI).

The Cell Global Identification (CGI) is defined as the 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 for the detailed coding.

8.6.3.2 TpLocationTriggerCamel

Defines the structure of data elements that specifies the criteria for a triggered location report to be generated.

Structure Member Name	Structure Member Type	Description
`UpdateInsideVlr`	`TpBoolean`	Generate location report when it occurs an location update inside the current VLR area.
`UpdateOutsideVlr`	`TpBoolean`	Generate location report when the user moves to another VLR area.

8.6.3.3 TpUserLocationCamel

Defines the structure of data elements that specifies the location of a mobile telephony user. Observe that if the statusCode is indicating an error, then neither geographicalPosition, `timestamp, vlrNumber, locationNumber, cellIdOrLai nor their associated presense flags are defined.`

Structure Member Name	Structure Member 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 location information was attained .
`VlrNumberPresent`	`TpBoolean`	Flag indicating if the VLR number is present.
`VlrNumber`	`TpAddress`	Current VLR number for the user.
`VocationNumberPresent`	`TpBoolean`	Flag indicating if the location number is present.
`LocationNumber`[2]	`TpAddress`	Current location number.
`CellIdOrLaiPresent`	`TpBoolean`	Flag indicating if cell-id or LAI of the user is present.

[2] The location number is the number to the MSC or in rare cases the roaming number.

CellIdOrLai	TpLocationCellIDOrLAI	Cell-id or LAI of the user.

8.6.3.4 TpUserLocationCamelSet

Defines a collection of TUserLocationCamel

8.7 User Status Data Definitions

The constants and types defined in the following sections are defined in the *org.threegpp.osa.mm.us* package.

8.7.1.1 TpUserStatus

Defines the structure of data elements that specifies the identity and status of a user.

Structure Element Name	Structure Element Type	Description
UserID	TpAddress	The user address.
StatusCode	TpMobilityError	Indicator of error.
Status	TpUserStatusIndicator	The current status of the user.

8.7.1.2 TpUserStatusSet

Defines a collection of TUserStatus.

8.7.1.3 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[3]	2	User is busy (only applicable for interactive user status request, not when triggers are used)

8.8 Terminal Capabilities Data Definitions

8.8.1 Interface Definitions

8.8.1.1 IpTerminalCapabilities

Defines the address of an IpTerminalCapabilities Interface.

8.8.1.2 IpTerminalCapabilitiesRef

Defines a reference to type IpTerminalCapabilities

8.8.2 Terminal Capabilities Data Definitions

The constants and types defined in the following sections are defined in the *org.threegpp.osa.termcap* package.

[3] Only applicable to mobile (Wireless) telephony users.

8.8.2.1 terminalIdentity

Identifies the terminal.

Name	Type	Documentation
terminalIdentity	TpString	Identifies the terminal. It may be a logical address known by the WAP Gateway/PushProxy.

8.8.2.2 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 [6] and adopted in the WAP UAProf specification [9]. It contains URLs; terminal attributes and values, in RDF format; or a combination of both.

8.8.2.3 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.

9 IDL Interface Definitions

The OSA API definitions have been divided into several CORBA modules. The common data definitions are placed in the root module while each of the specific service capability feature API definitions are being assigned their own module directly under that root. Each specific SCF functions, like User Status, have their data and interface definitions collocated. This structure has the advantage that explicit scoping is kept to a minimum.

The IDLs defined for the specific SCFs assumes that the OSA common definitions (interfaces and data) are provided in the org.threegpp.osa module within a file name called OSA.idl

Module Name	Description	IDL file name
org.threegpp.osa	Common data/interface definitions	OSA.idl
org.threegpp.osa.fw	common Framework data-types	FW.idl
org.threegpp.osa.fw.discovery	Discovery data-types and interfaces	DISC.idl
org.threegpp.osa.fw.trust_and_security	Trust and Security date-types and interfaces	TandS.idl
org.threegpp.osa.fw.integrity	Integrity management data-types and interfaces	IM.idl
org.threegpp.osa.fw.registration	Registration data-types and interfaces	REG.idl
org.threegpp.osa.cc	Call Control data-types	CC.idl
org.threegpp.osa.cc.gcc	Generic Call Control interfaces	GCC.idl
org.threegpp.osa.cc.ecc	data-types and interfaces specific for Enhanced Call Control. This is only needed to compile the User Interaction IDL	ECC.idl
org.threegpp.osa.ui	User Interaction data-types	UI.idl
org.threegpp.osa.ui.gui	User Interaction interfaces	GUI.idl
org.threegpp.osa.dsc	Data Session data-types and interfaces	DSC.idl
org.threegpp.osa.mm	Common mobility data definitions (root)	MM.idl
org.threegpp.osa.mm.ul	Network User Location (UL)	MMul.idl
org.threegpp.osa.mm.us	User Status (US)	MMus.idl
org.threegpp.osa.termcap	Terminal Capabilities	TERMCAP.idl

Some of the interfaces contain more operations than defined in the interface classes of Chapter 6. These operations must return a “Method not supported” exception in case the interface is implemented on a SCS based on this specification.

9.1 Generic IDL

```
#ifndef __OSA_DEFINED
#define __OSA_DEFINED

module org
{
    module threegpp
    {
        module osa
        {
            /*******************************************************************************/
            //                              Primitive data types
            /*******************************************************************************/

            typedef boolean   TpBoolean;   // Defines a Boolean data type
            typedef long      TpInt32;     // Defines a signed 32 bit integer
            typedef float     TpFloat;     // Defines a single precision real number.
            typedef string    TpString;    // Defines a string comprising length and data.

            // Primitive based OSA datatypes

            typedef TpInt32    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
default
                                            // duration.
            typedef TpInt32   TpSessionID;  // Defines a network unique session ID. OSA
                                            // uses this ID to identify sessions, e.g. call or call
leg
                                            // sessions, within an object implementing an interface
                                            // capable of handling multiple sessions. For the
different
                                            // OSA service capability feature, the sessionIDs are
unique
```

```
                                        // only in the context of a manager instantiation (e.g.,
within
                                        // the context of one generic call control manager). As
such
                                        // if an application creates two instances of the same
SCF
                                        // manager it shall use different instantiations of the
                                        // callback objects which implement the callback
interfaces.
            typedef TpInt32 TpAssignmentID; // This data type is identical to a TpInt32. It
                                        // specifies a number which identifies an individual
                                        // event notification enabled by the application or
                                        // OSA service capability feature.

            typedef sequence < TpSessionID> TpSessionIDSet;

      exception TpGeneralException
    {
      TpInt32 exceptionType;

    };

      const TpInt32 P_RESULT_INFO_UNDEFINED = 0;
      const TpInt32 P_INVALID_DOMAIN_ID = 1;
      const TpInt32 P_INVALID_AUTH_CAPABILITY = 2;
      const TpInt32 P_INVALID_AGREEMENT_TEXT = 3;
      const TpInt32 P_INVALID_SIGNING_ALGORITHM = 4;
      const TpInt32 P_INVALID_INTERFACE_NAME = 5;
      const TpInt32 P_INVALID_SERVICE_ID = 6;
      const TpInt32 P_INVALID_EVENT_TYPE = 7;
      const TpInt32 P_SERVICE_NOT_ENABLED = 8;
      const TpInt32 P_INVALID_ASSIGNMENT_ID = 9;
      const TpInt32 P_INVALID_PARAMETER = 10;
      const TpInt32 P_INVALID_PARAMETER_VALUE = 11;
      const TpInt32 P_PARAMETER_MISSING = 12;
      const TpInt32 P_RESOURCES_UNAVAILABLE = 13;
      const TpInt32 P_TASK_REFUSED = 14;
      const TpInt32 P_TASK_CANCELLED = 15;
      const TpInt32 P_INVALID_DATE_TIME_FORMAT = 16;
      const TpInt32 P_NO_CALLBACK_ADDRESS_SET = 17;
      const TpInt32 P_INVALID_SIGNATURE = 18;
      const TpInt32 P_INVALID_SERVICE_TOKEN = 19;
      const TpInt32 P_ACCESS_DENIED = 20;
      const TpInt32 P_INVALID_PROPERTY = 21;
      const TpInt32 P_METHOD_NOT_SUPPORTED = 22;
      const TpInt32 P_NO_ACCEPTABLE_AUTH_CAPABILITY = 23;
      const TpInt32 P_INVALID_INTERFACE_TYPE = 24;
      const TpInt32 P_SERVICE_ACCESS_TYPE = 25;
      const TpInt32 P_SERVICE_ACCESS_DENIED = 26;
      const TpInt32 P_USER_NOT_SUBSCRIBED = 48;
      const TpInt32 P_APPLICATION_NOT_ACTIVATED = 49;
      const TpInt32 P_USER_PRIVACY = 50;


            /*************************************************************************/
            /******************** Date and Time related data definitions *************/
            /*************************************************************************/

            // This data type is identical to a TpString. It specifies the data in
            // accordance with International Standard ISO 8601. This is defined as the
            // string of characters in the following format:
            //      YYYY-MM-DD
            // where the date is specified as:
            //      YYYY    four digits year
            //      MM      two digits month
            //      DD      two digits day
            // The date elements are separated by a hyphen character (-).
            typedef TpString TpDate;

            // This data type is identical to a TpString. It specifies the time in accordance
            // with International Standard ISO 8601. This is defined as the string of
            // characters in the following format:
            //      HH:MM:SS.mmm
            // or
            //      HH:MM:SS.mmmZ
            // where the time is specified as:
            //      HH  two digits hours (24h notation)
            //      MM  two digits minutes
            //      SS  two digits seconds
            //      mmm three digits fractions of a second (i.e. milliseconds)
            // The time elements are separated by a colon character (:).The date and time
            // are separated by a space. Optionally, a capital letter Z may be appended
            // to the time field to indicate Universal Time (UTC). Otherwise, local time
            // is assumed.
            typedef TpString TpTime;

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

```
//
//         YYYY-MM-DD HH:MM:SS.mmm
//     or  YYYY-MM-DD HH:MM:SS.mmmZ
//
// 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
typedef TpString TpDateAndTime;

/*****************************************************************************/
//                      Address related data definitons
/*****************************************************************************/

// Defines whether an address can be presented to an end user
enum TpAddressPresentation
{
    P_ADDRESS_PRESENTATION_UNDEFINED,            // Undefined
    P_ADDRESS_PRESENTATION_ALLOWED,              // Presentation Allowed
    P_ADDRESS_PRESENTATION_RESTRICTED,           // Presentation Restricted
    P_ADDRESS_PRESENTATION_ADDRESS_NOT_AVAILABLE // Address not available for
                                                 // presentation
};

// Defines whether an address has been screened by the application
enum TpAddressScreening
{
    P_ADDRESS_SCREENING_UNDEFINED,            // Undefined
    P_ADDRESS_SCREENING_USER_VERIFIED_PASSED, // user provided address verified
                                              // and passed
    P_ADDRESS_SCREENING_USER_NOT_VERIFIED,    // user provided address not verified
    P_ADDRESS_SCREENING_USER_VERIFIED_FAILED, // user provided address verified and
                                              // failed
    P_ADDRESS_SCREENING_NETWORK               // Network provided address
};

// Defines the address plan (or numbering plan) used. It is also used to indicate
// whether an address is actually defined in a TAddress data element
enum TpAddressPlan
{
    P_ADDRESS_PLAN_NOT_PRESENT,  // No Address Present
    P_ADDRESS_PLAN_UNDEFINED,    // Undefined
    P_ADDRESS_PLAN_IP,           // IP
    P_ADDRESS_PLAN_MULTICAST,    // Multicast
    P_ADDRESS_PLAN_UNICAST,      // Unicast
    P_ADDRESS_PLAN_E164,         // E.164
    P_ADDRESS_PLAN_AESA,         // AESA
    P_ADDRESS_PLAN_URL,          // URL
    P_ADDRESS_PLAN_NSAP,         // NSAP
    P_ADDRESS_PLAN_SMTP,         // SMTP
    P_ADDRESS_PLAN_NOT_USED,
    P_ADDRESS_PLAN_X400          // X.400
};

// Defines the reasons why an address is invalid.
enum TpAddressError
{
    P_ADDRESS_INVALID_UNDEFINED,       // Undefined error
    P_ADDRESS_INVALID_MISSING,         // Mandatory address not present
    P_ADDRESS_INVALID_MISSING_ELEMENT, // Mandatory address element not present
    P_ADDRESS_INVALID_OUT_OF_RANGE,    // Address is outside of the valid range
    P_ADDRESS_INVALID_INCOMPLETE,      // Address is incomplete
    P_ADDRESS_INVALID_CANNOT_DECODE    // Address cannot be decoded
};

// Defines the structure of data elements that specifies an address
struct TpAddress
{
    TpAddressPlan         plan;
    TpString              astring;
    TpString              name;
    TpAddressPresentation presentation;
    TpAddressScreening    screening;
    TpString              subAddressString;
};

// Defined a collection of TpAddress elements
typedef sequence < TpAddress> TpAddressSet;

// Defined a collection of TpAddress elements
typedef TpAddress TpAddressRange;

// This data type is identical to a TpString and contains a URL address.
typedef TpString TpURL;

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

```
            typedef TpString TpPrice;


struct TpChargePerTime {
    TpInt32 InitialCharge; /*Initial charge amount (in currency units * 0.0001)*/
TpInt32 CurrentChargePerMinute; /* Current tariff (in currency units * 0.0001)*/
TpInt32 NextChargePerMinute; /* Next tariff (in currency units * 0.0001) after tariff switch
Only used in setAdviceOfCharge()*/
};

enum TpAoCOrderCategory {

P_CHARGE_ADVICE_INFO,   /* Set of GSM Charge Advice Information elements according to 3GPP TS
22.024*/
P_CHARGE_PER_TIME,  /* Charge per time*/
P_CHARGE_NETWORK    /* Operator specific charge plan specification, e.g. charging table name /
charging table entry*/
};

/* Defines the Sequence of Data Elements that specify theCharging Advice Information elements
according to 3GPP TS 22.024.*/
struct TpCAIElements {

TpInt32 UnitsPerInterval;       /* Units per interval */
TpInt32 SecondsPerTimeInterval;     /* Seconds per time interval  */
TpInt32 ScalingFactor;      /* Scaling factor */
TpInt32 UnitIncrement;      /* Unit increment */
TpInt32 UnitsPerDataInterval;       /* Units per data interval */
TpInt32 SegmentsPerDataInteral;     /* Segments per data interal */
TpInt32 InitialSecsPerTimeInterval; /* Initial secs per time interval */
};

struct TpChargeAdviceInfo {
        TpCAIElements CurrentCAI ; /* Current tariff*/
        TpCAIElements NextCAI ; /* Next tariff after tariff switch*/
};

/* Defines the Tagged Choice of Data Elements that specify the charge plan */
    union TpAoCOrder switch(TpAoCOrderCategory) {
        case P_CHARGE_ADVICE_INFO:
            TpChargeAdviceInfo ChargeAdviceInfo;
        case P_CHARGE_PER_TIME:
            TpChargePerTime ChargePerTime;
        case P_CHARGE_NETWORK:
            TpString NetworkCharge;
    };
struct TpAoCInfo {
        TpAoCOrder ChargeOrderType; /* Charge order*/
        TpString Currency; /* Currency unit according to ISO-4217:1995*/
};


            /************************************************************************/
            //                          base OSA interface
            /************************************************************************/

            // All application, framework and service capability features interfaces inherit
            // from the following interface. This API Base Interface does not provide any
            // additional methods.
            interface IpOsa
            {
            };

            // All service capability feature interfaces inherit from the following interface.
            interface IpService : IpOsa
            {
                // This method specifies the reference address of the callback interface
                // that a SCF uses to invoke methods on the application.
                void setCallback(in IpOsa appInterface) raises(TpGeneralException);
                void setCallbackWithSessionID(in IpOsa appInterface, in TpSessionID sessionID)
raises(TpGeneralException);
            };
        };
    };
};

#endif
```

9.2 Framework IDL

9.2.1 Common Data Types for the Framework

```
#include <OSA.idl>
```

```
module org{

module threegpp{

module osa{

module fw{


typedef TpString     TpClientAppID;        // Identifies the client appl to the framework.

typedef sequence     <TpClientAppID> TpClientAppIDList;
            /* Defines either the framework or the type of entity attempting to access the framework
            The framework
            A client application
            An enterprise operator
            A registered service
            A service supplier */
            enum TpDomainIDType
            {
                P_FW,
                P_CLIENT_APPLICATION,
                P_ENT_OP,
                P_REGISTERED_SERVICE,
                P_SERVICE_SUPPLIER
            };

typedef TpString TpEntOpID;

typedef sequence < TpEntOpID >    TpEntOpIDList;

            typedef TpString TpFwID;

            typedef TpString TpServiceSupplierID;

            /* 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 */

            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;
            };

typedef TpString TpPropertyName;

typedef TpString TpPropertyValue;

typedef sequence < TpProperty > TpPropertyList;

       struct TpProperty {
    TpPropertyName      PropertyName;
    TpPropertyValue        PropertyValue;
      };
typedef  TpString   TpServiceID;    // A string of characters, generated automatically by the
// Framework and comprising a TpUniqueServiceNumber,
// TpServiceNameString, and a number of relevant
// TpServiceSpecString, concatenated using a forward
// separator (/), that uniquely identifies an instance of a
// SCF interface.

typedef sequence <TpServiceID>      TpServiceIDList;

 typedef TpString       TpServiceNameString;        // Uniquely identifies the name of an SCF
// interface. For OSA release 99 the following
// values have been defined: NULL (no SCF name),
// P_CALL_CONTROL, P_USER_INTERACTION,
// P_USER_LOCATION_CAMEL, P_TERMINAL_CAPABILITIES and
// P_USER_STATUS.

typedef TpString        TpServiceSpecString;        // Uniquely identifies the name of a SCF
// specialization interface. For OSA release 99
// the following values have been defined: NULL
// no SCF specialization) and P_CALL.
```

```
typedef TpString        TpUniqueServiceNumber;      // A string of characters that represents a
// unique number.
    enum TpServicePropertyMode {
        NORMAL,                     // The value of the corresponding SCF property type may
optionally be
// provided.
MANDATORY,                  // The value of the corresponding SCF property type must be provided
// at SCF registration.
        _READONLY,                  // The value of the corresponding SCF property is optional, nut
once
// given a value it may not be modified.
        MANDATORY_READONLY      // The value of the corresponding SCF property type must be provided
// and may not be modified subsequently.
    };

typedef TpString        TpServicePropertyTypeName;

typedef TpString        TpServicePropertyName;

typedef sequence <TpServicePropertyName>    TpServicePropertyNameList;

typedef TpString        TpServicePropertyValue;

typedef sequence <TpServicePropertyValue>   TpServicePropertyValueList;

    struct TpServiceProperty {              // Describes a SCF property
    TpServicePropertyName       ServicePropertyName;
    TpServicePropertyValueList  ServicePropertyValueList;
    TpServicePropertyMode       ServicePropertyMode;
};

typedef sequence <TpServiceProperty>        TpServicePropertyList;

typedef TpString        TpServiceTypeName;

typedef sequence <TpServiceTypeName>        TpServiceTypeNameList;

struct  TpService {                 // Describes a registered SCF.
        TpServiceID     ServiceID;
        TpServicePropertyList   ServicePropertyList;
    };

typedef sequence <TpService>    TpServiceList;

    struct TpServiceDescription {           // Describes the properties of a registered SCF.
        TpServiceTypeName   ServiceTypeName;
        TpServicePropertyList   ServicePropertyList;
    };

struct TpServiceTypeProperty {                  // Describes a SCF property.
        TpServicePropertyName               ServicePropertyName;
        TpServicePropertyMode       ServicePropertyMode;
        TpServicePropertyTypeName   ServicePropertyTypeName;
    };

typedef sequence <TpServiceTypeProperty>    TpServiceTypePropertyList;

    struct TpServiceTypeDescription {           // Describes a SCF type.
        TpServiceTypePropertyList           ServiceTypePropertyList;
        TpServiceTypeNameList       ServiceTypeNameList;
        TpBoolean                       EnabledOrDisabled;
    };

};};};};
```

9.2.2 Service Discovery IDL

```
#include <fw.idl>

module org{
module threegpp{
module osa{
module fw{
module discovery{

/*******************************************************************************/
//                              Interface definitions                          //
/*******************************************************************************/

/* The Service Discovery Framework interface is used by the client application to
know what types of services are supported by the Framework, and what are their
properties; and to obtain the services its subscription allows access to. */
interface IpServiceDiscovery : IpOsa {

  /* This method is invoked by the client application to obtain the names of all service
    types that are in the Framework repository. */
```

```
  void listServiceTypes (
  out TpServiceTypeNameList listTypes    // The names of the requested service types.
  ) raises (TpGeneralException);

  /* This method is invoked by the client application to obtain the detailed description of
  a particular service type. */
  void describeServiceType (
  in TpServiceTypeName name,                                    // Identifies the service
// type to be described.
  out TpServiceTypeDescription serviceTypeDescription           // Describes the specified
// service type.
  ) raises (TpGeneralException);

  /* This method is invoked by the client application to obtain the IDs of the services
  that meet its requirements. */
  void discoverService (
  in TpServiceTypeName serviceTypeName,            // Type of the required service.
  in TpServicePropertyList desiredPropertyList,    // Properties that the discovered  set
// of SCFs should satisfy.
  in TpInt32 max,                                  // Maximum number of SCFs that are
// to be returned.
  out TpServiceList serviceList                    // A list of matching SCFs.

  ) raises (TpGeneralException);

  /* This method is invoked by the client application to obtain a list of subscribed
  SCFs that they are allowed to access. */
  void listSubscribedServices (
  out TpServiceList serviceList      // A list of subscribed SCFs.
  ) raises (TpGeneralException);

};

};};};};};
```

9.2.3 Trust and Security Management IDL

```
#include <fw.idl>

module org{
module threegpp{
module osa{
module fw{
module trust_and_security{

/*******************************************************************************/
//                                Data definitions                             //
/*******************************************************************************/

    typedef TpString        TpAccessType;       // The type of access interface requested by the
client
// application. For OSA release 99 the following value
// has been defined: P_ACCESS.

    typedef TpString        TpAuthType;         // The type of authentication mechanism requested by
the
// client. For OSA release 99 the following values has
// been defined:
// P_AUTHENTICATION (indicates use of the OSA
// authentication interfaces).

    typedef TpString    TpAuthCapability;       // The authentication capabilities that could be
supported
// by the OSA. For OSA release 99 the following values
// have been defined: NULL (indicates no client
// capabilities, P_DES_56, P_DES_128, P_RSA_512 and P_RSA_1024).

    typedef TpString        TpAuthCapabilityList;   // A string of multiple TpAuthCapability
// concatenated using a commas.
    struct TpAuthDomain
            {
                TpDomainID DomainID;
       IpOSA AuthInterface;
            };

typedef TpPropertyList TpEndAccessProperties;

    typedef TpString        TpInterfaceName;    // Identifies the names of the framework SCFs that
are to be
// supported by the OSA API. For release 99 these are,
// P_DISCOVERY, P_OAM
// P_LOAD_MANAGER,
// P_FAULT_MANAGER,
// P_HEARTBEAT_MANAGEMENT,
// P_REGISTRATION

    struct TpServiceAccessControl {
```

```
        TpString        Policy;         // Access control policy information controlling access to
the
// service feature.
        TpString        TrustLevel;     // The level of trust that the network operator has assigned
to the
// client application.
    };

    typedef TpString        TpServiceToken; // Uniquely identifies a SCF.

    struct TpSignatureAndServiceMgrRef {
        TpString        DigitalSignature;       // The digital signature of the Framework for the
service
// agreement.
        IpOsa       ServiceMgrInterface;
    };

    typedef TpString        TpSigningAlgorithm;     // Identifies the signing algorithm that must be
// used. For OSA release 99 the follwing values have
// been defined: NULL (indicates no signing algorithm
// is required), P_MD5_RSA_512 and P_MD5_RSA_1024.

    typedef TpString        TpFwID;

    struct TpFwAuth {
        TpFwID  FwID;
        IpOsa   FwAuthInterface;
    };

/*******************************************************************************/
//                                  Interface definitions                      //
/*******************************************************************************/

/* The Initial Framework interface is used by the client application to initiate the mutual
authentication with the Framework and, when this is finished successfully, to request access
to it. */
interface IpInitial : IpOsa {

/* 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.
*/
void initiateAuthentication (
in TpAuthDomain appDomain,          // Identifies the client to the framework.
in TpAuthType authType,                 // Allows the client application to request a
// specific type of authentication mechanism.

out TpAuthDomain fwDomain          // Provides a framework identifier, and a reference
// to framework authentication interface.
) raises (TpGeneralException);

/* This method is invoked by the client application, once mutual authentication is
achieved, to request access to the framework and specify the type of access desired. */
void requestAccess (
in TpAccessType accessType,         // Identifies the type of access interface requested by
// the client application.
in IpOsa appAccessInterface,        // Provides a reference to the access interface of the
// client application.
out IpOsa fwAccessInterface          // Provides a reference to call the access interface of
 // the framework.
) raises (TpGeneralException);

};


/* The Access Framework interface is used by the client application to perform the mechanisms
necessary for it to obtain access to SCFs. */
interface IpAccess : IpOsa {

/* This method is invoked by the client application to obtain interface references to other
framework interfaces. */
void obtainInterface (
in TpInterfaceName interfaceName,   // The name of the framework interface to which a
// reference to the interface is requested.
out IpOsa fwInterface                   // The requested interface reference.
) raises (TpGeneralException);

/* This method is invoked by the client application to obtain interface references to other
framework interfaces, when it is required to supply a callback interface to the framework. */
void obtainInterfaceWithCallback (
in TpInterfaceName interfaceName,           // The name of the framework interface to which
// a reference to the interface is requested.
in IpOsa appInterface,                      // This is the reference to the client application
// interface which is used for callbacks.
out IpOsa fwInterface                           // The requested interface reference.
) raises (TpGeneralException);

/* This method may be invoked by the client application to check whether it has been
granted permission to access the specified SCF and, if granted, the level of trust that
will be applied. */
```

```
void accessCheck (
in TpServiceToken serviceToken,
in TpString securityContext,                              // A group of security relevant
// attributes.
in TpString securityDomain,                               // The security domain in which
// the client application is
// operating.
in TpString group,                                        // Used to define the access
// rights associated with all
// clients that belong to that
// group.
in TpString serviceAccessTypes,                           // Defined by the specific
// security model in use.
out TpServiceAccessControl serviceAccessControl           // 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.
) raises (TpGeneralException);

/* This method is invoked by the client application to identify the SCF that it wishes
to use. */
void selectService (
in TpServiceID serviceID,                        // Identifies the SCF.
out TpServiceToken serviceToken                  // A free format text token returned by
// the framework, which can be signed as
// part of a service agreement.
) raises (TpGeneralException);

/* This method is invoked by the client application to request that the framework sign an
agreement on the SCF, which allows the client application to use the SCF. */
void signServiceAgreement (
in TpServiceToken serviceToken,                               // Used to identify the SCF
// instance requested by the
// client application.
in TpString agreementText,                                    // The agreement text to be
// signed by the framework.
in TpSigningAlgorithm signingAlgorithm,                       // The algorithm used to compute
// the digital signature.
out TpSignatureAndServiceMgrRef signatureAndServiceMgr        // A reference to a structure
// that contains the digital
// signature of the framework
// for the service agreement,
// and a reference to the
// SCF manager interface of
// the SCF.
) raises (TpGeneralException);

/* This method is invoked by the client application to terminate an agreement for the
specified SCF. */
void terminateServiceAgreement (
in TpServiceToken serviceToken,     // Identifies the service agreement to be terminated.
in TpString terminationText,        // Describes the reason for the termination of the
// service agreement.
in TpString digitalSignature        // Used by the framework to check that the
// terminationText has been signed by the client.
) raises (TpGeneralException);

/* This method is invoked by the client application to end the access session
with the Framework. */
void endAccess () raises (TpGeneralException);

};



/* The Access client application interface is used by the Framework to perform the steps that
are necessary in order to allow it to SCF access. */
interface IpAppAccess : IpOsa {

/* This method is invoked by the Framework to request that client application sign an
agreement on a specified SCF. */
void signServiceAgreement (
in TpServiceToken serviceToken,             // Identifies the SCF instance to which
 // this service agreement corresponds.
in TpString agreementText,                  // Agreement text that has to be signed by the
 // client application.
in TpSigningAlgorithm signingAlgorithm,     // Algorithm used to compute the digital
 // signature.
out TpString digitalSignature               // Signed version of a hash of the service
// token and agreement text given by the
// framework.
) raises (TpGeneralException);

/* This method is invoked by the Framework to terminate an agreement for a specified
SCF. */
void terminateServiceAgreement (
```

```
in TpServiceToken serviceToken,         // Identifies the SCF agreement to be terminated.
in TpString terminationText,       // Describes the reason for the termination.
in TpString digitalSignature        // Used by the Framework to confirm its identity to the
// client.
) raises (TpGeneralException);

/* This method is invoked by the Framework to end the client application's access session
with the framework. */
void terminateAccess (
in TpString terminationText,               // Describes the reason for the termination of
 // the access session.
in TpSigningAlgorithm signingAlgorithm,     // The algorithm used to compute the digital
// signature.
in TpString digitalSignature               // Used by the Framework to confirm its
// identity to the client.
) raises (TpGeneralException);

};



/* The 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. */
interface IpAuthentication : IpOsa {

/* This method is invoked by the client application to start the authentication process,
informed the Framework of the authentication mechanisms it supports, and be informed by its
of its preferred choice. */
void selectAuthMethod (
in TpAuthCapabilityList auths,      // Informs the Framework of the authentication
// mechanisms supported by the client
// application.
out TpAuthCapability prescribedMethod           // Indicates the mechanism preferred by the
// framework.
) raises (TpGeneralException);

/* This method is invoked by the client application to authenticate the framework using the
mechanism indicated in the parameter prescribedMethod. */
void authenticate (
in TpAuthCapability prescribedMethod,       // Specifies the method accepted by that the
// framework for authentication.
in TpString challenge,                    // The challenge presented by the client
// application to be responded to by the
// framework.
out TpString response                       // The response of the framework to the
// challenge of the client application.
) raises (TpGeneralException);

/* This method is invoked by the client application to to abort the authentication
process.*/
void abortAuthentication() raises (TpGeneralException);

};



/* The Authentication client application interface is used by the Framework to authenticate

the client application. */

interface IpAppAuthentication : IpOsa {

/* This method is invoked by the Framework to authenticate the client application using the
mechanism indicated in prescribedMethod. */
void authenticate (
in TpAuthCapability prescribedMethod,       // The agreed authentication method.
in TpString challenge,                          // The challenge presented by the Framework.
out TpString response
) raises (TpGeneralException);

/* This method is invoked by the Framework to abort the authentication process. */
void abortAuthentication() raises (TpGeneralException);

};

};};};};};
```

9.2.4 Integrity Management IDL

```
#include <fw.idl>

module org{
module threegpp{
module osa{
module fw{
module integrity{
```

```
/*******************************************************************************/
//                                Data definitions                             //
/*******************************************************************************/

typedef TpString        TpActivityTestRes;      // An implementation specific result, whose values
// are Framework provider specific.

    struct TpTimeInterval {         // A time interval.
        TpDateAndTime       StartTime;
        TpDateAndTime       StopTime;
    };


    enum TpInterfaceFault {         // The cause of the interface fault detected.
        INTERFACE_FAULT_UNDEFINED,              // Undefined.
        INTERFACE_FAULT_LOCAL_FAILURE,      // A fault in the local API software or hardware has
been
// detected.
        INTERFACE_FAULT_GATEWAY_FAILURE,        // A fault in the gateway API software or hardware
has been
// detected.
        INTERFACE_FAULT_PROTOCOL_ERROR      // An error in the protocol used on the client-gateway
link
// has been detected.
    };

    struct TpFaultStats {           // Statistics on a per fault type basis.
        TpInterfaceFault    Fault;
        TpInt32         Occurrences;                        // The number of separate instances of
this fault
// during the period.
        TpInt32         MaxDuration;                        // The duration in seconds of the
longest fault.
        TpInt32         TotalDuration;                      // The cumulative total during the
period.
        TpInt32         NumberOfClientsAffected;        // Those informed of the fault by the
Framework.
    };
            typedef sequence <TpFaultStats> TpFaultStatsSet;

    struct TpFaultStatsRecord {     // The set of fault information records to be returned for the
// requested time period.
        TpTimeInterval      Period;
        TpFaultStatsSet FaultRecords;
    };

typedef TpInt32         TpActivityTestID;           // Used as a token to match activity test
requests
// with their results.

    enum TpSvcUnavailReason {               // The reason why a SCF is unavailable.
        SERVICE_UNAVAILABLE_UNDEFINED,          // Undefined.
        SERVICE_UNAVAILABLE_LOCAL_FAILURE,      // The local API software or hardware has failed.
        SERVICE_UNAVAILABLE_GATEWAY_FAILURE,    // The gateway API software or hardware has failed.
        SERVICE_UNAVAILABLE_OVERLOADED,         // The SCF is fully overloaded.
        SERVICE_UNAVAILABLE_CLOSED                  // The SCF has closed itself.
    };

    enum TpAPIUnavailReason {           // The reason why the API is unavailable.
        API_UNAVAILABLE_UNDEFINED,              // Undefined.
        API_UNAVAILABLE_LOCAL_FAILURE,      // The local API software or hardware has failed.
        API_UNAVAILABLE_GATEWAY_FAILURE,        // The gateway API software or hardware has failed.
        API_UNAVAILABLE_OVERLOADED,         // The gateway is fully overloaded.
        API_UNAVAILABLE_CLOSED,                 // The gateway has closed itself.
        API_UNAVAILABLE_PROTOCOL_FAILURE        // The protocol used on the client-gateway link has
failed.
    };


    enum TpLoadLevel {          // The load level values.
        LOAD_LEVEL_NORMAL,              // Normal load.
        LOAD_LEVEL_OVERLOAD,                // Overload.
        LOAD_LEVEL_SEVERE_OVERLOAD      // Severe overload.
    };

    struct TpLoadThreshold{     // The load threshold value.
    TpFloat     LoadThreshold;
};


    struct TpLoadInitVal {  // The pair of load level and associated load threshold  values.
        TpLoadLevel     LoadLevel;
        TpLoadThreshold LoadThreshold;
    };

    struct TpLoadPolicy {       // The load balancing policy.
        TpString        LoadPolicy;
    };
```

```
    enum TpLoadStatisticEntityType {
        P_LOAD_STATISTICS_FW_TYPE,
        P_LOAD_STATISTICS_SVC_TYPE,
        P_LOAD_STATISTICS_APP_TYPE
    };

                union TpLoadStatisticEntityID switch(TpLoadStatisticEntityType)
                {
                    case P_LOAD_STATITICS_FW_TYPE:
                    TpFwID FrameworkID;
                    case P_LOAD_STATITICS_SVC_TYPE:
                    TpServiceID ServiceID;
                    case P_LOAD_STATITICS_APP_TYPE:
                    TpClientAppID ClientAppID;
                };

       struct TpLoadStatisticData {
        TpFloat         LoadValue;      // Expressed in percentage.
        TpLoadLevel     LoadLevel;
    };

       enum TpLoadStatisticError {
        P_LOAD_INFO_ERROR_UNDEFINED,
        P_LOAD_INFO_UNAVAILABLE
    };

    enum TpLoadStatisticInfoType {
        P_LOAD_STATISTICS_VALID,
        P_LOAD_STATISTICS_INVALID
    };

                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;
/*******************************************************************************/
//                                  Interface definitions                      //
/*******************************************************************************/

/* The Heartbeat Framework interface is used by the client application to supervise the
Framework or a SCF. */
interface IpHeartBeat : IpOsa {

/* This method is invoked by the client application to make the service or Framework
supervision. */
void send (
in TpSessionID session          // The heartbeat session.
) raises (TpGeneralException);

};


/* The Heartbeat client application interface is used by the Framework to supervise the client
application. */
interface IpAppHeartBeat : IpOsa {

/* This method is invoked by the Framework to make the client application supervision. */
void send (
in TpSessionID session          // The heartbeat session.
) raises (TpGeneralException);

};


/* The Heartbeat Management Framework interface is used by the client application to
initialise a heartbeat supervision of the client application. */
interface IpHeartBeatMgmt : IpOsa {

/* This method is invoked by the client application to register at the Framework for
heartbeat supervision. */
void enableHeartBeat (
in TpDuration duration,                 // Duration in milliseconds between heartbeats.
in IpAppHeartBeat appInterface,         // The callback interface the heartbeat is calling.
out TpSessionID session                 // The heartbeat session.
) raises (TpGeneralException);
```

```
/* This method is invoked by the client application to stop its heartbeat supervision. */
void disableHeartBeat (
in TpSessionID session          // The heartbeat session.
) raises (TpGeneralException);

/* This method is invoked by the client application to change the heartbeat period. */
void changeTimePeriod (
in TpDuration duration,         // Duration in milliseconds between heartbeats.
in TpSessionID session          // The heartbeat session.
) raises (TpGeneralException);

};



/* The Heartbeat Management client application interface is used by the Framework to
initialise its heartbeat supervision of the Framework. */
interface IpAppHeartBeatMgmt : IpOsa {

/* This method is invoked by the Framework to register at the client application for its
heartbeat supervision. */
void enableAppHeartBeat (
in TpDuration duration,             // Time interval in milliseconds between the heartbeats.
in IpHeartBeat fwInterface,     // The callback interface the heartbeat is calling.
in TpSessionID session              // The heartbeat session.
) raises (TpGeneralException);

/* This method is invoked by the Framework to stop the heartbeat supervision by the
application. */
void disableAppHeartBeat (
in TpSessionID session              // The heartbeat session.
) raises (TpGeneralException);

/* This method is invoked by the Framework to change the heartbeat period. */
void changeTimePeriod (
in TpDuration duration,             // Interval in milliseconds between the heartbeats.
in TpSessionID session              // The heartbeat session.
) raises (TpGeneralException);

};


/* The Load Manager Framework interface is used by the client application for load balancing
management. */
interface IpLoadManager : IpOsa {

/* This method is invoked by the client application to notify framework its current load
level (0,1, or 2) when the load level on the application has changed. */
void reportLoad (

in TpLoadLevel loadLevel        // The application's load level.
) raises (TpGeneralException);

/* This method is invoked by the client application to request load statistic records for
the framework and specified SCFs. */
void queryLoadReq (

in TpServiceIDList serviceIDs,      // Specifies the framework and SCFs for which the
// load statistics shall be reported.
in TpTimeInterval timeInterval      // The time interval within which the load statistics
// are generated.
) raises (TpGeneralException);

/* This method is invoked by the client application to report load statistics back to the
framework that requested the information. */
void queryAppLoadRes (
in TpLoadStatisticList loadStatistics       // The application's load statistics.
) raises (TpGeneralException);

/* This method is invoked by the client application to return an error response to the
framework that requested the application's load statistics information. */
void queryAppLoadErr (
in TpLoadStatisticErrorList loadStatisticsError     // The error code associated with the
// failed attempt to retrieve the
// application's load statistics.
) raises (TpGeneralException);

/* This method is invoked by the client application to register the client application for
load management under various load conditions. */
void registerLoadController (

in TpServiceIDList serviceIDs           // Specifies the framework and SCFs to be
// registered for load control.
) raises (TpGeneralException);

/* This method is invoked by the client application to unregister for load management. */
void unregisterLoadController (
```

```
in TpServiceIDList serviceIDs       // Specifies the framework or SCFs to be
// unregistered for load control.
) raises (TpGeneralException);

/* This method is invoked by the client application to resume load management notifications
to it from the framework and specified SCFs. */
void resumeNotification (
in TpServiceIDList serviceIDs       // Specifies the framework and SCFs for which
// notifications are to be resumed.
) raises (TpGeneralException);

/* This method is invoked by the client application to suspend load management
notifications to it from the framework and specified SCFs, while it handles a temporary
load condition. */
void suspendNotification (
in TpServiceIDList serviceIDs       // Specifies the framework and SCFs for which
// notifications are to be suspended.
) raises (TpGeneralException);

};



/* The Load Manager client application interface is used by the Framework to access the
application load balancing SCF. */
interface IpAppLoadManager : IpOsa {

/* This method is invoked by the Framework to request for load statistic records produced
by a specified application. */
void queryAppLoadReq (
in TpServiceIDList serviceIDs,      // Specifies the SCFs or application for which the
// load statistics shall be reported.
in TpTimeInterval timeInterval      // The time interval within which the load statistics
// are generated.
) raises (TpGeneralException);

/* This method is invoked by the Framework to return load statistics to the application
which requested the information. */
void queryLoadRes (
in TpLoadStatisticList loadStatistics       // The load statistics supplied by the
 // Framework.
) raises (TpGeneralException);

/* This method is invoked by the Framework to return an error code to the application that
requested load statistics. */
void queryLoadErr (
in TpLoadStatisticErrorList loadStatisticsError     // The error code supplied by the
// Framework.
) raises (TpGeneralException);

/* This method is invoked by the Framework to disable load control activity at the client
application based on policy, after the load level of the Framework or SCF which has
been registered for load control moves back to normal. */
void disableLoadControl (
in TpServiceIDList serviceIDs        // Specifies the framework and SCFs for which the
// load has changed to normal.
) raises (TpGeneralException);

/* This method is invoked by the Framework to enable load management activity at the client
application based on the policy, upon detecting load condition change. */
void enableLoadControl (
in TpLoadStatisticList loadStatistics       // The new load statistics.
) raises (TpGeneralException);

/* This method is invoked by the Framework to resume the notification from an application
for its load status after the detection of load level change at the Framework and the
evaluation of the load balancing policy. */
void resumeNotification() raises (TpGeneralException);

/* This method is invoked by the Framework to suspend the notification from an application
for its load status after the detection of load level change at the Framework and the
evaluation of the load balancing policy. */
void suspendNotification() raises (TpGeneralException);

};



/* The Fault Manager Framework interface is used by the client application to inform the
Framework of events that affect the integrity of the Framework and SCFs, and to request
information about the integrity of the system. */
interface IpFaultManager : IpOsa {

/* This method may be invoked by the client application to test that the Framework or a
SCF is operational. */
void activityTestReq (
in TpActivityTestID activityTestID, // Identifier provided by the client
// application to correlate the
// response with this request.
```

```
in TpServiceID svcID                        // Identifies for which SCF the client
 // application is requesting the activity test
// be done.

) raises (TpGeneralException);

/* This method is invoked by the client application to return the result of a previously
requested activity test. */
void appActivityTestRes (
in TpActivityTestID activityTestID,         // Used by the Framework to correlate this
// response with the original request.
in TpActivityTestRes activityTestResult     // Result of the activity test.
) raises (TpGeneralException);

/* This method is invoked by the client application to inform the Framework that it can no
longer use the indicated SCF. */
void svcUnavailableInd (
in TpServiceID serviceID            // Identity of the SCF which can no longer be used.

) raises (TpGeneralException);

/* This method is invoked by the client application to request fault statistics from the
Framework. */
void genFaultStatsRecordReq (
in TpTimeInterval timePeriod,           // The period over which the fault statistics
// are to be generated.
in TpServiceIDs serviceIDList       // The SCFs that the application would like
// to have included in the general fault
// statistics record.

) raises (TpGeneralException);

};



/* The Fault Manager client application interface is used by the Framework to inform the
application of events that affect the integrity of the Framework, SCF or client
application. */
interface IpAppFaultManager : IpOsa {

/* This method is invoked by the Framework, in response to an activityTestReq, to return
the result of the activity test in this method. */
void activityTestRes (
in TpActivityTestID activityTestID,             // The identifier provided to correlate this
// response with the original request.
in TpActivityTestRes activityTestResult     // Result of the activity test.
) raises (TpGeneralException);

/* This method is invoked by the Framework to request that the client application carries
out an activity test to check that is it operating correctly. */
void appActivityTestReq (
in TpActivityTestID activityTestID      // The identifier provided to correlate this
// response with the original request.
) raises (TpGeneralException);

/* This method is invoked by the Framework to notify the client application of a failure
within the Framework. */
void fwFaultReportInd (
in TpInterfaceFault fault           // The fault that has been detected.
) raises (TpGeneralException);

/* This method is invoked by the Framework to notify the client application that a
previously reported fault has been rectified. */
void fwFaultRecoveryInd (
in TpInterfaceFault fault           // The fault from which the framework has recovered.
) raises (TpGeneralException);
         void fwUnavailableInd (
         in TpFwUnavailReason reason
  ) raises (TpGeneralException);
/* This method is invoked by the Framework to inform the client application that it can no
longer use the indicated SCF due to a failure. */
void svcUnavailableInd (
in TpServiceID serviceID,           // Identity of the SCF which can no longer be used.
in TpSvcUnavailReason reason        // The reason why the SCF is no longer available.
) raises (TpGeneralException);

/* This method is invoked by the Framework to provide fault statistics to a client

application in response to a genFaultStatsRecordReq. */

void genFaultStatsRecordRes (
in TpFaultStatsRecord faultStatistics,      // The fault statistics record.
in TpServiceIDList serviceIDs               // The SCFs that have been included in the
// general fault statistics record.
) raises (TpGeneralException);

};
```

```
/* The OAM Framework interface is used by the client application to query the system date and
time, for synchronization purposes. */
interface IpOAM : IpOsa {

/* This method is invoked by the client application to interchange the system an client
application date and time. */
void systemDateTimeQuery (
in TpDateAndTime clientDateAndTime,         // The date and time of the client.
out TpDateAndTime systemDateAndTime            // The date and time of the system.
) raises (TpGeneralException);

};


/* The OAM client application interface is used by the Framework to query the application date
and time, for synchronization purposes. */
interface IpAppOAM : IpOsa {

/* This method is invoked by the Framework to interchange the system an client application
date and time. */
void systemDateTimeQuery (
in TpDateAndTime systemDateAndTime,         // The date and time of the system.
out TpDateAndTime clientDateAndTime            // The date and time of the client.
) raises (TpGeneralException);

};


};};};};};
```

9.2.5 Registration IDL

```
#include <fw.idl>

module org{
module threegpp{
module osa{
module fw{
module registration{

/*******************************************************************************/
//                                Interface definitions                        //
/*******************************************************************************/

/* The Service Registration Framework interface provides the methods used for the registration
of network SCFs at the Framework. */
interface IpServiceRegistration : IpOsa {

/* This method is used to register a SCF in the Framework, for subsequent discovery by
the applications. */
void registerService (
in TpServiceTypeName                serviceTypeName,
in TpServicePropertyList        servicePropertyList,
out TpServiceID                 serviceID
) raises (TpGeneralException);

/* This method informs the Framework of the availability of a service factory for a
previously registered SCF. */
void announceServiceAvailability (
in TpServiceID      serviceID,
in IpOsa                serviceFactory
) raises (TpGeneralException);

/* This method is used to remove a registered SCF from the Framework. */
void unregisterService (
in TpServiceID      serviceID
) raises  (TpGeneralException);

/* This method is used to ebtain the decription of a certain SCF as it was registered in
the Framework. */
void describeService (
in TpServiceID                  serviceID,
out TpServiceDescription        serviceDescription
) raises (TpGeneralException);
        };

/* The Service Factory Framework interface provides the Framework with access to a manager
interface of a network SCF to be given to an application. */
interface IpSvcFactory : IpOsa {

/* This method returns an SCF manager interface reference for a specified application. */
void getServiceManager (
in TpDomainID       application,
```

```
in TpServicePropertyList        serviceProperties,
out IpServiceOsa                serviceManager
) raises (TpGeneralException);
        };
};};};};};
```

9.3 Call Control

9.3.1 Common Data Types for Call Control

```
// source file: CC.idl
// Generic Call Data description

#ifndef __OSA_CC_DEFINED
#define __OSA_CC_DEFINED

#include <OSA.idl>
#include <UI.idl>

module org
{
    module threegpp
    {
        module osa
        {
            module cc
            {
                /* Defines the mechanism that will be used to alert a called party. */
                typedef TpInt32 TpCallAlertingMechanism;

                /* Defines the bearer service associated with the call. */
                enum TpCallBearerService
                {
                    P_CALL_BEARER_SERVICE_UNKNOWN,                    /* Bearer capability
information
                                                                 unknown at this time*/
                    P_CALL_BEARER_SERVICE_SPEECH,                   /* Speech*/
                    P_CALL_BEARER_SERVICE_DIGITALUNRESTRICTED,      /* Unrestricted digital
information*/
                    P_CALL_BEARER_SERVICE_DIGITALRESTRICTED,          /* Restricted digital
information*/
                    P_CALL_BEARER_SERVICE_AUDIO,                        /* 3.1 kHz audio*/
                    P_CALL_BEARER_SERVICE_DIGITALUNRESTRICTEDTONES, /* Unrestricted digital
information
                                                                with tones/announcements*/
                    P_CALL_BEARER_SERVICE_VIDEO                         /*Video*/
                };



                /*This data defines the bearer capabilities associated with the call. (3GPP TS
24.002) This
           information is network operator specific and may not always be available because there
           is no standard protocol to retrieve the information */
                enum TpCallNetworkAccessType
                {
                    P_CALL_NETWORK_ACCESS_TYPE_UNKNOWN,     /* Network type information unknown at
this time */
                    P_CALL_NETWORK_ACCESS_TYPE_POT,         /* POTS */
                    P_CALL_NETWORK_ACCESS_TYPE_ISDN,        /* ISDN */
                    P_CALL_NETWORK_ACCESS_TYPE_DIALUPINTERNET,  /* Dial-up Internet  */
                    P_CALL_NETWORK_ACCESS_TYPE_XDSL,        /* xDSL */
                    P_CALL_NETWORK_ACCESS_TYPE_WIRELESS     /* Wireless */
                };

                /* Defines the category of a calling or called party (e.g. call priority, payphone,
            prepaid).*/
                enum TpCallPartyCategory
                {
                    P_CALL_PARTY_CATEGORY_UNKNOWN,      /*calling party's category unknown at this
time*/
                    P_CALL_PARTY_CATEGORY_OPERATOR_F,   /* operator, language French*/
                    P_CALL_PARTY_CATEGORY_OPERATOR_E,   /* operator, language English*/
                    P_CALL_PARTY_CATEGORY_OPERATOR_G,   /* operator, language German*/
                    P_CALL_PARTY_CATEGORY_OPERATOR_R,   /* operator, language Russian*/
                    P_CALL_PARTY_CATEGORY_OPERATOR_S,   /* operator, language Spanish*/
                    P_CALL_PARTY_CATEGORY_ORDINARY_SUB,     /* ordinary calling subscriber*/
                    P_CALL_PARTY_CATEGORY_PRIORITY_SUB,     /* calling subscriber with priority*/
                    P_CALL_PARTY_CATEGORY_DATA_CALL,    /* data call (voice band data) */
                    P_CALL_PARTY_CATEGORY_TEST_CALL,    /* test call*/
                    P_CALL_PARTY_CATEGORY_PAYPHONE      /* payphone*/
                };
```

```
                /* This data type defines the tele-service associated with the call. (Q.763: User
Teleservice Information,  Q.931: High Layer Compatitibility Information, and 3GPP TS 22.003)Defines
the tele-service associated with the call (e.g. speech, video, fax, file transfer, browsing). */
                enum TpCallTeleService
                {
                    P_CALL_TELE_SERVICE_UNKNOWN,        /* Teleservice information unknown at this
time*/
                    P_CALL_TELE_SERVICE_TELEPHONY,      /* Telephony */
                    P_CALL_TELE_SERVICE_FAX_2_3,        /* Facsimile Group 2/3 */
                    P_CALL_TELE_SERVICE_FAX_4_I,        /* Facsimile Group 4, Class I */
                    P_CALL_TELE_SERVICE_FAX_4_II_III,   /* Facsimile Group 4, Classes II and III */
                    P_CALL_TELE_SERVICE_VIDEOTEX_SYN,   /* Syntax based Videotex */
                    P_CALL_TELE_SERVICE_VIDEOTEX_INT,   /* International Videotex interworking via
gateways or interworking units */
                    P_CALL_TELE_SERVICE_TELEX,      /* Telex service*/
                    P_CALL_TELE_SERVICE_MHS,        /* Message Handling Systems */
                    P_CALL_TELE_SERVICE_OSI,        /* OSI application*/
                    P_CALL_TELE_SERVICE_FTAM,       /* FTAM application*/
                    P_CALL_TELE_SERVICE_VIDEO,      /* Videotelephony*/
                    P_CALL_TELE_SERVICE_VIDEO_CONF,     /* Videoconferencing*/
                    P_CALL_TELE_SERVICE_AUDIOGRAPH_CONF,    /* Audiographic conferencing*/
                    P_CALL_TELE_SERVICE_MULTIMEDIA,     /* Multimedia services*/
                    P_CALL_TELE_SERVICE_CS_INI_H221,    /* Capability set of initial channel of
H.221*/
                    P_CALL_TELE_SERVICE_CS_SUB_H221,    /* Capability set of subsequent channel of
H.221*/
                    P_CALL_TELE_SERVICE_CS_INI_CALL,    /* Capability set of initial channel
associated with an active 3.1 kHz audio or speech call.*/
                    P_CALL_TELE_SERVICE_DATATRAFFIC,    /* Data traffic.*/
                    P_CALL_TELE_SERVICE_EMERGENCY_CALLS,    /* Emergency Calls*/
                    P_CALL_TELE_SERVICE_SMS_MT_PP,      /* Short message MT/PP*/
                    P_CALL_TELE_SERVICE_SMS_MO_PP,      /* Short message MO/PP*/
                    P_CALL_TELE_SERVICE_CELL_BROADCAST,     /* Cell Broadcast Service*/
                    P_CALL_TELE_SERVICE_ALT_SPEECH_FAX_3,   /* Alternate speech and facsimile group
3*/
                    P_CALL_TELE_SERVICE_AUTOMATIC_FAX_3,    /* Automatic Facsimile group 3*/
                    P_CALL_TELE_SERVICE_VOICE_GROUP_CALL,   /* Voice Group Call Service*/
                    P_CALL_TELE_SERVICE_VOICE_BROADCAST     /* Voice Broadcast Service*/
                };


                /* Defines a specific call event report type. */
                enum TpCallAppInfoType
                {
                    P_CALL_APP_UNDEFINED,               /* Undefined */
                    P_CALL_APP_ALERTING_MECHANISM,      /* The alerting mechanism or pattern to use
*/
                    P_CALL_APP_NETWORK_ACCESS_TYPE,     /* The network access type (e.g. ISDN) */
                    P_CALL_APP_TELE_SERVICE,            /* Indicates the tele-service (e.g. speech)
and related info such as clearing programme */
                    P_CALL_APP_BEARER_SERVICE,          /* Indicates the bearer service (e.g. 64kb/s
unrestricted data). */
                    P_CALL_APP_PARTY_CATEGORY,          /* The category of the calling or called
party */
                    P_CALL_APP_PRESENTATION_ADDRESS,    /* The address to be presented to other call
parties */
                    P_CALL_APP_GENERIC_INFO,            /* Carries unspecified application-SCF
information */
                    P_CALL_APP_ADDITIONAL_ADDRESS       /* Indicates an additional address */
                };

                /* Defines the Tagged Choice of Data Elements that specify call application-related
specific information. */
                union TpCallAppInfo switch(TpCallAppInfoType)
                {
                    case P_CALL_APP_TELE_SERVICE:
                    TpCallTeleService CallAppTeleService;
                    case P_CALL_APP_BEARER_SERVICE:
                    TpCallBearerService CallAppBearerService;
                    case P_CALL_APP_PARTY_CATEGORY:
                    TpCallPartyCategory CallAppPartyCategory;
                    case P_CALL_APP_PRESENTATION_ADDRESS:
                    TpAddress CallAppPresentationAddress;
                    case P_CALL_APP_GENERIC_INFO:
                    TpString CallAppGenericInfo;
                    case P_CALL_APP_ADDITIONAL_ADDRESS:
                    TpAddress CallAppAdditionalAddress;
                    case P_CALL_APP_ALERTING_MECHANISM:
                    TpCallAlertingMechanism CallAppAlertingMechanism;
                    case P_CALL_APP_NETWORK_ACCESS_TYPE:
                    TpCallNetworkAccessType CallAppNetworkAccessType;
                };

                typedef sequence <TpCallAppInfo> TpCallAppInfoSet;

                enum TpCallChargeOrderCategory
                {
                    P_CALL_CHARGE_PER_TIME, /* Charge per time*/
```

```
                    P_CALL_CHARGE_NETWORK /* Operator specific charge plan specification, e.g.
charging table name / charging table entry*/
                };

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

                union TpCallChargeOrder switch(TpCallChargeOrderCategory)
                {
                    case P_CALL_CHARGE_PER_TIME:    TpChargePerTime ChargePerTime;
                    case P_CALL_CHARGE_NETWORK:     TpString NetworkCharge;
                };



                /* Defines the Sequence of Data Elements that specify the charge plan for the call
This data type is identical to a TpString, and defines the call charge plan to be used for the call.
The values of this data type are operator specific. */
                struct TpCallChargePlan
                {
                    TpCallChargeOrder ChargeOrderType;
                    TpString Currency;
                    TpString AdditionalInfo;
                };

                const TpInt32 P_EVENT_NAME_UNDEFINED = 0;                    // Undefined
                const TpInt32 P_EVENT_GCCS_OFFHOOK_EVENT = 1;                // Offhook event
                const TpInt32 P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT = 2;      // Address information
collected
                const TpInt32 P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT = 4;       // Address information
is analysed
                const TpInt32 P_EVENT_GCCS_CALLED_PARTY_BUSY = 8;            // Called party is
busy
                const TpInt32 P_EVENT_GCCS_CALLED_PARTY_UNREACHABLE = 16;    // Called party is
unreachable
                const TpInt32 P_EVENT_GCCS_NO_ANSWER_FROM_CALLED_PARTY = 32; // No answer from
called party
                const TpInt32 P_EVENT_GCCS_ROUTE_SELECT_FAILURE = 64;        // Failure in routing
the call
                const TpInt32 P_EVENT_GCCS_ANSWER_FROM_CALL_PARTY = 128;     // Party answered call

                typedef TpInt32 TpCallEventName; /*Defines the names of event being notified. */

                enum TpCallNotificationType
                {
                    P_ORIGINATING,  // The notification is related to the originating user in the
call.
                    P_TERMINATING   // The notification is related to the terminating user in the
call.
                };

                struct TpCallEventCriteria
                {
                    TpAddressRange DestinationAddress;      /*Destination address or address range*/
                    TpAddressRange OriginationAddress;      /*Origination address or address range
*/
                    TpCallEventName CallEventName;               /*Name of the event(s) */
                    TpCallNotificationType CallNotificationType;  /*Indicates whether the criteria
are related to the originating or terminating user in the call */
                    TpCallMonitorMode MonitorMode;

                };


                /* Defines a sequence of data elements that specify a requested call event
notification criteria with the associated assignmentID  */
                struct TpCallEventCriteriaResult
                {
                    TpCallEventCriteria EventCriteria;
                    TpInt32 AssignmentID;
                };

                /* Defines a set of TpCallEventCriteriaResult */
                typedef sequence <TpCallEventCriteriaResult> TpCallEventCriteriaResultSet;

                //Defines the type of notification.
                //Indicates whether it is related to the originating of the terminating user in the
call.
                struct TpCallEventInfo
                {
                    TpAddress DestinationAddress;
                    TpAddress OriginatingAddress;
                    TpAddress OriginalDestinationAddress;
                    TpAddress RedirectingAddress;
                    TpCallAppInfoSet CallAppInfo;
                    TpCallEventName CallEventName;
                    TpCallNotificationType CallNotificationType;
                    TpCallMonitorMode MonitorMode;
                };
```

```
                /* Defines the Sequence of Data Elements that specify the cause of the release of a
call.*/
                struct TpCallReleaseCause {
                TpInt32 Value;
                TpInt32 Location;
                };

                /* Defines the Sequence of Data Elements that specify the reason for the call
ending.*/
                struct TpCallEndedReport
                {
                    TpSessionID CallLegSessionID;
                    TpCallReleaseCause Cause;
                };

                /* Defines a specific call error. */
                enum TpCallErrorType
                {
                    P_CALL_ERROR_UNDEFINED,         /* Undefined */
                    P_CALL_ERROR_INVALID_ADDRESS,   /* The operation failed because an invalid
address was given */
                    P_CALL_ERROR_INVALID_STATE     /* The call was not in a valid state for the
requested operation */
                };

                /* Defines the Tagged Choice of Data Elements that specify additional call error and
call error specific information. This is also used to specify call leg errors and call information
errors. */
                union TpCallAdditionalErrorInfo switch(TpCallErrorType)
                {
                    case P_CALL_ERROR_INVALID_ADDRESS: TpAddressError CallErrorInvalidAddress;
                    default: short Dummy;   // allows initialization of the union in the default
case
                };

                /* Defines the Sequence of Data Elements that specify the additional information
relating to an undefined call error. */
                struct TpCallError
                {
                    TpCallAdditionalErrorInfo AdditionalErrorInfo;
                    TpCallErrorType ErrorType;
                    TpDateAndTime ErrorTime;
                };

                /* Defines the cause of the call fault detected. */
                enum TpCallFault
                {
                    P_CALL_FAULT_UNDEFINED,     /* Undefined */

                    P_CALL_TIMEOUT_ON_RELEASE,  /* Final report has been sent to the application,
but the application did not explicitly release or deassign the call object, within a specified time.
*/
                    P_CALL_TIMEOUT_ON_INTERRUPT /* 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.*/
                };

                /* Defines the type of call information requested and reported */
                const TpInt32 P_CALL_INFO_UNDEFINED = 0;         /* Undefined */
                const TpInt32 P_CALL_INFO_TIMES = 1;             /* Relevant call times */
                const TpInt32 P_CALL_INFO_RELEASE_CAUSE = 2;     /* Call release cause. */
                const TpInt32 P_CALL_INFO_INTERMEDIATE = 4;      /* Send only intermediate reports
(i.e., when a party leaves the call). */

                typedef TpInt32 TpCallInfoType;

                /* Defines the Sequence of Data Elements that specify the call information
requested. Information that was not requested may be undefined or not present. */
                struct TpCallInfoReport
                {
                    TpCallInfoType CallInfoType;
                    TpDateAndTime CallInitiationStartTime;
                    TpDateAndTime CallConnectedToResourceTime;
                    TpDateAndTime CallConnectedToDestinationTime;
                    TpDateAndTime CallEndTime;
                    TpCallReleaseCause Cause;
                };

                /* Defines the mode that the call will monitor for events, or the mode that the call
is in following a detected event. */
                enum TpCallMonitorMode
                {
                    P_CALL_MONITOR_MODE_INTERRUPT,     /* The call event is intercepted by the call
control SCF and call processing is interrupted. The application is notified of the event and call
processing resumes following an appropriate API call or network event (such as a call release) */
                    P_CALL_MONITOR_MODE_NOTIFY,        /* The call event is detected by the call
control SCF but not intercepted. The application is notified of the event and call processing
continues */
```

```
                    P_CALL_MONITOR_MODE_DO_NOT_MONITOR /* Do not monitor for the event */
                };

                /* Defines the type of call overload that has been detected (and possibly acted
upon) by the network. */
                enum TpCallOverloadType
                {
                    P_CALL_OVERLOAD_TYPE_UNDEFINED,   /* Infinite interval (do not admit any calls)
*/
                    P_CALL_OVERLOAD_TYPE_NEW_CALLS,   /* New calls to the application are causing
overload (i.e. inbound overload) */
                    P_CALL_OVERLOAD_TYPE_ROUTED_CALLS /* Calls being routed to destination or
origination addresses by the application are causing overload (i.e. outbound overload) */
                };

                /* Defines a specific call event report type. */
                enum TpCallReportType
                {
                    P_CALL_REPORT_UNDEFINED,       /* Undefined */
                    P_CALL_REPORT_PROGRESS,        /* Call routing progress event */
                    P_CALL_REPORT_ALERTING,        /* Call alerting at  address */
                    P_CALL_REPORT_ANSWER,          /* Call answered at address */
                    P_CALL_REPORT_BUSY,            /* Called address refused call due to busy */
                    P_CALL_REPORT_NO_ANSWER,       /* No answer at called address */
                    P_CALL_REPORT_DISCONNECT,      /* Call disconnect requested by address */
                    P_CALL_REPORT_REDIRECTED,
                    P_CALL_REPORT_SERVICE_CODE,
                    P_CALL_REPORT_ROUTING_FAILURE

                };

                /* Defines the Tagged Choice of Data Elements that specify additional call report
information. */
                union TpCallAdditionalReportInfo switch(TpCallReportType)
                {
                    case P_CALL_REPORT_BUSY: TpCallReleaseCause Busy;
                    case P_CALL_REPORT_DISCONNECT: TpCallReleaseCause CallDisconnect;
                    case P_CALL_REPORT_REDIRECTED: TpAddress ForwardAddress;
                    case P_CALL_REPORT_SERVICE_CODE: TpCallReleaseCause ServiceCode;
                    case P_CALL_REPORT_ROUTING_FAILURE: TpCallReleaseCause RoutingFailure;

                    default:  short Dummy;   // allows initialization of the union in the default
case
                };

                struct TpCallReport
                {
                    TpCallMonitorMode MonitorMode;
                    TpDateAndTime CallEventTime;
                    TpCallReportType CallReportType;
                    TpCallAdditionalReportInfo AdditionalReportInfo;
                };

                /* Defines the different types of service codes that can be received during the
call.*/
                enum TpCallServiceCodeType
                {
                    P_CALL_SERVICE_CODE_UNDEFINED,  /* The type of service code is unknown. The
corresponding string is operator specific.*/
                    P_CALL_SERVICE_CODE_DIGITS, /* The user entered a digit sequence during the
call. The corresponding string is an ascii representation of the received digits. */
                    P_CALL_SERVICE_CODE_FACILITY, /*  A facility information element is received.
The corresponding string contains the facility information element as defined in ITU Q.932*/
                    P_CALL_SERVICE_CODE_U2U, /* A user-to-user message was received. The associated
string contains the content of the user-to-user information element. */
                    P_CALL_SERVICE_CODE_HOOKFLASH,  /* The user performed a hookflash, optionally
followed by some digits. The corresponding string is an ascii representation of the entered digits.
*/
                    P_CALL_SERVICE_CODE_RECALL /* The user pressed the register recall button,
optionally followed by some digits. The corresponding string is an ascii representation of the
entered digits. */
                    };

                /* Defines the Sequence of Data Elements that specify the service code and type of
service code received during a call. The service code type defines how the value string should be
interpreted. Defines the service code received during a call. For example, this may be a digit
sequence, user-user information, recall, flash-hook or ISDN Facility Information Element. This data
type is identical to a TpString. The coding of this data type is operator specific. */
                struct TpCallServiceCode
                {
                    TpCallServiceCodeType CallServiceCodeType;
                    TpString ServiceCodeValue;
                };


                /* Defines the Tagged Choice of Data Elements that specify specific criteria. */
                union TpCallAdditionalReportCriteria switch(TpCallReportType)
                {
                    case P_CALL_REPORT_NO_ANSWER: TpDuration NoAnswerDuration;
```

```
                    case P_CALL_REPORT_SERVICE_CODE: TpCallServiceCode ServiceCode;
                    default: short Dummy;   // allows initialization of the union in the default
case
                };

                /* Defines the Sequence of Data Elements that specify the criteria relating to call
report requests. */
                struct TpCallReportRequest
                {
                    TpCallMonitorMode MonitorMode;
                    TpCallReportType CallReportType;
                    TpCallAdditionalReportCriteria AdditionalReportCriteria;
                };

                /* Defines a Numbered Set of Data Elements of TpCallReportRequest. */
                typedef sequence <TpCallReportRequest> TpCallReportRequestSet;

                const TpInt32 P_CALL_SUPERVISE_TIMEOUT = 1;      /* The call supervision timer has
expired. */
                const TpInt32 P_CALL_SUPERVISE_CALL_ENDED = 2;   /* The call has ended, either due
to timer expiry or calling or called party release. In case the called party disconnects but a
follow-on call can still be made also this indication is used.*/
                const TpInt32 P_CALL_SUPERVISE_TONE_APPLIED = 4; /* A warning  tone has been
applied. */
                const TpInt32 P_CALL_SUPERVISE_UI_FINISHED = 8;  /* The user interaction has
finished */

                /* Defines the responses from the call control SCF for calls that are supervised:*/
                typedef TpInt32 TpCallSuperviseReport;

                const TpInt32 P_CALL_SUPERVISE_RELEASE = 1;      /* Release the call when the call
supervision timer expires. */
                const TpInt32 P_CALL_SUPERVISE_RESPOND = 2;      /* Notify the application when the
call supervision timer expires. */
                const TpInt32 P_CALL_SUPERVISE_APPLY_TONE = 4;   /* Send a warning tone to the
controlling party when the call supervision timer expires. If call release is requested, then the
call will be released following the tone after an administered time period */


                /* Defines the following treatment of the call by the call control SCF when the call
supervision timer expires.*/
                typedef TpInt32 TpCallSuperviseTreatment;


                /* Define the possible Exceptions. */
                const TpInt32 P_GCCS_SERVICE_INFORMATION_MISSING = 256;
                const TpInt32 P_GCCS_SERVICE_FAULT_ENCOUNTERED = 257;
                const TpInt32 P_GCCS_UNEXPECTED_SEQUENCE = 258;
                const TpInt32 P_GCCS_INVALID_ADDDRESS = 259;
                const TpInt32 P_GCCS_INVALID_CRITERIA = 260;
                const TpInt32 P_GCCS_INVALID_NETWORK_STATE = 261;


                exception TpGCCSException
                {
                    TpInt32 exceptionType;
                };

        /* The next data type is not used for an SCF implementation based
           on this specification: */
           typedef TpInt32 TpCallLoadControlIntervalRate;

        /* The next data type is not used for an SCF implementation based
           on this specification: */
         const TpInt32 P_CALL_LOAD_CONTROL_ADMIT_NO_CALLS = 0;

        /* The next data type is not used for an SCF implementation based
           on this specification: */
         enum TpCallLoadControlMechanismType {
           P_CALL_LOAD_CONTROL_PER_INTERVAL
         };

        /* The next data type is not used for an SCF implementation based
           on this specification: */
         union TpCallLoadControlMechanism switch(TpCallLoadControlMechanismType) {
         case P_CALL_LOAD_CONTROL_PER_INTERVAL:
           TpCallLoadControlIntervalRate CallLoadControlPerInterval;
         };

        /* The next data type is not used for an SCF implementation based
           on this specification: */
         enum TpCallTreatmentType {
           P_CALL_TREATMENT_DEFAULT,
           P_CALL_TREATMENT_RELEASE,
           P_CALL_TREATMENT_SIAR
         };

        /* The next data type is not used for an SCF implementation based
           on this specification: */
```

```
        union TpCallAdditionalTreatmentInfo switch(TpCallTreatmentType) {
        case P_CALL_TREATMENT_SIAR: ui::TpUIInfo InformationToSend;
        default: short Dummy;
        };

       /* The next data type is not used for an SCF implementation based
          on this specification: */
        struct TpCallTreatment {
          TpCallTreatmentType CallTreatmentType;
          TpCallReleaseCause ReleaseCause;
          TpCallAdditionalTreatmentInfo AdditionalTreatmentInfo;
        };

                }; // end module cc
            }; // end module osa
        }; // end module threegpp
    }; // end module org

#endif

// END file CC.idl
```

9.3.2 Generic Call Control IDL

```
// source file: GCC.idl
// GenericCall Interface description

#ifndef __OSA_CC_GCC_DEFINED
#define __OSA_CC_GCC_DEFINED

#include <CC.idl>

module org {
 module threegpp {
  module osa {
   module cc {
    module gcc {


  interface IpAppCallControlManager; // forward definition
  interface IpAppCall;               // forward definition
  interface IpCall;                  // forward definition

/* Sequence of Data Elements that unambiguously specify the Generic Call object */
  struct TpCallIdentifier {
    IpCall CallReference;
    TpSessionID CallSessionID;
  };


  /* This interface is the SCF manager' interface for Generic Call Control. */
  interface IpCallControlManager : IpService {
    /* This method is used to enable call notifications. */
    void enableCallNotification (
      in IpAppCallControlManager appInterface,
      in TpCallEventCriteria eventCriteria,
      out TpAssignmentID assignmentID
      )
    raises (TpGCCSException, TpGeneralException);

    /* This method is used by the application to disable call notifications.*/
    void disableCallNotification (
      in TpAssignmentID assignmentID
      )
    raises (TpGCCSException, TpGeneralException);

    void changeCallNotification (
      in TpAssignmentID assignmentID,
      in TpCallEventCriteria eventCriteria
      )
    raises (TpGCCSException, TpGeneralException);

    void getCriteria (
      out TpCallEventCriteriaResultSet eventCriteria
      )
    raises (TpGCCSException, TpGeneralException);

    /* The next operation is not supported for Release 99 and must
       return the exception “Method not supported” when invoked on a SCF
       implementation based on this specification: */
       void createCall (
          in IpAppCall appCall,
          out TpCallIdentifier callReference
          )
       raises (TpGCCSException,TpGeneralException);

    /* The next operation is not supported for Release 99 and must
```

```
      return the exception “Method not supported” when invoked on a SCF
      implementation based on this specification: */
      void setCallLoadControl (
        in TpDuration duration,
        in TpCallLoadControlMechanism mechanism,
        in TpCallTreatment treatment,
        in TpAddressRange addressRange,
        out TpAssignmentID assignmentID
        )
       raises (TpGCCSException, TpGeneralException);

};

/* This interface provides the means to control a simple call. */
interface IpCall : IpService {
  /* This method requests routing of the call to the destination party.*/
    void routeReq (
    in TpSessionID callSessionID,
    in TpCallReportRequestSet responseRequested,
    in TpAddress targetAddress,
    in TpAddress originatingAddress,
    in TpAddress originalDestinationAddress,
    in TpAddress redirectingAddress,
    in TpCallAppInfoSet appInfo,
    out TpSessionID callLegSessionID
    )
  raises (TpGCCSException, TpGeneralException);

  /* This method requests the release of the call and associated objects.*/
  void release (
    in TpSessionID callSessionID,
    in TpCallReleaseCause cause
    )
  raises (TpGCCSException, TpGeneralException);

  /* This method requests that the relationship between the application and
     the call and associated objects be de-assigned.  */
  void deassignCall (
    in TpSessionID callSessionID
    )
  raises (TpGCCSException, TpGeneralException);

  /* This method requests information associated with the call.*/
  void getCallInfoReq (
    in TpSessionID callSessionID,
    in TpCallInfoType callInfoRequested
    )
  raises (TpGCCSException, TpGeneralException);

  /* Set an operator specific charge plan for the call.  */
  void setCallChargePlan (
    in TpSessionID callSessionID,
    in TpCallChargePlan callChargePlan
    )
  raises (TpGCCSException, TpGeneralException);

  /* The application calls this method to supervise a call.  */
  void superviseCallReq (
    in TpSessionID callSessionID,
    in TpDuration time,
    in TpCallSuperviseTreatment treatment

    )
  raises (TpGCCSException, TpGeneralException);

  void setAdviceOfCharge(
    in TpSessionID callSessionID,
    in TpAoCInfo aOCInfo,
    in TpDuration tariffSwitch
    )
  raises (TpGCCSException, TpGeneralException);

  /* The next operation is not supported for Release 99 and must
     return the exception “Method not supported” when invoked on a SCF
     implementation based on this specification: */
     void getMoreDialledDigitsReq (
       in TpSessionID callSessionID,
       in TpInt32 length
       )
     raises (TpGeneralException, TpGCCSException);

};


/* The generic call control manager application interface provides the
   application call control management functions to the generic call control
   SCF. */
interface IpAppCallControlManager : IpOsa {
  void callAborted (
```

```
      in TpSessionID callReference
      )
    raises (TpGCCSException, TpGeneralException);

    /* This method notifies the application of the arrival of a call-related event. */
    void callEventNotify (
      in TpCallIdentifier callReference,
      in TpCallEventInfo eventInfo,
      in TpAssignmentID assignmentID,
      out IpAppCall appInterface
      )
    raises (TpGCCSException, TpGeneralException);

        /* This method indicates to the application that all event notifications
       have been terminated .*/
    void callNotificationInterrupted ()
      raises (TpGCCSException, TpGeneralException);

    void callNotificationContinued ()
      raises (TpGCCSException, TpGeneralException);

    /* The next operation is not supported for Release 99 and must
       return the exception “Method not supported” when invoked on a SCF
       implementation based on this specification: */
       void callOverloadEncountered (
         in TpAssignmentID assignmentID
         )
       raises (TpGeneralException,TpGCCSException);

    /* The next operation is not supported for Release 99 and must
       return the exception “Method not supported” when invoked on a SCF
       implementation based on this specification: */
       void callOverloadCeased (
         in TpAssignmentID assignmentID
         )
       raises (TpGeneralException,TpGCCSException);
  };

  /* The application side of the simple call interface is used to handle call
     request responses and state reports. */
  interface IpAppCall : IpOsa {
    /* This method indicates that the request to route the call to the
       destination was successful.*/
    void routeRes (
      in TpSessionID callSessionID,
      in TpCallReport eventReport,
      in TpSessionID callLegSessionID
      )
    raises (TpGCCSException, TpGeneralException);

    /* This method indicates that the request to route the call to the
       destination party was unsuccessful. */
    void routeErr (
      in TpSessionID callSessionID,
      in TpCallError errorIndication,
      in TpSessionID callLegSessionID
      )
    raises (TpGCCSException, TpGeneralException);

    /* This method reports all necessary information requested by the
       application, for example to calculate charging.*/
    void getCallInfoRes (
      in TpSessionID callSessionID,
      in TpCallInfoReport callInfoReport
      )
    raises (TpGCCSException, TpGeneralException);

    /* This asynchronous method reports that the original request was erroneous,
       or resulted in an error condition.*/
    void getCallInfoErr (
      in TpSessionID callSessionID,
      in TpCallError errorIndication
      )
    raises (TpGCCSException, TpGeneralException);

    /* This asynchronous method reports a call supervision event to the application.*/
    void superviseCallRes (
      in TpSessionID callSessionID,
      in TpCallSuperviseReport report,
      in TpDuration usedTime
)
    raises (TpGCCSException, TpGeneralException);

    /* This asynchronous method reports a call supervision error to the application.*/
    void superviseCallErr (
      in TpSessionID callSessionID,
      in TpCallError errorIndication
      )
    raises (TpGCCSException, TpGeneralException);
```

```
    /* This method indicates to the application that a fault in the network has
       been detected.*/
    void callFaultDetected (
      in TpSessionID callSessionID,
      in TpCallFault fault
      )
    raises (TpGCCSException, TpGeneralException);

void callEnded (
      in TpSessionID callSessionID,
      in TpCallEndedReport report
      )
    raises (TpGCCSException, TpGeneralException);

    /* The next operation is not supported for Release 99 and must
       return the exception "Method not supported" when invoked on a SCF
       implementation based on this specification: */
       void getMoreDialledDigitsRes (
         in TpSessionID callSessionID,
         in TpString digits
         )
       raises (TpGeneralException,TpGCCSException);

    /* The next operation is not supported for Release 99 and must
       return the exception "Method not supported" when invoked on a SCF
       implementation based on this specification: */
       void getMoreDialledDigitsErr (
         in TpSessionID callSessionID,
         in TpCallError errorIndication
         )
       raises (TpGeneralException,TpGCCSException);

  };

    }; // end module gcc
   }; // end module cc
  }; // end module osa
 }; // end module threegpp
}; // end module org

#endif

// END file GCC.idl
```

9.3.3 Enhanced Call Control IDL

The IDL in this section is only supplied in order to make the User Interaction IDL compile.

With the createUICall() method on the UIManager object it is possible to associate the UICall object to a Call object as well as a CallLeg object. The CallLeg object is not used in this specification. However the IDL for this interface has to be supplied otherwise the User Interaction IDL will not compile.

```
// source file: ECC.idl

#ifndef __OSA_CC_ECC_DEFINED
#define __OSA_CC_ECC_DEFINED

#include <GCC.idl>

module org {
 module threegpp {
  module osa {
   module cc {
    module ecc {


            typedef TpInt32 TpMediaType;

            const TpInt32 P_AUDIO = 1;
            const TpInt32 P_VIDEO = 2;
            const TpInt32 P_DATA = 4;

            typedef TpInt32 TpAudioCapabilitiesType;

            typedef TpInt32 TpVideoCapabilitiesType;

            typedef TpInt32 TpDataCapabilities;

            union TpChannelDataTypeRequest switch(TpMediaType) {
                case P_DATA: TpDataCapabilities Data;
                case P_VIDEO: TpVideoCapabilitiesType Video;
                case P_AUDIO: TpAudioCapabilitiesType Audio;
            };
```

```
            typedef TpChannelDataTypeRequest TpChannelDataType;

            enum TpChannelDirection {
                P_INCOMING,
                P_OUTGOING
            };


            struct TpChannelRequest {
                TpChannelDataTypeRequest DataTypeRequest;
                TpChannelDirection Direction;
            };

            typedef sequence <TpChannelRequest> TpChannelRequestSet;

enum TpCallLegType {
            P_CALL_LEG_TYPE_UNDEFINED,
            P_CALL_LEG_TYPE_CONTROLLING,
            P_CALL_LEG_TYPE_PASSIVE
 };

enum TpCallLegInfoType {
            P_CALL_LEG_INFO_UNDEFINED,
            P_CALL_LEG_INFO_ADDRESS,
            P_CALL_LEG_INFO_RELEASE_CAUSE,
            P_CALL_LEG_INFO_APPINFO,
            P_CALL_LEG_INFO_TIMES
 };

    interface IpMMChannel : IpService {
            void close (
            in TpSessionID channelSessionID
            )
    raises (TpGeneralException,TpGCCSException);

    };

            struct TpChannel {
                TpChannelDirection Direction;
                IpMMChannel Channel;
                TpChannelDataType DataType;
                TpInt32 ChannelNumber;
            };

            typedef sequence <TpChannel> TpChannelSet;


interface IpCallLeg : IpService {
        void routeCallLegToOrigination (
            in TpSessionID callLegSessionID,
            in TpAddress targetAddress,
            in TpAddress originatingAddress,
            in TpAddress originalCalledAddress,
            in TpAddress redirectingAddress,
            in TpCallAppInfoSet appInfo
            )
        raises (TpGeneralException,TpGCCSException);

        void routeCallLegToDestination (
            in TpSessionID callLegSessionID,
            in TpAddress targetAddress,
            in TpAddress originatingAddress,
            in TpAddress originalCalledAddress,
            in TpAddress redirectingAddress,
            in TpCallAppInfoSet appInfo
            )
        raises (TpGeneralException,TpGCCSException);

        void eventReportReq (
            in TpSessionID callLegSessionID,
            in TpCallReportRequestSet eventReportsRequested
            )
        raises (TpGeneralException,TpGCCSException);

        void release (
            in TpSessionID callLegSessionID,
            in TpCallReleaseCause cause
            )
        raises (TpGeneralException,TpGCCSException);

        void getInfoReq (
            in TpSessionID callLegSessionID,
            in TpCallLegInfoType callLegInfoRequested
            )
        raises (TpGeneralException,TpGCCSException);

        void getType (
            in TpSessionID callLegSessionID,
            out TpCallLegType callLegType
```

```
            )
        raises (TpGeneralException,TpGCCSException);

        void getCall (
            in TpSessionID callLegSessionID,
            out org::threegpp::osa::cc::gcc::TpCallIdentifier callReference
            )
        raises (TpGeneralException,TpGCCSException);

        void mediaChannelAllow (
            in TpSessionID callLegSessionID,
            in TpSessionIDSet channelList
            )
        raises (TpGeneralException,TpGCCSException);

        void getMediaChannels (
            in TpSessionID callLegSessionID,
            out TpChannelSet channels
            )
        raises (TpGeneralException,TpGCCSException);

        void mediaChannelMonitorReq (
            in TpSessionID callLegSessionID,
            in TpChannelRequestSet channelEventCriteria,
            in TpCallMonitorMode monitorMode
            )
        raises (TpGeneralException,TpGCCSException);
};


        struct TpCallLegIdentifier {
          TpSessionID CallLegSessionID;
        IpCallLeg CallLegReference;
        };


    }; // end module ecc
   }; // end module cc
  }; // end module osa
 }; // end module threegpp
}; // end module org

#endif

// END file ECC.idl
```

9.4 User Interaction IDL

9.4.1 Common data types for User Interaction

```
// source file: UI.idl
// User Interaction data description

#ifndef __OSA_UI_DEFINED
#define __OSA_UI_DEFINED

#include <OSA.idl>

module org {
 module threegpp {
  module osa {
   module ui {

    /* Defines the additional properties for the collection of information */
    struct TpUICollectCriteria {
        TpInt32 MinLength;            /* minimum number of characters to collect */
        TpInt32 MaxLength;        /* maxmum number of characters to collect */
        TpString EndSequence;     /* character(s) which terminate an input of variable length. */
        TpDuration StartTimeout;      /* defines a duration (in seconds) */
        TpDuration InterCharTimeout;  /* value for the inter-character time-out timer. */
    };

    /* Defines the UI call error codes. */
    enum TpUIError {
        P_UI_ERROR_UNDEFINED,        /* Undefined error */
        P_UI_ERROR_ILLEGAL_ID,       /* The information id specified is invalid */
        P_UI_ERROR_ID_NOT_FOUND,         /* Information id is not known to the the User Interaction
SCFs */
        P_UI_ERROR_RESOURCE_UNAVAILABLE,     /* Resources used by the User Interaction SCFs are
unavailable. */
        P_UI_ERROR_ILLEGAL_RANGE,        /* The values for manimum and maximum collection length are
out of range */
        P_UI_ERROR_IMPROPER_CALLER_RESPONSE, /* Improper user response */
```

```
        P_UI_ERROR_ABANDON,              /* Specified leg is disconnected before the send
information completed */
        P_UI_ERROR_NO_OPERATION_ACTIVE,  /* No active user interaction for the specified leg. */
        P_UI_ERROR_NO_SPACE_AVAILABLE    /* There is no more storage capacity to record the
message.*/
    };

/* Defines the type of the dataString parameter in the method userInteractionEventNotify */
 enum TpUIEventInfoDataType {
        P_UI_EVENT_DATA_TYPE_UNDEFINED,      /* Undefined */
        P_UI_EVENT_DATA_TYPE_UNSPECIFIED,        /* Unspecified data */
        P_UI_EVENT_DATA_TYPE_TEXT,             /* Text */
        P_UI_EVENT_DATA_TYPE_USSD_DATA       /* USSD data starting with coding scheme */
    };


/* Defines the Sequence of Data Elements that specify the additional criteria for receiving a UI
notification */
    struct TpUIEventCriteria {
        TpAddressRange OriginatingAddress;  /* Address of the end-user for which notification shall
be handled */
  TpAddressRange DestinationAddress;
        TpString ServiceCode;  /* 2 digit code indicating the UI to be triggered. */
    };

    /* Defines the Sequence of Data Elements that specify a UI notification */
    struct TpUIEventInfo {
        TpAddress OriginatingAddress; /* Address of the end-user for which notification shall be
handled */
     TpAddress DestinationAddress;
    TpString ServiceCode; /* 2 digit code indicating the UI to be triggered.  */
 TpUIEventInfoDataType DataTypeIndication;
 TpString DataString;
    };

    /* Defines the cause of the UI fault detected. */
    enum TpUIFault {
        P_UI_FAULT_UNDEFINED,   /* Undefined */
        P_UI_CALL_DEASSIGNED        /* The related Call object has been deassigned. */
    };

    /* Defines the type of  information send to the end-user */
    enum TpUIInfoType {
        P_UI_INFO_ID,       /* The information consists of an ID */
        P_UI_INFO_DATA, /* The information consists of a data string */
        P_UI_INFO_ADDRESS   /* The information consists of a URL. */
    };

    /* Defines the Tagged Choice of Data Elements that specifies the information to be send to a
end-user. */
    union TpUIInfo switch(TpUIInfoType) {
        case P_UI_INFO_ID: TpInt32 InfoID;          /*Defines the ID of the user information script
or stream to send to an end-user.*/
        case P_UI_INFO_DATA: TpString InfoData;  /*Defines the data to be sent to an end-user's
terminal.*/
        case P_UI_INFO_ADDRESS: TpURL InfoAddress;   /*Defines the URL of the text or stream to be
sent to an end-user's terminal*/
    };

    /* Defines the criteria for recording of messages */
    struct TpUIMessageCriteria {
    TpString EndSequence;      /* Defines the character(s) which terminate an input of variable
length. */
    TpDuration MaxMessageTime; /* Specifies the maximum allowed duration in seconds. */
    TpInt32 MaxMessageSize;    /* Specifies the maximum allowed size in bytes of the message. */
    };

    /* Defines the UI call reports if a response was requested. */
    enum TpUIReport {
        P_UI_REPORT_UNDEFINED,      /* Undefined report */
        P_UI_REPORT_ANNOUNCEMENT_ENDED, /* Confirmation that the announcement has ended */
        P_UI_REPORT_LEGAL_INPUT,        /* Information collected., meeting the specified criteria.
*/
        P_UI_REPORT_NO_INPUT,       /* User immediately entered the delimiter character. No valid
information has been returned */
        P_UI_REPORT_TIMEOUT,              /* User did not input any response before the input
timeout expired */
        P_UI_REPORT_MESSAGE_STORED,     /* A message has been stored successfully */
        P_UI_REPORT_MESSAGE_NOT_STORED  /* The message has not been stored successfully */
    };

    /* Defines the situations for which a response is expected following the user interaction. */
 const TpInt32 P_UI_RESPONSE_REQUIRED = 1; /* A response must be sent when the request  has
completed. */
    const TpInt32 P_UI_LAST_ANNOUNCEMENT_IN_A_ROW = 2; /* This is the final announcement within a
sequence. */
    const TpInt32 P_UI_FINAL_REQUEST = 4; /* This is the final request. */
```

```
    typedef TpInt32 TpUIResponseRequest; /* Defines the situations for which a response is expected
following the user interaction. */


    /* Defines the type of the variable parts in the information to send to the user. */
    enum TpUIVariablePartType {
        P_UI_VARIABLE_PART_INT, /* Variable part is of type integer */
        P_UI_VARIABLE_PART_ADDRESS, /* Variable part is of type address */
        P_UI_VARIABLE_PART_TIME,    /* Variable part is of type time */
        P_UI_VARIABLE_PART_DATE,    /* Variable part is of type date */
        P_UI_VARIABLE_PART_PRICE    /* Variable part is of type price */
    };


    /* Defines the Tagged Choice of Data Elements that specify the variable parts in the information
to send to the user. */
    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;
    };

/* Defines a Numbered Set of Data Elements of TpUIVariableInfo. */
  typedef sequence <TpUIVariableInfo> TpUIVariableInfoSet;

    /* Define the possible Exceptions. */
    exception TpGUISException {
        TpInt32 exceptionType;
    };

    const TpInt32 P_GUIS_INVALID_CRITERIA = 768;           /* Invalid criteria specified */
    const TpInt32 P_GUIS_ILLEGAL_ID = 769;              /* Information id specified is invalid */
    const TpInt32 P_GUIS_ID_NOT_FOUND = 770;               /* Information id is not known to the
User Interaction Service */
    const TpInt32 P_GUIS_ILLEGAL_RANGE = 771;              /* The values for minimum and maximum
collection length are out of range */
    const TpInt32 P_GUIS_INVALID_COLLECTION_CRITERIA = 772; /* Invalid collection criteria specified
*/
    const TpInt32 P_GUIS_INVALID_NETWORK_STATE = 773;      /* Although the sequence of method calls
is allowed by the gateway, the underlying protocol can not support it. */
 const TpInt32 P_GUIS_UNEXPECTED_SEQUENCE = 774;        /* Although the sequence of method calls is
allowed by the gateway, the underlying protocol can not support it. */

   }; // end module ui
  }; // end module osa
 }; // end module threegpp
}; // end module org

#endif

// END file UI.idl
```

9.4.2 Generic User Interaction IDL

```
// source file: GUI.idl
// GUIS Interface description

#ifndef __OSA_UI_GUI_DEFINED
#define __OSA_UI_GUI_DEFINED

#include <UI.idl>
#include <ECC.idl>

module org {
 module threegpp {
  module osa {
   module ui {
    module gui {

  interface IpAppUIManager; // forward definition;
  interface IpAppUI;        // forward definition;
  interface IpAppUICall;    // forward definition;

  /* The Generic User Interaction SCF Interface provides functions to send
     information to, or gather information from the user. */
  interface IpUI : IpService {
    /* This method plays an announcement or sends other information to the user.*/
    void sendInfoReq (
      in TpSessionID userInteractionSessionID,
      in TpUIInfo info,
      in TpUIVariableInfoSet variableInfo,
      in TpInt32 repeatIndicator,
      in TpUIResponseRequest responseRequested,
      out TpAssignmentID assignmentID
    )
```

```
    raises (TpGUISException, TpGeneralException);

    /* This method plays an announcement or sends other information to the user
       and collects some information from the user.  */
    void sendInfoAndCollectReq (
      in TpSessionID userInteractionSessionID,
      in TpUIInfo info,
      in TpUIVariableInfoSet variableInfo,
      in TpUICollectCriteria criteria,
      in TpUIResponseRequest responseRequested,
      out TpAssignmentID assignmentID
    )
    raises (TpGUISException, TpGeneralException);

    /* This method requests that the relationship between the application and
       the user interaction object be released. */
    void release (
      in TpSessionID userInteractionSessionID
    )
    raises (TpGUISException, TpGeneralException);

  };

  /* Defines the Sequence of Data Elements that unambiguously specify the UI object */
  struct TpUIIdentifier {
    TpSessionID UserInteractionSessionID;
    IpUI UIRef;
  };


  /* The Call User Interaction Service Interface provides functions to send
     information to, or gather information from,  the user. */
  interface IpUICall : IpUI {
    /* This asynchronous method aborts the specified user interaction operation. */
    void abortActionReq (
      in TpSessionID userInteractionSessionID,
      in TpAssignmentID assignmentID
    )
    raises (TpGUISException, TpGeneralException);

    /* The next operation is not supported for Release 99 and must
       return the exception “Method not supported” when invoked on a SCF
       implementation based on this specification: */
    void recordMessageReq (
      in TpSessionID userInteractionSessionID,
      in TpUIInfo info,
      in TpUIMessageCriteria criteria,
      out TpAssignmentID assignmentID
    )
    raises (TpGUISException, TpGeneralException);

  };

  /* Defines the Sequence of Data Elements that unambiguously specify the UICall object. */
  struct TpUICallIdentifier {
    IpUICall UICallRef;
    TpSessionID UserInteractionSessionID;
  };

  /* This interface is the 'SCF manager' interface for the Generic User Interaction SCF. */
  interface IpUIManager : IpService {
    /* This method is used to create a new user interaction object for non-call related purposes */
    void createUI (
      in IpAppUI appUI,
      in TpAddress userAddress,
      out TpUIIdentifier userInteraction
    )
    raises (TpGUISException, TpGeneralException);

    /* This method is used to create a new user interaction object for call related purposes. */
    void createUICall (
      in IpAppUICall appUI,
      in cc::gcc::TpCallIdentifier callIdentifier,
      in cc::ecc::TpCallLegIdentifier callLegIdentifier,
      out TpUICallIdentifier userInteraction
    )
    raises (TpGUISException, TpGeneralException);

    /* This method is used to enable the reception of user initiated user interaction. */
    void enableUINotification (
      in IpAppUIManager appInterface,
      in TpUIEventCriteria eventCriteria,
      out TpAssignmentID assignmentID
    )
    raises (TpGUISException, TpGeneralException);

    /* This method is used by the application to disable UI notifications. */
    void disableUINotification (
      in TpAssignmentID assignmentID
```

```
    )
    raises (TpGUISException, TpGeneralException);
  };


  /* The Generic User Interaction SCF manager application interface provides
     the application call management functions to the Generic User Interaction SCF. */
  interface IpAppUIManager : IpOsa {
    /* This method indicates to the application that the User Interaction SCF
       instance has terminated or closed abnormally.  */
    void userInteractionAborted (
      in TpUIIdentifier userInteraction
    )
    raises (TpGUISException, TpGeneralException);

    /* This method notifies the application of an user initiated request for user interaction. */
    void userInteractionEventNotify (
      in TpUIIdentifier ui,
      in TpUIEventInfo eventInfo,
      in TpAssignmentID assignmentID,
      out IpAppUI appInterface
    )
    raises (TpGUISException, TpGeneralException);

    void userInteractionNotificationInterrupted ()
    raises (TpGUISException, TpGeneralException);

    void userInteractionNotificationContinued ()
    raises (TpGUISException, TpGeneralException);

  };

  /* The User Interaction Application Interface is used to handle generic user
     interaction request responses and reports. */
  interface IpAppUI : IpOsa {
    /* This method informs the application about the start or the completion of a sendInfoCallReq().
*/
    void sendInfoRes (
      in TpSessionID userInteractionSessionID,
      in TpAssignmentID assignmentID,
      in TpUIReport response
    )
    raises (TpGUISException, TpGeneralException);

    /* This asynchronous method indicates that the request to send information was unsuccessful. */
    void sendInfoErr (
      in TpSessionID userInteractionSessionID,
      in TpAssignmentID assignmentID,
      in TpUIError error
    )
    raises (TpGUISException, TpGeneralException);

    /* This asynchronous method returns the information collected to the application. */
    void sendInfoAndCollectRes (
      in TpSessionID userInteractionSessionID,
      in TpAssignmentID assignmentID,
      in TpUIReport response,
      in TpString info
    )
    raises (TpGUISException, TpGeneralException);

    /* This asynchronous method indicates that the request to send information
       and collect a response was unsuccessful. */
    void sendInfoAndCollectErr (
      in TpSessionID userInteractionSessionID,
      in TpAssignmentID assignmentID,
      in TpUIError error
    )
    raises (TpGUISException, TpGeneralException);

    /* This method indicates to the application that a fault has been detected in the user
interaction. */
    void userInteractionFaultDetected (
      in TpSessionID userInteractionSessionID,
      in TpUIFault fault
    )
    raises (TpGUISException, TpGeneralException);
  };


  /* The Call User Interaction Application Interface is used to handle call user
     interaction request responses and reports. */
  interface IpAppUICall : IpAppUI {
    /* This method confirms that the request to abort a user interaction operation on a call was
successful. */
    void abortActionRes (
      in TpSessionID userInteractionSessionID,
      in TpAssignmentID assignmentID
    )
```

```
    raises (TpGUISException, TpGeneralException);

    /* This asynchronous method indicates that the request to abort a user interaction
       operation on a call resulted in an error.*/
    void abortActionErr (
      in TpSessionID userInteractionSessionID,
      in TpAssignmentID assignmentID,
      in TpUIError error
    )
    raises (TpGUISException, TpGeneralException);
    /* The next operation is not supported for Release 99 and must
       return the exception “Method not supported” when invoked on a SCF
       implementation based on this specification: */
    void recordMessageRes (
      in TpSessionID userInteractionSessionID,
      in TpAssignmentID assignmentID,
      in TpUIReport response,
      in TpInt32 messageID
    )
    raises (TpGUISException, TpGeneralException);

    /* The next operation is not supported for Release 99 and must
       return the exception “Method not supported” when invoked on a SCF
       implementation based on this specification: */
    void recordMessageErr (
      in TpSessionID userInteractionSessionID,
      in TpAssignmentID assignmentID,
      in TpUIError error
    )
    raises (TpGUISException, TpGeneralException);

  };


    }; // end module gui
   }; // end module ui
  }; // end module osa
 }; // end module threegpp
}; // end module org


#endif

// END file GUI.idl
```

9.5 Data Session Control

```
// OSA data session control

#ifndef __OSA_DSC_DEFINED
#define __OSA_DSC_DEFINED

#include "osa.idl"

module org
{
    module threegpp
    {
        module osa
        {


            // data session control
            module dsc
            {


                interface IpDataSessionControlManager; // forward definition
                interface IpDataSession;               // forward definition
                interface IpAppDataSessionControlManager; // forward definition
                interface IpAppDataSession;               // forward definition



                const TpInt32 P_EVENT_NAME_UNDEFINED = 0;                    // Undefined
                const TpInt32 P_EVENT_DSCS_ESTABLISHED_ = 1;                 // Data Session
established

                typedef TpInt32 TpDataSessionEventName; /*Defines the names of event being notified.
*/

                enum TpDataSessionChargeOrderCategory
                {
                    P_DATA_SESSION_CHARGE_PER_VOLUME,
                    P_DATA_SESSION_CHARGE_NETWORK
                };
```

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

                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;
                };

                struct TpDataSessionEventCriteria
                {
                    TpAddressRange DestinationAddress;       /*Destination address range*/
                    TpAddressRange OriginationAddress;       /*Origination address range */
                    TpDataSessionEventName DataSessionEventName;           /*Name of the event(s)
*/
                };



                    /* Defines the mode that the data session will monitor for events, or the mode
that the data session is in following a detected event. */
                enum TpDataSessionMonitorMode
                {
                    P_DATA_SESSION_MONITOR_MODE_INTERRUPT,     /* The data session event is
intercepted by the data session control SCF 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,        /* The data session event is detected
by the data session control SCF but not intercepted. The application is notified of the event data
session establishment continues */
                    P_DATA_SESSION_MONITOR_MODE_DO_NOT_MONITOR /* Do not monitor for the event */
                };

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

                    /* Defines the Sequence of Data Elements that specify the cause of the release
of a call.*/
                    struct TpDataSessionReleaseCause
                {
                    TpInt32 Value;
                    TpInt32 Location;
                };

                    /* Defines a specific data session error. */
                    enum TpDataSessionErrorType
                {
                    P_DATA_SESSION_ERROR_UNDEFINED,         /* Undefined */
                    P_DATA_SESSION_ERROR_INVALID_ADDRESS,   /* The operation failed because an
invalid address was given */
                    P_DATA_SESSION_ERROR_INVALID_STATE     /* The data session was not in a valid
state for the requested operation */
                };

                    /* Defines the Tagged Choice of Data Elements that specify additional data
session error and data session error specific information. */
                union TpDataSessionAdditionalErrorInfo switch(TpDataSessionErrorType)
                {
                    case P_DATA_SESSION_ERROR_INVALID_ADDRESS: TpAddressError
DataSessionErrorInvalidAddress;
                };

                    /* Defines the Sequence of Data Elements that specify the additional information
relating to an undefined data session error. */
                struct TpDataSessionError
                {
                    TpDataSessionAdditionalErrorInfo AdditionalErrorInfo;
                    TpDataSessionErrorType ErrorType;
                    TpDateAndTime ErrorTime;
                };
```

```
                    /* Defines the cause of the Data Session fault detected. */
                enum TpDataSessionFault
                {
                    P_DATA_SESSION_FAULT_UNDEFINED,     /* Undefined */
                    P_DATA_SESSION_FAULT_USER_ABORTED,  /* User has finalised the data session
before any message could be sent by the application. */
                    P_DATA_SESSION_TIMEOUT_ON_RELEASE,  /* Final report has been sent to the
application, but the application did not explicitly release data session object, within a specified
time. */
                    P_DATA_SESSION_TIMEOUT_ON_INTERRUPT /* Application did not instruct the gateway
how to handle the data session within a specified time, after the gateway reported an event that was
requested by the application in interrupt mode.*/
                };


                /* Defines a specific data session event report type. */
                enum TpDataSessionReportType
                {
                    P_DATA_SESSION_REPORT_UNDEFINED,       /* Undefined */
                    P_DATA_SESSION_REPORT_CONNECTED,          /* Data session established*/
                    P_DATA_SESSION_REPORT_DISCONNECT      /* data session disconnect requested by
data session party */
                };

                /* Defines the Tagged Choice of Data Elements that specify additional data session
report information. */
                union TpDataSessionAdditionalReportInfo switch(TpDataSessionReportType)
                {
                    case P_DATA_SESSION_REPORT_DISCONNECT: TpDataSessionReleaseCause
DataSessionDisconnect;
                };

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


                    /* Defines the Sequence of Data Elements that specify the criteria relating to
Data Session report requests. */
                struct TpDataSessionReportRequest
                {
                    TpDataSessionMonitorMode MonitorMode;
                    TpDataSessionReportType DataSessionReportType;
                };

                /* Defines a Numbered Set of Data Elements of TpDataSessionReportRequest. */
                typedef sequence <TpDataSessionReportRequest> TpDataSessionReportRequestSet;

                const TpInt32 P_DATA_SESSION_SUPERVISE_VOLUME_REACHED = 1;      /* The Data Session
supervision volume has been reached. */
                const TpInt32 P_DATA_SESSION_SUPERVISE_DATA_SESSION_ENDED = 2;   /* The data session
has ended, either due to reach of maximum volume or calling or called party release. */
                const TpInt32 P_DATA_SESSION_SUPERVISE_MESSAGE_SENT = 4; /* A warning message has
been sent. */

                /* Defines the responses from the data session control SCF for data sessions that
are supervised:*/
                typedef TpInt32 TpDataSessionSuperviseReport;

                const TpInt32 P_DATA_SESSION_SUPERVISE_RELEASE = 1;      /* Release the Data Session
when the Data Session supervision volume has been reached. */
                const TpInt32 P_DATA_SESSION_SUPERVISE_RESPOND = 2;      /* Notify the application
when the data session supervision volume has been reached. */
                const TpInt32 P_DATA_SESSION_SUPERVISE_INFORM = 4;   /* 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 */
                /* Defines the following treatment of the data session by the data session control
SCF when the maximum volume has been reached.*/
                typedef TpInt32 TpDataSessionSuperviseTreatment;

                /* Defines the Sequence of Data Elements that specify the amount of volume that is
allowed to be transmitted for the specific connection. */
                struct TpDataSessionSuperviseVolume {
                TpInt32 VolumeQuantity; /* Qantity of the granted volume that can be transmitted for
the specific connection. */
                TpInt32 VolumeUnit;     /* Unit of the granted volume that can be transmitted for
the specific connection. */
                };

                /* Define the possible Exceptions. */
                const TpInt32 P_DSCS_SERVICE_INFORMATION_MISSING = 1024;
                const TpInt32 P_DSCS_SERVICE_FAULT_ENCOUNTERED = 1025;
                const TpInt32 P_DSCS_UNEXPECTED_SEQUENCE = 1026;
                const TpInt32 P_DSCS_INVALID_ADDDRESS = 1027;
                const TpInt32 P_DSCS_INVALID_STATE = 1028;
```

```
const TpInt32 P_DSCS_INVALID_CRITERIA = 1029;
const TpInt32 P_DSCS_INVALID_NETWORK_STATE = 1030;

exception TpDSCSException
{
    TpInt32 exceptionType;
};

/* Sequence of Data Elements that unambiguously specify the Data Session object */
struct TpDataSessionIdentifier
{
    IpDataSession DataSessionReference;
    TpSessionID DataSessionSessionID;
};



/* This interface is the SCF manager' interface for Data Session Control. */
interface IpDataSessionControlManager : IpService
{
    /* This method is used to enable data session notifications. */
    void enableDataSessionNotification (
    in IpAppDataSessionControlManager appInterface,
    in TpDataSessionEventCriteria eventCriteria,
    out TpAssignmentID assignmentID)
    raises (TpDSCSException, TpGeneralException);

/* This method is used by the application to disable data session notifications.*/
void disableDataSessionNotification
(
    in TpAssignmentID assignmentID)
    raises (TpDSCSException, TpGeneralException);
};

/* This interface provides the means to control a data session. */
interface IpDataSession : IpService
{
    /* This method requests connection of the data session to the destination
party.*/
    void connectReq (
    in TpSessionID dataSessionID,
    in TpDataSessionReportRequestSet responseRequested,
    in TpAddress targetAddress,
    out TpAssignmentID assignmentID)
    raises (TpDSCSException, TpGeneralException);

    /* This method requests the release of the data session and associated
objects.*/
    void release (
        in TpSessionID dataSessionID,
        in TpDataSessionReleaseCause cause)
        raises (TpDSCSException, TpGeneralException);


    /* The application calls this method to supervise a data session.  */
    void superviseDataSessionReq (
        in TpSessionID dataSessionID,
        in TpDataSessionSuperviseTreatment treatment,
        in TpDataSessionSuperviseVolume bytes)
        raises (TpDSCSException, TpGeneralException);


    /* The application calls this method to set the charge plan */
    void setDataSessionChargePlan (
        in TpSessionID dataSessionID,
        in TpDataSessionChargePlan dataSessionChargePlan)
        raises (TpDSCSException, TpGeneralException);


    /* The application calls this method to send advice of charge information */
    void setAdviceOfCharge (
        in TpSessionID dataSessionID,
        in TpAoCInfo aoCInfo,
        in TpDuration tariffSwitch)
        raises (TpDSCSException, TpGeneralException);
};



/* The data session control manager application interface provides the
application data session control management functions to the data session control
SCF. */
interface IpAppDataSessionControlManager : IpOsa
{
    void dataSessionAborted (
        in TpSessionID dataSessionReference)
```

```
                        raises (TpDSCSException, TpGeneralException);

                    /* This method notifies the application of the arrival of a data session-related
event. */
                    void dataSessionEventNotify (
                        in TpDataSessionIdentifier dataSessionReference,
                        in TpDataSessionEventInfo eventInfo,
                        in TpAssignmentID assignmentID,
                        out IpAppDataSession appInterface)
                        raises (TpDSCSException, TpGeneralException);

                    /* This method indicates to the application that all event notifications
                    are resumed.*/
                    void dataSessionNotificationContinued()
                        raises (TpDSCSException, TpGeneralException);

                    /* This method indicates to the application that all event notifications
                    are temporarely iterrupted.*/
                    void dataSessionNotificationInterrupted()
                        raises (TpDSCSException, TpGeneralException);


                };



                /* The application side of the data session interface is used to handle data session
                request responses and state reports. */
                interface IpAppDataSession : IpOsa
                {
                    /* This method indicates that the request to route the data session to the
                    destination was successful.*/
                    void connectRes (
                        in TpSessionID dataSessionSessionID,
                        in TpDataSessionReport eventReport,
                        in TpAssignmentID assignmentID)
                        raises (TpDSCSException, TpGeneralException);

                    /* This method indicates that the request to connect the data session to the
                    destination party was unsuccessful. */
                    void connectErr (
                        in TpSessionID dataSessionSessionID,
                        in TpDataSessionError errorIndication,
                        in TpAssignmentID assignmentID)
                        raises (TpDSCSException, TpGeneralException);


                    /* This asynchronous method reports a data session supervision event to the
application.*/
                    void superviseDataSessionRes (
                        in TpSessionID dataSessionSessionID,
                        in TpDataSessionSuperviseReport report,
                        in TpDataSessionSuperviseVolume usedVolume)
                        raises (TpDSCSException, TpGeneralException);

                    /* This asynchronous method reports a data session supervision error to the
application.*/
                    void superviseDataSessionErr (
                        in TpSessionID dataSessionSessionID,
                        in TpDataSessionError errorIndication)
                        raises (TpDSCSException, TpGeneralException);

                    /* This method indicates to the application that a fault in the network has
                    been detected.*/
                    void dataSessionFaultDetected (
                        in TpSessionID dataSessionSessionID,
                        in TpDataSessionFault fault)
                        raises (TpDSCSException, TpGeneralException);

                };

            }; // end module dsc

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

        }; // osa
    }; // threegpp
};

#endif
```

9.6 Network User Location and User Status IDL

9.6.1 Common definitions for Network User Location and User Status: MM.idl

```
#include <OSA.idl>

module org {
module threegpp {
module osa {
module mm {

   // Defines the type of uncertainty shape.
    enum TpLocationUncertaintyShape {
        P_M_SHAPE_NONE,   // No uncertainty shape present.
        P_M_SHAPE_CIRCLE, // Uncertainty shape is a circle.
        P_M_SHAPEa_CIRCLE_SECTOR,    // Uncertainty shape is a circle sector.
        P_M_SHAPE_CIRCLE_ARC_STRIPE, // Uncertainty shape is a circle arc stripe.
        P_M_SHAPE_ELLIPSE,           // Uncertainty shape is an ellipse.
        P_M_SHAPE_ELLIPSE_SECTOR,    // Uncertainty shape is an ellipse sector.
        P_M_SHAPE_ELLIPSE_ARC_STRIPE // Uncertainty shape is an ellipse arc stripe.
    };

    // Defines the structure of data elements that specify a geographical position.
    // An “ellipsoid point with uncertainty shape” defines the horizontal location.
    // The reference system chosen for the coding of locations is the World Geodetic
    // System 1984 (WGS 84).
    struct TpGeographicalPosition {
        TpFloat Longitude;
        TpFloat Latitude;
        TpLocationUncertaintyShape TypeOfUncertaintyShape;
        TpFloat UncertaintyInnerSemiMajor;
        TpFloat UncertaintyOuterSemiMajor;
        TpFloat UncertaintyInnerSemiMinor;
        TpFloat UncertaintyOuterSemiMinor;
        TpInt32 AngleOfSemiMajor;
        TpInt32 SegmentStartAngle;
        TpInt32 SegmentEndAngle;
    };




// Defines a diagnostic value that is reported in addition to an error by
    // the Network User Location or User Status SCFs.
    enum TpMobilityDiagnostic {
        P_M_NO_INFORMATION,     // No diagnostic information present.
                                // Valid for all type of errors.
        P_M_APPL_NOT_IN_PRIV_EXCEPT_LST, // Application not in privacy exception list.
                                         // Valid for 'Unauthorised Application' error.
        P_M_CALL_TO_USER_NOT_SETUP,      // Call to user not set-up. Valid for
                                         // 'Unauthorised Application' error.
        P_M_PRIVACY_OVERRIDE_NOT_APPLIC, // Privacy override not applicable. Valid for
                                         // 'Unauthorised Application' error.
        P_M_DISALL_BY_LOCAL_REGULAT_REQ, // Disallowed by local regulatory requirements.
                                         // Valid for 'Unauthorised Application' error.
        P_M_CONGESTION,                  // Congestion. Valid for 'Position Method
                                         // Failure' error.
        P_M_INSUFFICIENT_RESOURCES, // Insufficient resources. Valid for 'Position
                                    // Method Failure' error.
        P_M_INSUFFICIENT_MEAS_DATA, // Insufficient measurement data. Valid for
                                    // 'Position Method Failure' error.
        P_M_INCONSISTENT_MEAS_DATA, // Inconsistent measurement data. Valid for
                                    // 'Position Method Failure' error.
        P_M_LOC_PROC_NOT_COMPLETED, // Location procedure not completed. Valid for
                                    // 'Position Method Failure' error.
        P_M_LOC_PROC_NOT_SUPP_BY_USER, // Location procedure not supported by user.
                                       // Valid for 'Position Method Failure' error.
        P_M_QOS_NOT_ATTAINABLE  // Quality of service not attainable. Valid for
                                // 'Position Method Failure' error.
    };

    // Defines an error that is reported by  the Network User Location or User Status SCFs.
    enum TpMobilityError {
        P_M_OK,             // No error occurred while processing the request.
        P_M_SYSTEM_FAILURE, // System failure. The request can not be handled because
                            // of a general problem in the network user location SCF
                            // , the user status SCFor the
                            // underlying network.    Fatal
        P_M_UNAUTHORIZED_NETWORK, // Unauthorised network, The requesting network is
                                  // not authorised to obtain the user's location or
                                  // status. Non fatal
        P_M_UNAUTHORIZED_APPLICATION, // Unauthorised application. The application is
```

```
                                    // not authorised to obtain the user's location
                                    // or status. Fatal
        P_M_UNKNOWN_SUBSCRIBER, // Unknown subscriber. The user is unknown, i.e. no
                                // such subscription exists. Fatal
        P_M_ABSENT_SUBSCRIBER,  // Absent subscriber. The user is currently not
                                // reachable. Non fatal
        P_M_POSITION_METHOD_FAILURE // Position method failure. The network user location SCF
                                   // failed to obtain the user's position. Non fatal
    };

    // This enumeration is used in requests to stop network user location or user status
    // reports that are
    // sent from a network user location or user status SCFs to an application.
    enum TpMobilityStopScope {
        P_M_ALL_IN_ASSIGNMENT, // The request concerns all users in an assignment.
        P_M_SPECIFIED_USERS    // The request concerns only the users that are
                               // explicitly specified in a collection.
    };

    // Defines the structure of data element that specifies a request to stop whole or parts of an
    // assignment. Assignments are used for periodic or triggered reporting of a
    // user locations or statuses. Observe that the parameter 'Users' is optional.
    // If the parameter 'stopScope' is set to P_M_ALL_IN_ASSIGNMENT, the parameter
    // 'stopScope' is undefined. If the parameter stopScope is set to
    // P_M_SPECIFIED_USERS, then the assignment shall be stopped only for the users
    // specified in the 'users' collection.
    struct TpMobilityStopAssignmentData {
        // Identity of the session that shall be stopped.
        TpSessionID          AssignmentId;
        // Specify if only a part of the assignment or if whole the assignment
        // shall be stopped.
        TpMobilityStopScope  StopScope;
        // Optional parameter describing which users a stop request is
        // addressing when only a part of an assignment is to be stopped.
        TpAddressSet         Users;
    };

}; }; }; };
```

9.6.2 Network User Location: MMul.idl

```
/************************************************************************/
// Data Definitions & Interfaces
// Network User Location
/************************************************************************/

#include <MM.idl>

module org {
module threegpp {
module osa {
module mm {
module ul {

    /************************************************************************/
    //                          Data definitions
    /************************************************************************/

    // This data type is identical to a TString. It specifies the Cell Global
    // Identification or the Location Area Identification (LAI).
    // The Cell Global Identification (CGI) is defined as the 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 for
    // the detailed coding.

    typedef TpString TpLocationCellIDOrLAI;

    // Defines the structure of data elements that specifies the criteria for a
    // triggered location report to be generated.
    struct TpLocationTriggerCamel {
        TpBoolean UpdateInsideVlr; // Generate location report when it occurs an
                                   // location update inside the current VLR area.
        TpBoolean UpdateOutsideVlr;// Generate location report when the user moves
                                   // to another VLR area.
```

```
    };

    // Defines the structure of data elements that specifies the location of a mobile
    // telephony user. Observe that if the StatusCode is indicating an error ,
    // then neither GeographicalPosition, Timestamp, VlrNumber, LocationNumber,
    // CellIdOrLai nor their associated presense flags are defined.
    struct TpUserLocationCamel {
        TpAddress       UserID;     // The address of the user.
        TpMobilityError StatusCode; // Indicator of error.
        TpBoolean       GeographicalPositionPresent; // Flag indicating if the
                                                     // geographical position is present.
        TpGeographicalPosition GeographicalPosition; // Specification of a position
                                                     // and an area of uncertainty.
        TpBoolean       TimestampPresent; // Flag indicating if the timestamp is present.
        TpDateAndTime   Timestamp;        // Timestamp indicating when the location information//
was attained
        TpBoolean       VlrNumberPresent; // Flag indicating if the VLR number is present.
        TpAddress       VlrNumber;        // Current VLR number for the user.
        TpBoolean       LocationNumberPresent; // Flag indicating if the location
                                               // number is present.
        TpAddress       LocationNumber;        // Current location number.
        TpBoolean       CellIdOrLaiPresent;    // Flag indicating if cell-id or
                                               // LAI of the user is present.
        TpLocationCellIDOrLAI CellIdOrLai;     // Cell-id or LAI of the user.
    };

    typedef sequence <TpUserLocationCamel> TpUserLocationCamelSet;

    /************************************************************************/
    //                        Interface definitions
    /************************************************************************/

    interface IpAppUserLocationCamel; // Forward definition

    // Inherits from the generic service capability feature interface.
    // This interface is the SCF manager's interface for Network User Location.
    interface IpUserLocationCamel : IpService {

        // Request for mobile-related location information on one or several wireles users.
        void locationReportReq(
             in  IpAppUserLocationCamel appLocationCamel,
             in  TpAddressSet           users,
             out TpSessionID            assignmentId)
                raises (TpGeneralException);

        // Request for periodic mobile location reports on one or several users.
        void periodicLocationReportingStartReq(
             in  IpAppUserLocationCamel appLocationCamel,
             in  TpAddressSet           users,
             in  TpDuration             reportingInterval,
             out TpSessionID            assignmentId)
                raises (TpGeneralException);

        // This method stops the sending of periodic mobile location reports for
        // one or several users.
        void periodicLocationReportingStop(
             in  TpMobilityStopAssignmentData stopRequest)
                raises (TpGeneralException);

        // Request for user location reports, containing mobile related information,
        // when the location is changed (the report is triggered by the location change).
        void triggeredLocationReportingStartReq(
             in  IpAppUserLocationCamel   appLocationCamel,
             in  TpAddressSet             users,
             in  TpLocationTriggerCamel   trigger,
             out TpSessionID              assignmentId)
                raises (TpGeneralException);

        // Request that triggered mobile location reporting should stop.
        void triggeredLocationReportingStop(
             in  TpMobilityStopAssignmentData stopRequest)
                raises (TpGeneralException);
    };

    // Inherits from the generic service capability feature interface.
    // The network user location application interface is implemented by the client
    // application developer and is used to handle location reports that are
    // specific for mobile telephony users.
    interface IpAppUserLocationCamel : IpOsa {

        // Delivery of a mobile location report. The report is containing
        // mobile-related location information for one or several users.
        void locationReportRes(
             in TpSessionID              assignmentId,
             in TpUserLocationCamelSet   locations)
                raises (TpGeneralException);

        // This method indicates that the location report request has failed.
        void locationReportErr(
```

```
            in TpSessionID            assignmentId,
            in TpMobilityError        cause,
            in TpMobilityDiagnostic   diagnostic);

        // Periodic delivery of mobile location reports. The reports are
        // containing mobile-related location information for one or several users.
        void periodicLocationReport(
            in TpSessionID              assignmentId,
            in TpUserLocationCamelSet   locations)
               raises (TpGeneralException);

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

        // Delivery of a report that is indicating that one or several user's
        // mobile location has changed.
        void triggeredLocationReport(
            in TpSessionID              assignmentId,
            in TpUserLocationCamel      location,
            in TpLocationTriggerCamel   criterion)
               raises (TpGeneralException);

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

};};};};};
```

9.6.3 User Status: MMus.idl

```
/************************************************************************/
//  Data Definitions & Interfaces
// User Status
/************************************************************************/

#include <MM.idl>

module org {
module threegpp {
module osa {
module mm {
module us {

    /************************************************************************/
    //                          Data definitions
    /************************************************************************/

    // Defines the status of a user.
    enum TpUserStatusIndicator {
        P_US_REACHABLE,     // User is reachable
        P_US_NOT_REACHABLE, // User is not reachable
        P_US_BUSY           // User is busy (only applicable for interactive user
                            // status request, not when triggers are used)
    };

    // Defines the structure of data elements  that specify the identity
    // and status of a user.
    struct TpUserStatus {
        TpAddress             UserID;       // The user address.
        TpMobilityError     StatusCode; // Indicator of error.
        TpUserStatusIndicator Status;       // The current status of the user.
};

    typedef sequence <TpUserStatus> TpUserStatusSet;

    /************************************************************************/
    //                      Interface definitions
    /************************************************************************/

    interface IpAppUserStatus; // Forward definition

    // Inherits from the generic service capability feature interface.
    // The user status interface represents the interface to the user status SCF.
    interface IpUserStatus : IpService {

        // Request for a report on the status of one or several users.
        void statusReportReq(
            in  IpAppUserStatus    appStatus,
```

```
            in  TpAddressSet       users,
            out TpSessionID        assignmentId)
               raises (TpGeneralException);

        // Request for triggered status reports when one or several user's
        // status is changed. The user status SCF will send a report when
        // the status changes.
        void triggeredStatusReportingStartReq (
            in  IpAppUserStatus    appStatus,
            in  TpAddressSet       users,
            out TpSessionID        assignmentId)
               raises (TpGeneralException);

        // This method stops the sending of status reports for one or several users.
        void triggeredStatusReportingStop (
            in TpMobilityStopAssignmentData stopRequest)
               raises (TpGeneralException);
    };

    // Inherits from the base osa interface.
    // The user-status application interface is implemented by the client
    // application developer and is used to handle user status reports.
    interface IpAppUserStatus : IpOsa {

        // Delivery of a report, that is containing one or several user's status.
        void statusReportRes(
            in TpSessionID     assignmentId,
            in TpUserStatusSet status)
               raises (TpGeneralException);

        // This method indicates that the status report request has failed.
        void statusReportErr(
            in TpSessionID          assignmentId,
            in TpMobilityError      cause,
            in TpMobilityDiagnostic diagnostic);

        // Delivery of a report that is indicating that a user's status has changed.
        void triggeredStatusReport(
            in TpSessionID          assignmentId,
            in TpUserStatus         status)
               raises (TpGeneralException);

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

};};};};};
```

9.7 Terminal Capabilities: TERMCAP.idl

```
#ifndef __TERMCAP_DEFINED
#define __TERMCAP_DEFINED

#include <OSA.idl>

module org {
module threegpp {
module osa {
module termcap {

   enum TpTerminalCapabilitiesError {
        P_TERMCAP_ERROR_UNDEFINED,      /* Undefined */
        P_TERMCAP_INVALID_TERMINALID,       /* Terminal ID not valid */
        P_TERMCAP_SYSTEM_FAILURE        /* General problem in terminal capabilities SCF or in
underlying network */

      };

     exception TpTermCapException {
         TpTerminalCapabilitiesError error;
   };

    /* TpTerminalCapabilities: Structure containing status code and terminal
    capabilities. */
    struct TpTerminalCapabilities {
        /* statusCode: Indicates whether or not the terminalCapabilities
        are available. */
        TpBoolean StatusCode;
        /* terminalCapabilities: Specifies the latest available capabilities of the user´s terminal.
```

```
This information, if available, is returned as CC/PP headers as specified in W3C [6] and adopted in
the WAP UAProf specification [9]. It contains URLs; terminal attributes and values, in RDF format;
or a combination of both.  */
        TpString TerminalCapabilities;
    };

    interface IpTerminalCapabilities : IpService {
        /*  Method: getTerminalCapabilities()
            This method is used by an application to get the capabilities of a
            user's terminal. Direction: Application to Network

            In parameter TerminalIdentity: Identifies the terminal. It may be
            a logical address known by the WAP Gateway/PushProxy.
            Out parameter, see TerminalCapabilityStruct*/
        void getTerminalCapabilities (
            in TpString terminalIdentity,
            out TpTerminalCapabilities result
            )
         raises (TpTermCapException, TpGeneralException);

    };


};};};};

#endif
```

Annex A (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
Jun 2000	CN_08	NP-000310			Approval of Specification	2.0.0	3.0.0
Sep 2000	CN_09	NP-000519	001	1	Improvement of User Interaction STDs	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	003	2	Correction of numbering in TpResultInfo	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	004	1	Remove of E.164 Mobile and correction of numbering in TpAddressPlan	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	005		Common IDL interfaces for Generic Call Control and Generic User Interaction between 3GPP, ETSI SPAN3 and Parlay	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	006		Correction to table with overview of IDL files	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	007		Reduction in name scoping in IDL for createUICall operation on IpUICall interface	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	008	2	Alignment of Framework with Parlay 2.1, improvement on business entity identification	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	009	2	Alignment of Framework with Parlay 2.1, correction of missing service token	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	010	2	Alignment of Framework with Parlay 2.1, parameter name and data-type alignments	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	011	1	Alignment of Framework with Parlay 2.1, one interface per application correction	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	012	1	Alignment of Framework with Parlay 2.1, only one error returned in load manager query	3.0.0	3.1.0
Sep 2000	CN_09	NP-000519	013	1	Alignment of Framework with Parlay 2.1, missing operation fwUnavailableInd in IpAppFaultManager.	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	014	1	Alignment of Framework with Parlay 2.1, missing service properties parameter in getServiceManager() operation of IpSvcFactory.	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	015	1	Alignment of Framework with Parlay 2.1 undefined datatype in endaccess operation of IpAccess.	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	016	1	Alignment of Framework with Parlay 2.1, service and interface naming correction.	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	017	1	Alignment of Framework with Parlay 2.1, renaming of TpPropertyStruct to TpServiceTypeProperty	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	018	1	Alignment of Framework with Parlay 2.1 addition of DES 128 bit authentication.	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	019	2	Alignment of Framework with Parlay 2.1, improvement of load statistic data-types.	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	020	1	Correction in descriptive text for Call STD regarding user interaction in 2 Parties in Call State.	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	021		"Removal of double description of the type TpCallServiceCode".	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	022	1	"Removal of unused types TpUIMessageCriteria, TpEntOpID and TpEntOpIDList".	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	023		Alignment of Framework with Parlay 2.1, addition of setCallbackWithSessionID operation to IpService.	3.0.0	3.1.0
Sep 2000	CN_09	NP-000520	024		Clarification of life time of parameters in TpAuthDomain	3.0.0	3.1.0
Dec 2000	CN_10	NP-000718	025		Removal of the originatingAddress from the connectReq method in IpDataSession	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	026	1	Alignment between new ETSI document for common data and TS29.198	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	027		Correction of the type TpTerminalCapabilities	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	028		Incorrect Date and Time example in Data Definitions	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	029		Double IDL definition for TpGCCSException	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	030		Parameter EnabledOrDisbled in TpServiceTypeDescription	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	031		readonly is an IDL keyword	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	032		Error correction in the Scope definition, section 1	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	034		Specific exceptions for method invocations in invalid states	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	035		Unclear default value for TpAccessType	3.1.0	3.2.0

Dec 2000	CN_10	NP-000718	036	1	Unclear description for TpAuthType	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	037		TpInterfaceName in method obtainInterface()	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	038		Correction on numbering in TpCallAppInfoType	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	039		Addition of MonitorMode in TpCallEventInfo	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	040		Renaming of P_CALL_REPORT_REFUSED_BUSY	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	043		Removal of the parameter serviceProperties in the method selectService	3.1.0	3.2.0
Dec 2000	CN_10	NP-000718	044		Inclusion of missing state transitions in case call related information could not be retrieved.	3.1.0	3.2.0
Mar 2001	CN_11	NP-010133	045		Correction of IDL implementation of data-type TpDomainID	3.2.0	3.3.0
Mar 2001	CN_11	NP-010133	046		Correction to terminal capability parameter reference	3.2.0	3.3.0
Jun 2001	CN_12	NP-010325	048		IDL Correction of TpCallEventCriteria	3.3.0	3.4.0

History

Document history		
V3.0.0	June 2000	Publication
V3.1.0	September 2000	Publication
V3.2.0	December 2000	Publication
V3.3.0	March 2001	Publication
V3.4.0	June 2001	Publication